Universiteit
Leiden
The Netherlands

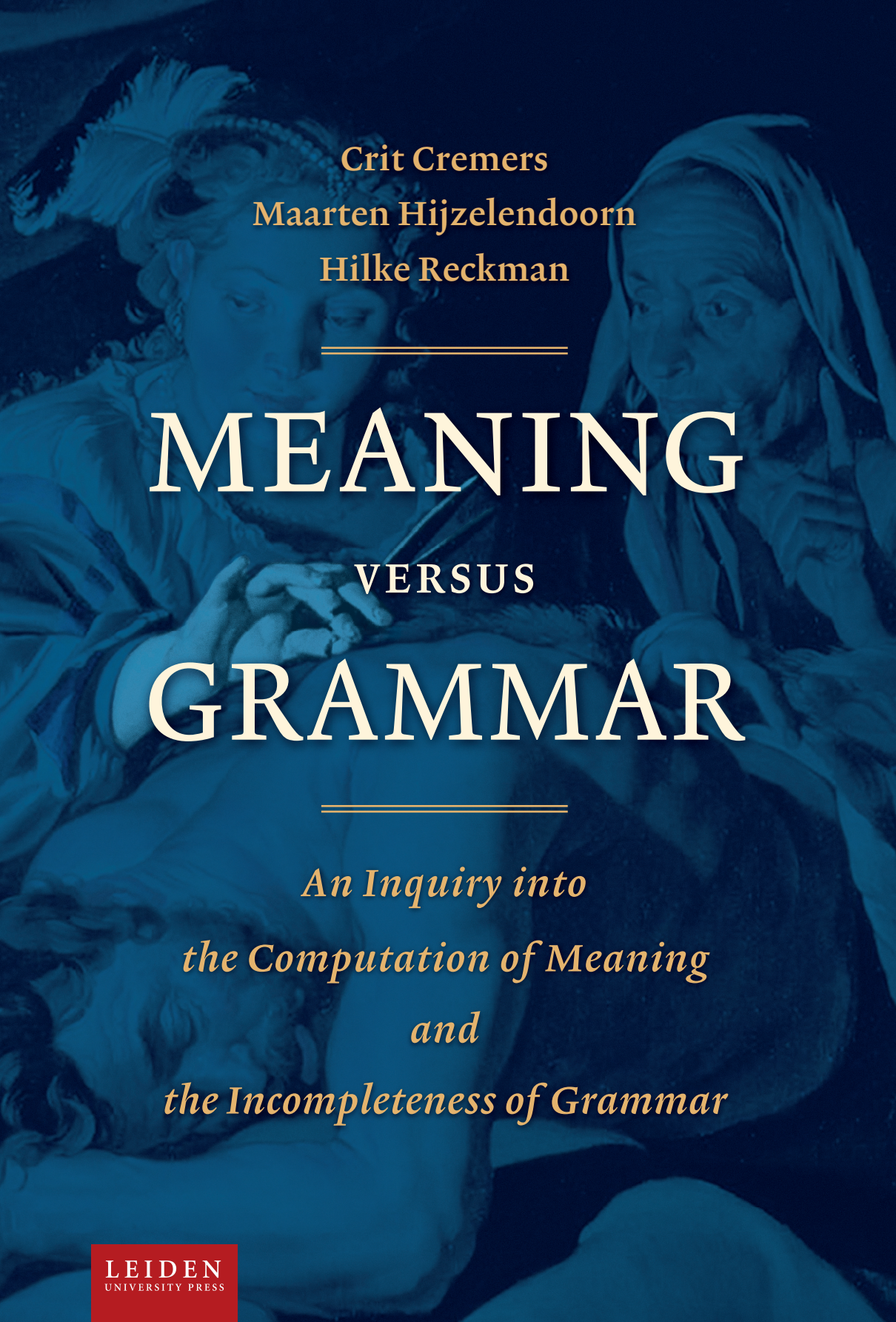
Meaning versus Grammar : an inquiry into the computation of meaning and the incompleteness of grammar

Cremers, Crit; Hijzelendoorn, Maarten; Reckman, Hilke

Citation
Cremers, C., Hijzelendoorn, M., & Reckman, H. (2014). *Meaning versus Grammar : an inquiry into the computation of meaning and the incompleteness of grammar*. Leiden University Press. Retrieved from https://hdl.handle.net/1887/33874

Version:	Not Applicable (or Unknown)
License:	Leiden University Non-exclusive license
Downloaded from:	https://hdl.handle.net/1887/33874

Note: To cite this publication please use the final published version (if applicable).



Crit Cremers
Maarten Hijzelendoorn
Hilke Reckman

MEANING VERSUS GRAMMAR

An Inquiry into the Computation of Meaning and the Incompleteness of Grammar

LEIDEN
UNIVERSITY PRESS

MEANING VERSUS GRAMMAR

MEANING VERSUS GRAMMAR

An Inquiry into the Computation of Meaning and the Incompleteness of Grammar

Crit Cremers
Maarten Hijzelendoorn
Hilke Reckman

Leiden University Press

Cover design: Kok Korpershoek
Lay-out: Jurgen Leemans

Cover illustration: Matthias Stom, *Samson and Delilah*, 1630s, Palazzo Barberini, Galleria Nazionale d'Arte Antica, Rome, Italy

ISBN 978 90 8728 212 7
e-ISBN 978 94 0060 183 3 (e-pdf)
e-ISBN 978 94 0060 184 0 (e-pub)
NUR 616

© Crit Cremers / Maarten Hijzelendoorn / Hilke Reckman / Leiden University Press, 2014

All rights reserved. Without limiting the rights under copyright reserved above, no part of this book may be reproduced, stored in or introduced into a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photocopying, recording or otherwise) without the written permission of both the copyright owner and the author of the book.

This book is distributed in North America by the University of Chicago Press (www.press.uchicago.edu).

TABLE OF CONTENTS

ACKNOWLEDGEMENTS 9

0. INTRODUCTION 11

0.1 A Language Machine 11

0.2 Language and computability 16

0.3 The book 21

1. SYNTAX: the game of recursion and discontinuity 25

1.1 The need for syntax 25

1.2 Forms of Dutch 30

1.3 The task for syntax 34

1.4 The logic and the algebra of lists, flags, types and modes 41

1.4.1 Categories and complex symbols 41

1.4.2 Basics 44

1.4.3 Merge 47

1.4.4 Modalities 49

1.4.5 Argument list 50

1.4.6 Deduction schemes 52

1.4.7 The algebra of strings 53

1.4.8 Disharmonic composition and categorial logic: grammar and reasoning 56

1.5 The calculi 60

1.5.1 Merge and Move 62

1.5.2 Soundness and completeness 63

1.5.3 The fundamental asymmetry of merge 68
1.5.4 No manipulation of structure 70

1.6 The case for Dutch *71*
1.6.1 General Format 71
1.6.2 A concise syntax of Dutch 74
1.6.3 Dealing with adjuncts 90

1.7 The grammar of discontinuity and coordination *94*
1.7.1 The source of discontinuity and connectedness 94
1.7.2 Discontinuity in CLG 95
1.7.3 Patterns of Dutch 102
1.7.4 Coordination as discontinuity 106

1.8 Parsing the syntax *114*
1.8.1 The syntax of CCG 115
1.8.2 Comparing the syntaxes of CCG and CLG 116
1.8.3 Comparing the generative capacity of CCG and CLG 120
1.8.4 CLG and the Chomsky hierarchy 124
1.8.5 Parsing CCGs 125
1.8.6 Parsing CLG 129
1.8.7 Extending and restricting a parser for CLG 130
1.8.8 Parsing coordination 136

1.9 Generating by syntax: agendas and linearization *138*
1.9.1 Two-directional grammar 138
1.9.2 Categories as agendas 145

2. SEMANTICS: the game of scope and intensionality 153

2.1 The ways of meaning *153*

2.2 The forms of meaning *157*

2.3 Scope and Specification *162*
2.3.1 Quantifiers 162
2.3.2 The structures of quantification 167
2.3.3 Compositionality and underspecification 170

2.4 Intensionality and semantic dependency *175*
2.4.1 Intensionality 175
2.4.2 Skolemization of dependent events 180

2.5 Events and states: reification of predication *182*
2.5.1 Reference to events 182
2.5.2 Pluractionality 190

2.6 Exploiting Logical Form for Parsing *196*
2.6.1 Logical Form, Grammar and Computation 196
2.6.2 Logical form in DELILAH 197
2.6.3 Stored Logical Form 198
2.6.4 Applied Logical Form 208
2.6.5 Flat Logical Form 210
2.7 Generating from Logic *217*
2.7.1 Generation as translation 217
2.7.2 From logical form to lexical line up 219
2.7.3 From lexical line-up to sentence: intertwining agendas 223
2.7.4 Testing logical form by entailment 226

3. LEXICON: the language's encyclopaedia and database 229

3.1 Storing knowledge of language *229*
3.1.1 Lexicalism: atoms and molecules 229
3.1.2 DELILAH and HPSG 234
3.1.3 Words and worlds: a lexicon is not a dictionary 237
3.1.4 Lexical chemistry: cooking the graphs 240
3.2 Modes of lexical knowledge *242*
3.2.1 Phonological form: the one-dimensional grammar 242
3.2.2 Morphology: the combinatoric guide 244
3.2.3 Syntax: the unification agenda 245
3.2.4 Semantics: ultimate knowledge of language 249
3.2.5 Information Structure: to the limits of decidability 258
3.3 Unification: powering grammar conservatively *259*
3.3.1 Procedures and specifications 259
3.3.2 Problems with re-entrance 264
3.4 The making of the lexicon *265*
3.4.1 Organizing lexical knowledge: no lemmas – economy vs flexibility 265
3.4.2 General and special: everything as a graph 270
3.4.3 The rules that make the language 272
3.4.4 The constructive lexicon 279
3.4.5 The lexicon is local 283
3.5 Disclosing the lexicon: object-orientation and speed for semantic generation *284*
3.5.1 The enterprise 284
3.5.2 Two models 286

3.5.3 Methods 292
3.5.4 Creating and accessing the object lexicon 294
3.5.5 Perspectives 297
3.6 The lexicon while parsing *298*

4. GRAMMAR: the reward of incompleteness 301

4.1 The three duals of grammar *301*
4.2 The conservativity of syntax *302*
4.2.1 The yield of syntax 302
4.2.2 The restrictedness of unification 303
4.2.3 The generality of unification 305
4.3 The destructivity of semantics *308*
4.3.1 Divorcing constituents and entailments 308
4.3.2 Entailments as products of composition and deduction 309
4.4 The denial of structure *314*
4.5 The mismatch of structure and meaning *316*
4.5.1 The mismatch in exception phrases 318
4.5.2 The mismatch in comparatives 320
4.5.3 The mismatch in ellipsis 322
4.5.4 A conjecture on the interface 323
4.6 The lexicon as an oracle: the case of behalve *324*
4.7 The incompleteness of grammar *327*
4.8 The fruit of incompleteness *329*

REFERENCES 333

INDEX 349

ACKNOWLEDGEMENTS

This book reports on almost twenty years of struggling with grammar-in-action. The language machine it describes has been contributed to by many students in the Leiden University linguistic programmes since 1995. Essential components were co-designed by Christiaan van de Woestijne and Mathieu Fannee. Essential criticism was delivered by Marius Doornenbal, Erik Schoorlemmer, Peter Zinn, Martin Honcoop, Boban Arsenijevic, Emil van den Berg, Jochem Poeder, Jay Landsdaal, Robin Langendijk, Pieter Lamers, and John Terheijden. New fields around the machine have been explored by Marijn van 't Veer, Marja Oudega, Maartje Schulpen, David Shakouri, Mika Poß, Emma Vanden Wyngaerd, and Boris Kozlov.

Many more students inspired us by discussing pitfalls and perspectives of meaningful grammar, and by writing papers and theses on aspects of it. Like many colleagues at Leiden University, they made the sweating worthwhile, showing more than just polite interest in our efforts to keep grammar precise, explicit, deep and operational. Presuming not to have stepped aside from real linguistics more than necessary, we owe this orientation to them.

By and large, the research behind this monograph was facilitated by the Leiden University Centre for Linguistics. Hilke Reckman's research was funded by the Dutch Science Organization NWO as part of the Narrator project. The first author has been hosted by the Netherlands Institute of Advanced Studies to almost-complete the book. We gratefully acknowledge the support of these institutions.

Finally, we are indebted to Saskia de Haan for watching over the book's language.

Crit Cremers
Maarten Hijzelendoorn
Hilke Reckman

0. INTRODUCTION

0.1 A LANGUAGE MACHINE

Amidst efforts to Christianize Muslims, to describe '... profane love in scenes of such repulsive realism that they would shock even an admirer of Henry Miller's fiction' (Gardner 1958:7), to develop voting systems (which are still in use), and to square the circle – his construction approached a circle's surface up to 0.04 % – Ramón Llull invented *logic machines*. In 13th-century Catalonia, he designed and exploited the systematic and mechanical production of propositions. For example, he wrote phrasal concepts on the edges of concentric circles. Connecting the concepts in one system of circles (or disks), by moving the circles with respect to each other, yields propositions with and about these concepts, according to the semantic frame of the particular system. Here is Llull's generator for truisms on the soul (from: *Ars demonstrativa*, c. 1283)– one among hundreds of systems on all fields of philosophical theology.

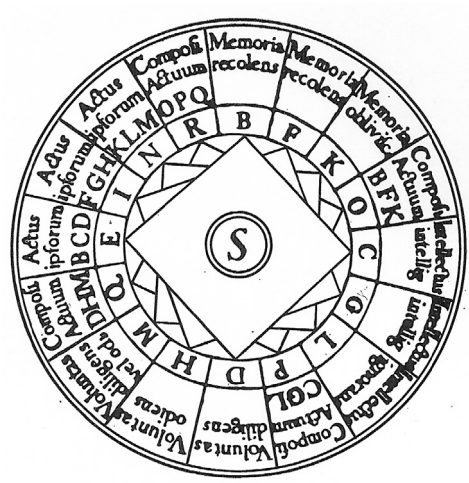


Figure 1 Ramon Llull's Prima figura S

It produces in each state four conjunctions of three 'small clauses', where each conjunction describes a state of the soul, according to Gardner (1958:13) up to a total of 136 combinations. Llull considered these generators as his *Ars Magna*, his major trick. And though Llull contributed little to the material knowledge of his days, Leibniz recognized him as a pioneer in combinatory logic: *let us calculate* instead of endlessly disputing. Llull's disks are grammars, generative grammars of meaningful sentences. Unfortunately, Llull considered his grammars to generate (all and only) true propositions – a quite common overestimation of the power of linguistics. Yet, the idea that propositions are meaningful *because* or *to the extent that* they are generated by a system and that meaningfulness is not an accident, we may grant to the 'obscurantist' Ramon Llull.

Not everyone was and is convinced by Llull's way of working, just as not every scholar of language finds comfort in its mechanization. Llull and Leibniz tried to tell *truth* from *falsehood* – an utterly philosophical and *non-linguistic* enterprise. This book seeks to explore the computability of meaningful language, in order to tell *meaningful* from *meaningless* Dutch – an utterly *linguistic* enterprise. Llull was after the grammar of truth. Unfortunately, there is none. This book is after the grammar of meaningfulness, trying to show that such a grammar can be established and be made to work. It investigates the *modus operandi* of a particular language machine, called Delilah – the biblical connotation is a mere accident. This machine, available at http://www.delilah.eu, executes a language program electronically, inspecting data, computing actions and storing results. It produces meanings that correlate with meaningful sentences, and meaningful sentences correlating with meanings. It can be seen as an act of grammar engineering, an effort to make grammar work and do part of the job of the language processing human brain. It is certainly not an effort to improve on that processing, or to take away the burden of processing. It is meant to model one human language in order to grasp how it is designed and why it works, and in order to be able to test hypotheses on its deepest feature: the relation between form and meaning, the success of arbitrary forms conveying inter-subjective meaningfulness. The machine that we describe and investigate is not an application. It is just a defective, but also instructive and promising model of human language processing. It is a contribution to linguistics-by-computation – it is computational linguistics. Thus, it is neither a 'bachelor machine' – pure design and un-built – like the language machines of Raymond Roussel (Carrouges 1975) – nor just a poem: 'A poem is a (small or) large machine made out of words' (Williams 1969: 256). It operates language, and it works while you sleep. Yet, the concept of mechanical literature, and poetry in particular, is fascinating, and McHale (2000: 11)

tells why. Answering the question why computer-generated poetry '... reads like some mode or other of contemporary American avant-garde writing ...' he suggests that some of these avant-garde texts ' ... may themselves have been generated "mechanically" '. He proceeds to distinguish four methods of mechanical composition: imposing constraints, applying procedures, imposing selection and using combination – indeed the ingredients of modern construction grammar for natural language. Men's built-in text generator may behave mechanically now and then.

The basic hypothesis that we will investigate is that language entertains a non-trivial level of semantic computability, and that meaning is the tractable product of an (almost) mechanical architecture – meaning, not truth, that is, nor adequacy, information, normality or any other non-truth-functional epiphenomenon of language-in-use. Our claim is not that all meaningful aspects of language are computable; in particular, we don't believe that the contextual aspects of meaning are in the scope of deterministic algorithms. Pragmatics of Gricean breed may be understood as a level of meaningful analysis by Levinson (2000) and others: conversational implicatures *e.g.* assume intelligence, awareness and a theory of mind which we consider to reach beyond the minimal assessment for human language learning, usage and understanding. We talk about those aspects of meaning that are not affected by a human's degree of autism. On the other hand, no sentence varies its meaning over the infinitely many contexts in which it can be used. Those components of meaning that do not vary with contexts but rather stay constant are the ones we are after. For example, there are no contexts in which a sentence and its negation mean the same. Their pragmatic impact may be the same, however, for example in a situation where one just says something irrelevant to break a painful silence. Therefore, the difference between a sentence and its negation is a dimension of meaning that we want to explore by computation. This bottom line of meaning we consider to be '... the "holy grail" not only of linguistics, but also of philosophy, psychology and neuroscience – not to mention more distant domains such as cultural and literary theory' (Jackendoff 2002: 268).

Although the aspects of meaning that we aim to compute are rather elementary when compared to 'the pragmatic penumbra closely surrounding sentence-meaning' (Levinson 2000:1), analysis of meaning cannot do without these elements. Even for determining the relevance, adequacy and specificity of a text with or without conversational implicatures, all the small and inevitable seeds of the sentences in it have to be taken into consideration. The big meaning lives on small components, emanating from sentence construal. In

that sense, this book is about the grammatical conditions for computing the propositional molecules of meaning in its broadest sense.

Sentences are meaningful because most strings – statistically: almost all strings – of words are not well formed. To tell the fraction of well-formed and meaningful strings from the virtual overflow of verbal nonsense is the task of grammar, and of a grammar machine. We are not talking furiously sleeping green ideas here. Some sentences may sound weird *because* they are meaningful. The grammar cannot but approve of them, for exactly that reason. The grammar-in-action must identify that utterly small-though-infinite subset of serial words that happens to contain the meaningful propositions, whether they are or turn out to be weird, false, insulting or inadequate. A grammar machine must not stick with Boolean judgements. It arrives at a decision by computing structure, and this structure conveys the semantics. All we have to show, then, is that this semantics is not trivial. That amounts to the view that there is meaning to a proposition, and not just to words. For the present purpose, we will even assume that the meaning of words is *not* the non-trivial level of meaning we are after. The hypothetical level of analysis is the one at which semantic consequences can be established. That level is propositional, by nature. It is, just like syllogisms, not dependent on the particular meaning of words, but it assumes the meaningfulness of whatever constituent that plays a role.
Remarkably, the meaning of a sentence is neither trivial nor vague. Although this meaning is neither listed nor learned, it is almost beyond debate when compared to the meanings of words or to the pragmatics of a text. We need not agree on any particular word meaning in order to agree flawlessly on the meaning of sentences. But agreement, in this case, presupposes shared knowledge and shared interests. Consequently, our agreement on sentence meaning cannot stem from inspection of 'the world': that is exactly where our views and interest diverge. We can only – silently – agree on a domain we never debate: the structure of the language. Therefore, the non-trivial but unquestioned level of semantic computability cannot be established except by language structure. Propositional meaning is neither an accident nor an individual decision, but an intrinsic attribute of computed structure.

The general Turing machine models computability of the partial recursive functions. The Turing machine is neither a model of the human brain nor a model of computability in general. In particular, our brain may be able to compute stuff that is far beyond the reach of our Turing machines. When we try to 'do' language on a Turing device, we just try to determine whether or

not certain aspects of language are Turing-computable. The only importance of this game lies in the question *which* aspects of language can be shown to be Turing-computable. Here, we propose that propositional meaning and logical entailment are in this range. To the extent that we can establish this, it shows that an interesting part of the human language capacity is recursive enumerable or even more restricted. The part of the human language capacity we envisage here, is interesting because it contains semantic combinatorics. Given the computability of syntax and a compositionality ideology, one can argue that the computability of semantics comes for free. That is not true. The reason here is transparent: syntactic structure does not impose an upper limit to the number of propositions it induces. The possible propositions easily outnumber the given syntactic structures they may be related to. Linguistics, then, is bound to demonstrate that compositional semantics is computable all the way up and down. To hope, to say or to claim it is does not suffice. Linguists must do what Leibniz appreciated Llull for: calculate instead of debating the possibilities of calculation.

Consequently, this book is on linguistic engineering, rather than on linguistic theory. But then, linguistic theory is so incomplete or undirected that engineering has to fill the gaps. Another way to look at the problem involved is to say that theories, models and problem solving in linguistics are rather unbalanced. The theories may be *about* the data, and sometimes the data *are* the theory, as in naive constructionism. There is too little distance; there is so little distance that the theory can hardly guide the engineering. The point is made by Shieber (1988) and has hardly lost relevance. This book, therefore, presents linguistic engineering on all levels of sentence analysis, including the theory of grammar. In doing so, it assumes what we call *the language machine hypothesis:*

- that language is partly computable,
- that human language capacities are to be modelled by mechanisms rather than by representations,
- that human grammar is a process – a parsing or generating process – in the spirit of Marcus (1980), and that it is fast, faulty, correctable and goal oriented, rather than being guided by principles of efficiency, elegance and economy, and
- that an automaton may model but not describe the brain.

0.2 LANGUAGE AND COMPUTABILITY

Not every scholar will be convinced that the conjunction heading this chapter makes sense. Even in computational linguistic circles, the computability of natural language is disputed, in at least two senses. In one respect, some computational linguists believe that relevant parts of language are effectively beyond the type of formalization that is required for Turing-computability. In another though not unrelated respect, some computational linguists doubt whether such formalization can be done efficiently. This second type of objection is taken for granted here. In a certain sense, even the authors of this book are convinced that a whole class of language tasks can be performed more efficiently by statistical models than by recursive analysis. We do not believe, however, that interesting semantic tasks are in that class – notwithstanding the numerous efforts to perform inferential tasks by shallow processing, gathering in the TREC and other challenges; for a critique see Reckman (2009). We just try to show that effective computation of dynamic semantics on the basis of a lexicalized categorial grammar is within reach, for a non-trivial fragment of Dutch, on the parsing track as well as for generation.

For the basic construal here, the outline of which we owe to Jan van Leeuwen (p.c.), we are interested in the relation **Means** between sentences of Dutch (NL) and propositions from a formal language L. **Means(**φ, ψ**)** holds between $\varphi \in$ NL and $\psi \in$ L iff ψ is a meaning of φ. Furthermore, we will say that NL is semantically computable iff there is a partial recursive function *f* such that for all p in NL, if f(p) = ψ then **Means(**p, ψ**),** and L is enumerable. Clearly, this function *f* is a parser, sending p to ψ. Suppose we have that parser. Then, NL is semantically computable iff L is enumerable. If L has a finite lexicon – which we assume by definition – then its sentences are enumerable by induction on their length. For every q in this enumeration, check whether f(p) = q. There must be at least one, by the definition of *f*; the first one is found after a finite number of steps. Thus, NL is semantically computable if **Means(**φ, ψ**)** is decidable – equivalency is not at stake here.

The decidability of **Means** amounts to the general question whether human beings can agree on whether a sentence means something with finite means. In general, this is the anchor of successful communication, and there is little reason to doubt we can. On the other hand, L is a formal language, to which we don't have intuitive access. Therefore, we offer a formal way to test **Means**. Suppose we have the counterpart to *f:* a function *g* assigning NL sentences to

formulas in L. Now consider the relation **NLMeans(**φ, ψ**)** between sentences in NL. The relation holds if φ *means* ψ, that is, ψ can replace φ *salve veritate.* **NLMeans** expresses agreement between human beings on the semantic idem-potency of φ and ψ. We take it to be decidable: if humans cannot agree on semantic equivalence, who can? Next, suppose we have a reliable generator *g* mapping meanings onto sentences. Let *g* be such that for all propositions q, **Means(**φ, q**)** iff g(q, χ) and **NLMeans(**φ, χ**)** for φ, χ ∈ NL. In the presence of such a generating function *g,* **Means** is decidable if **NLMeans** is. QED.

In this book we try to partially define the functions *f* and *g* of the foregoing sketch of proof. The *f* is a parser assigning formal meanings to Dutch sentences and *g* a generator assigning Dutch sentences to formal meanings. To the extent that they turn out to be sound and reliable, the fragment of Dutch covered by them qualifies as a semantically computable language by the reasoning above.

The parser and the generator could have been designed as each other's inverse, but actually they are not. That is: in our approach, neither the parser nor the generator has an inverse, for functional reasons. The parser maps a sentence under a particular analysis to a family of meanings. These meanings are underspecified (cf. Bunt 2008): they do not – yet – specify the full net of interpretative dependencies that can be identified in a sentence. There is a post-derivational algorithm, inspecting the complete derivation of the sentence that 'spells out' the full inferential semantic network, including effects of scope and intensionality. The generator is fed with this type of input, but cannot invert the 'spell out' algorithm, for the simple reason that there is no one-to-one correspondence between fully specified meanings and sentences. In fact, both the parser and the generator map their input into classes of expressions. The parser maps a sentence in NL – Dutch – to a subset of L – the representation language – and the generator maps a proposition in L to a subset of NL. So, the parser *f* is a function from NL into the powerset of L, and the generator *g* is a function from L into the powerset of NL. Here is the picture.

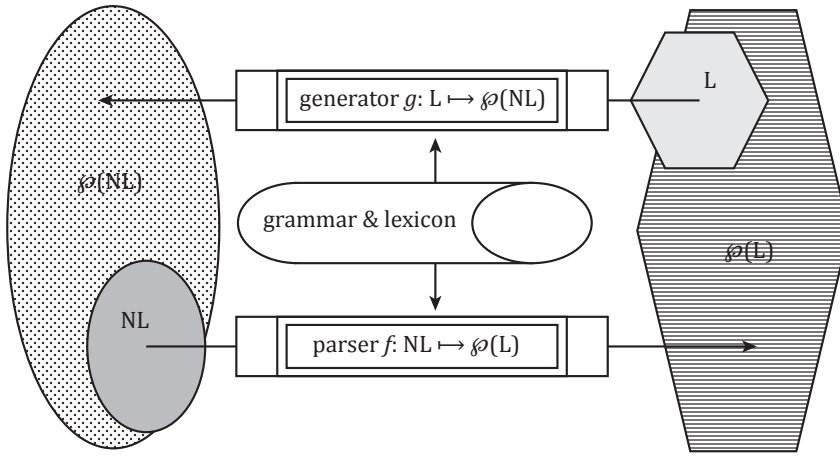


Figure 2 The parser and the generator as functions

Thus we claim that reversible grammars only exist outside the realm of fully specified meaning. The functions *f* and *g* cannot be inverted, as they target domains of a different order.

Talking about meaning, we can hardly escape the question what it is that language is about. Suppose that the interpretation of a language expression were to depend on and vary with the structure of its intended model. It is a good guess to assume that there are more than countably many possible models for a language expression. This interpretational dependency of a language expression on its model would, then, run into incomputability of meaning. Most certainly, however, the power to refer does not reside in language's capacity to refer to – or into or onto – a particular world. It is rather the other way round: language can refer because it does not presuppose any particular organization of what it is about. Dutch can be understood even if the world turns out to be created in six days and constitutes a prototypical black hole, inhabited by square circled demons. As a matter of fact, we talk in Dutch about every possible and impossible world. For language to refer, the minimal model is a non-empty set. That set provides denotations of type *e* – but they may be discarded in favour of generalized quantifiers, according to Montague (1972). The basic sets give rise, without any additional ontology, to subsets, providing denotations of type *<e,t>*, the predicates. One step further, we have sets of sets, of type *<<e,t>,t>*, the quantifiers. That is all we need. Reference to denotations of type *<<<e,t>,t>, t^n>* is obsolete, since those domains will not provide an essentially new algebraic structure (cf. Partee 1992). This three-layered window of denotational types – for names, predicates and quantifiers, basically – provides enough semantic structures to refer to whatever ontology – things, diseases, events, proposi-

tions, times, beliefs, ... This is what natural languages share with programming languages: the ontology is not fixed, but the syntax is. The difference between natural languages and programming languages is that the latter, but not the former, can switch ontologies in one single proposition, without declaration. This is what makes natural language universal and communicative, and the three-type window is an excellent candidate for *the* universal feature of natural language. For this window and its derivatives like functions from one type into another, the lambda calculus provides more than enough possible receipts to catch every possible denotation living there. Neither extensionally nor intensionally, will we easily run out of gas for our semantic machinery.

The basic concept behind Turing's and Church's notions of computability is recursion. The computable functions are the partially recursive ones: those the computation of which can be performed by a sequence or composition of functions with 'smaller arguments' or of less arity, up to those functions that are called primitive recursive. Recursion, more than anything else, is what links the computation of meaning in natural language to Turing computability. Unfortunately, recursion of semantic construal has been mixed up with compositionality in linguistic scenes: the meaning of a structure depends on the meaning of its parts. Some call it *Frege's principle* (see *e.g.* Heim and Kratzer 1998) but Janssen (1986) denies this attribution. *Compositionality* is an unfortunate term because it has been used to detract from the process: the meaning of the whole is a function of the meaning of the parts. Interpretation, though, is a process, not a property, and it is the recursion of the composition process, rather than the meaningfulness of the proper parts of a constituent, that must guarantee the semantic validity of the outcome. On the other hand, mathematics is the art of describing processes in a stative way. But still, what we are looking for is a *procedure* to apply meanings to each other, recursively. Semantic recursion and compositionality are not equivalent concepts, in this respect. A layered treatment of an oracle's proclamation of meaning may be recursive, but not compositional, and the simple one-step concatenation of meanings may be compositional but not recursive. This observation is crucial. The proposition assigned as a meaning to a sentence may itself be composed of propositions, but natural language is not such that simple concatenation suffices to assure that assignment. That is: if *abc* is a well-formed sentence meaning φ and the meaning of *a*, *b* and *c* are the conjunctions of propositions p_{a1} & ... p_{ai} ... &... p_{an}, p_{b1} & ... p_{bi} ... &... p_{bm} and p_{c1} & ... p_{ci} ... &... p_{ck} respectively, then we know for sure that $\varphi \not\equiv p_{a1}$ & ... p_{ai} ... &... p_{an} & p_{b1} & ... p_{bi} ... &... p_{bm} & p_{c1} ... p_{ci} ... &... p_{ck}. More generally:

(1) If ⟦abc⟧ = φ then $\varphi \not\equiv$ ⟦a⟧ & ⟦b⟧ & ⟦c⟧

The basic idea behind this lemma is that the meanings of the parts are supposed to be intertwined in a way that cannot be modelled by simple concatenation. In particular, we assume that the parts have variable-like placeholders that have to be synchronized. This is where Montague's Proper Treatment of Quantification in Ordinary English comes in (Montague 1972). Here meanings are composed by typed lambda conversion, in particular by function composition. The functions are such that the output of one is input to the other. As a consequence, variables are 'pipelined' and the resulting propositions become connected: every variable that was abstracted over is converted 'away' (or bound by operators outside its origin) in the composition process. Modelling this recursive process of composition is the challenge for computational semantics. The recursion must be steered in such a way that the place-holding variables are accounted for properly. Conjunction alone won't do. Function composition might. Still, we will argue in chapter 2 that the composed meaning can be designed as a conjunction of small clauses: unordered self-contained semantic objects in their own right. But this conjunction is neither immediate nor a trivial consequence of structure.

Semantic recursion stresses the arbitrariness of the primitive argument. The difference relates to the nature of semantic units. A complex phrase may be semantically atomic (or less complex) whereas its proper parts may be subject to syntactic manipulation. Picking up the phrase's meaning can be done recursively, but not very compositionally. Since phrasal meanings are utterly important – and, as constructionist linguists argue, dominant – in sentence construal, recursion is more adequate than composition. Yet, one of the most prominent approaches in computational semantics is dubbed *Minimal Recursion Semantics*, where *minimal* modifies *recursion* (Copestake *et al.* 2005). The basic idea, also native to the approach applied here (chapter 2), is that the derivation specifies a family of interpretations – a *packed forest* – and that the spelling out of full meanings is done post-derivationally, according to a protocol that 'reads' and unfolds the packed forest. The family of interpretations is dubbed *under-specification*. Essentially, unpacking the underspecified forest is not part of the grammatical combinatorics. Thus, the computation of the full meaning is set apart from grammatical computation, though fed by it. As a consequence, the nasty implications of spurious ambiguity for the complexity of grammar are avoided by sequencing algorithms, in a way comparable to the cataract of finite state mechanisms that has been developed to fight the opaqueness of monolithic systems (*e.g.* Van Noord 1997).

To make a long history short: from our perspective, Aristotle, Frege, Church, Chomsky, Montague, and Van Benthem equally contributed to the insight that language can be captured by formal means. Aristotle revealed the laws of syllogism and the algebra of reasoning with words. Frege detected the polarity of reference and meaning. Church made functions operational all the way down. Chomsky re-styled grammar, for linguistics and psychology. Montague gave meaning back to grammar. Van Benthem tied lambdas, types and derivations together. And then, there were generalized quantifiers (Barwise and Cooper 1981, Zwarts 1981) and they were the first semantic objects the algebraic structure of which turned out to be linguistically relevant: definable, testable, refutable creatures in an algebraic landscape, interfering with the distribution of phrases and connected to very general, if not universal, properties of natural language (Zwarts 1983). Their being there and their nature made clear that language lives somewhere in the brain, and that it makes itself known. There is no reason to underestimate or obscure or neglect it. There is a system that facilitates conveyance of thoughts, findings, emotions. It may have arisen from the lust for communication, or from the lust for expressing one's thought or from both – communication is the interaction between the verbalized straw men of our minds. Like most things in evolution, it is not designed to do what it turns out to do, but it is heavily affected by its function. That system is computable, partly by symbolic means according to Marcus (2003), and it may as well have a Dutch face. *Quod sit demonstrandum.*

0.3 THE BOOK

As we maintain a rather conservative view both on computational linguistics – it is linguistics – and on semantics – meaning exists and can be computed – the book has a rather conservative set-up. It consists of three chapters devoted to the main linguistic engines of the language machine: syntax, semantics and the lexicon. And it concludes with a fourth chapter in which the relationship between the three engines is reviewed.

The chapter on syntax introduces a rigid, modal, constructionist, combinatory categorial grammar, Categorial List Grammar (CLG). Its main linguistic and computational properties are made explicit and evaluated. The chapter describes the main syntactic patterns of Dutch and the way CLG covers them, in an all-and-only perspective, while linearizing strings and structuring com-

plex symbols at the same time. Furthermore, the chapter scrutinizes how the CLG is exploited in parsing and generation by the Delilah system.

The chapter on semantics presents three different but related instances of logical form, dubbed Stored, Applied and Flat LF. They differ in specificity and the way they are derived. The need for the threefold is established by reference to the variety of tasks with which a formal interpretation of natural language has to comply. In particular, the interference of compositionality with reading selection, scopal phenomena, intensionality and semantic inference is highlighted. The chapter describes how parsing in the Delilah system leads to a fully specified semantic representation and how this fully specified semantic representation can drive generation.

The third chapter deals with the data for meaning-driven parsing and generation, the lexicon. The Delilah system is lexicalistic, in an almost extreme way: it holds entries – lemmas, in effect – for each use of each phrase. The explosion of phrases following from this strategy is countered with smart indices and utterly fast retrieval. The lexicon consists of unique fully specified complex symbols, in the form of graphs open to unification. The chapter describes the off-line construction and the on-line retrieval of the database. The way we exploit the lexicon efficiently for parsing and for generation is revealed. As a matter of fact, the lexicon described here is more of a *constructicon*: it holds the phrasal components of Dutch in a non-hierarchical, matrix-like database, the front-office access to which is much more relevant for computability than its back-office genesis and hidden structure. In these chapters, essential data structures are taken from the Delilah operation mode. The make-up of the data structures may slightly differ depending on the process module handling them. Notwithstanding these differences, they always represent the very same type of underlying graph. In discussing the data structures we do not differentiate between categories and types in a fundamental way.

The final chapter proposes a revision of the formal relationship between syntax, semantics, and the lexicon. It claims that some of the less elegant solutions in the preceding chapters trace back to a fundamental incongruence in the architecture of formal grammar. It claims that grammar is bound to be incomplete – that certain valid statements about form and meaning cannot be derived by valid syntactic or semantic computations, and that this is a desirable state-of-affairs. In this sense, we feel this book to be a positive interpretation of Hugo Brandt Corstius' almost untranslatable 'first law of computational linguistics': *Wat men ook doet, de betekenis gooit roet* (Brandt Corstius 1978: 110) – 'Whatever you do, the semantics comes through'.

To summarize, the book is meant to be a defence of at least the following claims:

- the propositional content of a Dutch sentence can be computed;
- for every proposition, a Dutch sentence can be computed that entails that proposition;
- the proposition associated with a Dutch sentence can be represented as a flat conjunction of small clauses, indexing the propositional content, with skolemized events and states as a backbone;
- when computing meaning, underspecified and specified levels of representation are produced by essentially different algorithms;
- given this separation, no homomorphism between syntactic and semantic procedures needs to be established for computational reasons;
- the parser computing propositions and the generator producing sentences are not each other's inverses but exploit the same lexical and grammatical resources;
- underspecified levels of meaning can be computed by unification;
- the lexicon of Dutch is phrasal;
- the lexicon of Dutch can be organized and accessed as a non-hierarchical family of complex signs;
- all relevant unifications in the grammar of Dutch can be controlled by a limited number of modes defined on a rigid categorial grammar;
- formal grammar is incomplete iff it is consistent.

1. SYNTAX: the game of recursion and discontinuity

1.1 THE NEED FOR SYNTAX

Dutch is a natural language – to a certain extent. As such, it is torn by finiteness and two infinities. First, there is the finite set of all lexical atoms, the lexicon. For the present purpose, it is immaterial how we define the elements of this set, as words, morphemes or phrases. There has to be a finite set that is not an algebra: its members are not deduced but rather elected or assigned or proposed or enforced, but by no means are they theorems of a system. Without this set being finite, a language would be neither learnable nor decidable. Second, we have the lexicon's *Kleene closure*: the infinite set of all finite strings over that lexicon. Since there is no reason to assume a longest string, there are infinitely many of them, but each is of finite length. Third, there is the infinite set of finite strings over the lexicon that we take to be expressions of the language – the language in extension. Again, it is immaterial how we define this set – by well-formedness, by interpretability, by pragmatic criteria or any other sensible norm.

Language users are familiar with each of the three sets. They recognize the elements of the lexicon. They are able to produce and recognize lists of lexical items of their language without being seduced to interpret those lists as meaningful. They are able to recognize and interpret members of the distinguished set of (would-be) sentences. At the same time, there is no reason to assume that felicitous language use presupposes any awareness of the nature or extension of these sets. In order to describe the human language faculty, however, it is advisable to postulate the sets.

The basic fact about language-as-a-system is that the two infinite sets do not coincide: a language is a *proper* subset of the Kleene closure over its lexicon. No grammarian ever proposed any language to coincide with its Kleene closure. No author of formal languages ever proposed an *anything goes* language. Such a language would be bound to challenge meaningfulness: meaning results from distinction and distinctivity, but in the Kleene closure all strings are born equal. Only length or the occurrence of individual atoms could count as contributions to meaning. Since every atom occurs in any order with other atoms, it is hard to see what an individual atom in a random string could contribute to its specific meaning. Of course, the length of a string could count as meaningful. Since the length is a fixed extensional property of a string, length can only be meaningful in the way integers are meaningful. Numbers, however, are meaningful in a way languages cannot afford to restrict themselves to. Numbers are rigid designators. Tomorrow, they will refer in exactly the same way as they did yesterday. They are *one-world* designators, and do not provide content outside that world – unless we use numbers in other languages that are not rigid. That leaves only the possibility for strings in the Kleene closure to carry meaning randomly or by oracle. It is hard, however, to look at meaning as an accident, not steered by material properties of the expression. It would make your present enterprise of reading this text vacuous. The famous but not uncontroversial concept of *compositionality* of meaning represents an effort to explain why that enterprise is not completely vacuous (even if the concept cannot be attributed to Gottlob Frege, as Janssen (1986) convincingly argues). Accidental meaning would not convey information, just sound.

Almost by definition, then, languages are 'smaller' than their Kleene closure. Since we have no reason to assume that the order of infinity between the sets is different – as is the case with *e.g.* the sets of rational and real numbers – 'being smaller' here must be approached stochastically: the chance that a string, randomly chosen from the Kleene closure, is also in the language is less than one. As a matter of fact, the chance is close to zero.

The selection procedure that (dynamically) identifies that small island of language in the sea of the Kleene closure is called grammar. It is a characteristic function over that closure, the function dividing the sheep from the goats. Grammar is the *theory* of the language. Most linguists assume that the function is operative in a man's brain, but they entertain some disagreement as to its whereabouts and 'wherefroms'. The function, however, is most certainly required in a computational system – if that system aims at interpretation. Of course, a lot of language-related tasks could be performed without access to an intensional description of the language; you don't need grammar to

count words or to find strings. But if meaning comes into play, distinction is required. No automaton can reliably assign meanings to sentences and not tell interpretable from uninterpretable ones, for the same reason that a computer that adds * and #, can hardly be trusted when adding 3 and 4. If anything is meaningful, meaning evaporates. Language lives on selectivity.

We now have the following situation: a finite lexicon L, its Kleene closure L* = $\{<w_1 \ldots w_n> \mid w_i \in L\}$ and natural language NL $\subset$ L, where NL is defined by a Boolean function f_{NL}, its grammar. NL is not necessarily well-defined. Its characteristic function may be fuzzy, and certainly a family of languages that intersects with NL can be defined or, more precisely, a family of languages that shares infinite subsets with NL. This family covers the varieties of Dutch one might want to focus on. It is more intriguing whether we could find natural languages in the complement of NL in L*, in the set L* - NL: languages that live on the same lexicon but would not share any expression with NL or, more precisely, that share at most finite subsets with NL, *e.g.* a set of one word expressions. Probably, no such languages would be accepted as possible natural languages. For formal languages, though, complementary languages can be defined and exploited in simple ways. We might want to have, then, a theory that determines which subsets of L* qualify as natural languages. That theory is the *theory of grammar,* embodying the classical Chomskyan concept of possible natural language. We won't touch upon that theory here directly, as we are dealing mainly with the theory of Dutch. We assume, however, that the theory of grammar will cover our analysis of Dutch. In defending that 'local' theory, we will have to refer to more general points, now and then, in particular when matters of restrictivity are at stake.
Grammar must be finite to be applicable. That is why a grammar cannot be identified with the language in extension. We (humans, computers) need a finite, intensional description – in whatever form: phrase structure rules, algorithms, parsing routines, integrated circuits – to handle the inevitable infinity of language. Exploiting finite means to deal with infinite sets provokes recursive use of those means. That is: infinite, structured languages may have sentences that show bracketing patterns like

(2) $\ldots[_x \ldots [_Y \ldots [_X \ldots] \ldots] \ldots]$

In these patterns, a certain type of string occurs within an instance of the same type, with or without – mostly with – intervention of some other type or category. In many natural languages, all 'major' categories (sentences, nominal constituents, prepositional constituents, verbal constituents) have

this property of re-entrance within themselves. Crucially, the pattern postpones the full interpretation of the upper occurrence until after the interpretation of the lower occurrence. And that is where syntax comes in: the syntax specifies which phrase depends on which phrase. Syntax is the carrier of recursive embedding and dependency. It defines a finite family of categories or equivalence classes and a finite class of relations between them. It assigns lexical atoms to at least one class or category. It guarantees that every string *of the language*, no matter how long or complex, can be divided into representatives of those categories. The process of that division is called *parsing*, and it is one of the more interesting ways to execute a characteristic function over the set of all strings.

This is the – by now – classical approach to grammar, as practised in that branch of mathematics that studies formal languages and automata (cf. Hopcroft *et al.* 2001). It relates grammar to computation: an explicit formal grammar determines a class of automata with respect to its use of resources, and every computable problem can be represented as a characteristic function over a set of sequences. The notion of automaton is used here in the mathematical sense, as a unit of data, processing, control, and storage, *aka* the Turing machine. The first track of the relation has become canonised as the Chomsky hierarchy, relating properties of languages, grammars and automata. The second track is due to digitalization: all automata can be defined in a digital language, with a grammar; the sentences of that language are interpretable as automata. In that sense, the circle is closed: languages have grammars, grammars determine automata – they can be written as programs – and automata are defined in languages.
In this view, syntax is the backbone of grammar because recursion resides here and recursion mediates between finiteness and the infinities of language. Moreover, in the ever-swelling literature on language evolution and animal communication, recursion has become prominent in the field of cognition after Hauser and Fitch (2003) reduced the exclusivity of human language to just that. Independent of its cognitive status, though, it must be something other than just "... the product of a given theory designed to account for language structure" (Heine and Kuteva 2007:265). To see why, consider this. Language is meaningful *because* not every conceivable string is meaningful. To tell the infinite number of meaningless strings from the infinite number of meaningful sentences, and/or determine the meanings of the meaningful ones, we need a system. That system can be learnt and/or acquired, and must be finite, for that reason. For the system to be finite and still be effective and discriminatory, re-use or re-entrance of resources (cat-

egories, constructs, embedding procedures) is essential. That is recursion: linking phrases to each other in an asymmetrical style, *ad libitum*. Recursion offers a framework for making expressions semantically dependent upon each other, as well as for thematic relations, referentiality, intensionality, mood, aspect, and so on. This view leaves open many ways of embedding expressions semantically into each other. Physical – prosodic – embedding as in (2) is just one of them. Anaphora or kataphora is another: the semantic effect of (2) – the interesting effect – can be mimicked by a structure like (3), where the indices indicate co-evaluation.

(3) $\ldots[_X \ldots \text{it}_j \ldots] \ \ldots \quad [_y \ \ldots[_X \ldots] \ldots]_j$

Here too, the valuation of the 'highest' X is dependent on the interpretation of the lower X. For example, the following two texts are idempotent. They both subsume meaningful expressions under each other.

(4) The man who read a book said that the river was too wild
(5) There was a man. He read a book. He said this. The river is too wild.

Their logical forms must be the same, as their entailments converge: in both texts, the speaking subject is described as a man reading a book, and the wildness of the river is not presented as a fact, like the speaking event, but is attributed to that man.

Anaphora, realized by copying, binding or co-reference, places the antecedent under the semantic regime of the sentence containing the pro-form. As a matter of fact, anaphora and kataphora do *overtly* what in the semantic representation of a complex sentence is conducted *covertly* by reification and binding; there, *logical* anaphora are called for. Overt anaphora and kataphora enforce recursion.

In the preceding examples, the semantic dependencies are marked by syntactic means: categories, valency, embedding, linear order. Thus, interpretation is syntax-driven, and the interpretability of language is anchored by the recursivity of syntax. That turns the syntax in a grammar into a crucial factor for the computation of meaning. In this vein, it cannot be accidental that an explicitly semantic and wide-coverage grammar model like that of Sag *et al.* (2003) sails under *Syntactic Theory*. And consequently, the nature of the syntax is part of the message. The system presented below functions well for parsing and generating Dutch. It is neither *the* syntax of Dutch nor *a theory of* syntax, but describes a viable and operational way of demonstrating what our enterprise is about: relating meaning to form in a domain where form is extremely discriminating and meaning is far from trivial.

1.2 FORMS OF DUTCH

As a natural language, Dutch shows signs of syntax. This section presents a very concise characterization of those aspects of Dutch syntax that determined the nature of DELILAH's grammar. It is neither complete nor balanced, as it focusses on those properties of Dutch phrase-structure formation that resist uncomplicated accumulative combinatorics.

Dutch is an SOV language in the following sense: the nominal and prepositional objects of a verb precede it. Verbal and propositional complements follow the verb. Prepositional complements can occur to the right of a verb, and verbal complements may occur to the left. Whether SOVness is a derived or a fundamental property of languages is not relevant here.

Some verbs selecting infinitival complements must or tend to cluster with the verbal head of that complement. This may lead to multi-clusters of verbs. Dutch is infamous for maintaining a particular order in that cluster: the selecting verb can or must precede the head of its complement. In multi-clusters of verbs that also select nominal complements, this order gives rise to crossing dependencies between nominal and prepositional objects and their respective verbs. Here is an example, inspired by Evers (1976).

(6) … dat Jan de man Marie een koe zag helpen leren dansen
… *that Jan the man Marie a cow saw help teach dance*
… 'that Jan saw the man help Marie to teach a cow (how) to dance'

The verbs cluster, and this cluster is preceded by their aggregated nominal arguments in the same order: *zag* selects *Jan*, *helpen* selects *de man*, ... and *een koe* controls *dansen*. Evers observed that in German verbs cluster too, but tend to do so in reverse order. A major difference between the Dutch and German orders is that in Dutch the head of the sentence – the verb dictating argument structure at top level – occurs more to the left. For parsing, this may be an advantage, since the sentence structure is clarified at an earlier stage. In the German order, the verb cluster must be processed as a whole before any clue as to the sentence structure and the status of the argument is revealed. The Dutch order may be more complex from the point of view of memory management (cf. section 1.8), but the more sophisticated appeal to random access memory – *i.e.* exploiting memory as it comes – may improve parsing, as one can conclude from observations reported in De Vries *et al.* (2011).

This state of affairs is rather rare among the languages of the world. It has been used to challenge the idea that the grammar of natural languages be context-free. The point for Dutch was made by Huijbregts (1976). The concept of context-freeness amounts to the claim that the internal structure of categories in natural language is not essentially determined from outside, and, equivalently, that natural languages can be parsed – and generated – with a memory management that is more restrictive than random accessibility. The discussion on the status of crossing dependencies and related phenomena has been part of the genesis of Generalised Phrase Structure Grammar; *cf.* Pullum and Gazdar (1982). Since crossing dependencies are what made Dutch famous among linguists, this phenomenon influenced our choice of the particular categorial grammar exploited in DELILAH. Considerations relevant to this choice are discussed by Stabler (2004) in an extremely useful exercise on grammatical capacities beyond context-freeness. The topic will be further discussed in section 1.8.
Crossing dependencies and verb-clustering are important sources of *discontinuity* in the syntax of Dutch: the phenomenon that two phrases that establish an essential grammatical relationship occur separated from each other rather than adjacent in the sentence. Of course, a verb and its object are striking examples of such a pair themselves. But adverbial or propositional adjuncts can also occur amidst phrases to which they cannot be linked directly:

(7) … dat Jan de man misschien Marie een koe zag helpen leren dansen
… *that Jan the man maybe Marie a cow saw help teach dance*
… 'that John maybe saw …'

Misschien 'maybe' is linked to the verb complex and modifies some event, but occurs here at a seemingly random position in the sequence of nominal phrases. Its position may have an effect at the level of information structure, though, but that is not well established. Verb clustering provokes this type of discontinuity in the *Mittelfeld*: verb clustering 'obscures' constituency, and consequently the semantic or syntactic target of adjuncts has to be determined and/or reconstructed by more complex manoeuvres than just concatenation.

Just like some other SOV languages, Dutch has a verb-second effect. In the absence of a complementizer, a finite verb or auxiliary – in Dutch, finiteness is exclusively morphologically marked – can or must occupy that position, thereby determining the status of the sentence.

(8) Niemand heeft de dronken oom durven corrigeren
nobody has the drunken uncle dare correct
'Nobody dared to correct the drunken uncle'

(9) Gaat het mis, treedt de minister af
goes it wrong, steps the minister down
'If it goes wrong, the minister will step down'

In the first clause of (9), the finite verb is in complementizer position, marking it as a (conditional) subordinate clause. Verb-second (or verb-first, for that matter) also enhances discontinutity: it can strand adverbial modifiers and may separate a verb from its immediate objects.

The complex structure of the Dutch *Mittelfeld* allows for complex forms of coordination. Cremers (1993a) argues that in Dutch almost any position in a sentence can give rise to coordination, including coordination of non-constituents. Thus, coordination is a source of discontinuity; in the following example, the verb *schilderen* 'paint' is cruelly separated from its intended objects *de auto* and *de fiets:*

(10) Ik had de auto met jou blauw en de fiets met haar groen willen schilderen
I had the car with you blue and the bike with her green want paint
'I wanted to paint the car blue together with you and (to paint) the bike green together with her'

Complex coordination is not necessarily the same as ellipsis, though ellipsis does involve coordination, in our analysis. Ellipsis, however, is the kind of complex coordination where the left context of the coordination element must be sentential.

(11) Ik zal Jan een laptop geven en jij Peter
I will Jan a laptop give and you Peter
'I will give Jan a laptop and you, Peter'
(12) * Ik zal Jan en jij Peter een laptop geven
I will Jan and you Peter a laptop give

Complex coordination in Dutch influenced the set-up of the DELILAH syntactic architecture to large extent, as will be explained in section 1.7.4.
A standard form of discontinuity entertained by Dutch is movement of constituents to the left periphery of a sentence, the position also known as *SPEC CP.* As usual, movement here is a metaphor for one constituent governing or controlling two non-equivalent positions in the sentence structure. Question words, relative pronouns, arguments and adjuncts, even verbal complements, occur at *SPEC CP* – obligatorily or optionally. Though they may come from far and deep, their origins are not arbitrary, for the sake of wellformedness and interpretability. As a consequence, Dutch defines islands of all kinds from

which a constituent cannot escape, or only under strict conditions (cf. Honcoop 1998, Szabolcsi 2006). These islands may be syntactic, lexical and/or semantic. Here are some examples, where the intended rightmost position is marked *t*.

(13) *Welke prinsen* denk jij dat *t* de receptie zullen bezoeken?
which princes think you that the reception will visit
'Which princes do you think will visit the reception?'

(14) * *Welke prinsen* denk jij dat de receptie die *t* bezochten in de kerk was?
Which princes think you that the recepetion that visited in the church was

(15) Elke sonate *die* ik *t* speelde, wekte ergernis
Every sonata that I played raised annoyance

(16) * Elke sonate *die* ik een deel van *t* speelde, wekte ergernis
Every sonata that I a part of played, provoked annoyance

(17) *Mijn vader* heeft zij *t* niet gekend
My father has she not known
'My father she never knew'

(18) * *Mijn vader* betwijfelde zij of *t* kon lezen of schrijven
My father she doubted whether could read or write

(19) *Aan het dak* hing mijn vader hammen te drogen *t*
At the roof hung my father hams to dry
'From the roof my father hung hams for drying'

(20) * *Aan de wilgen* hing de grijsaard de viool *t*
In the willows hung the old-man the violin
(compare: * The bucket he kicked)

(21) *Met onwaarschijnlijk geweld* raasden de drie orkanen *t* over de eilanden
With incredible violence raged the three tornados over the islands
'Incredible violently, the three tornados raged over the islands'

(22) * *Met onwaarschijnlijk geweld* raasde geen van de orkanen *t* over het eiland
With incredible violence raged none of the tornados over the islands

The grammar of Dutch has to account for such barriers to interpretation. And since recursion seems to be the issue here, syntax must carry the load.

One could easily come up with many more intrinsicalities of Dutch syntax. As a matter of fact, Dutch is one of those languages the structure of which has been studied and documented extensively. Above, only a few of the more complicated patterns are set out. An extensive, authorized description of Dutch is available at http://ans.ruhosting.nl/. A less descriptive, more theoretically motivated 'modern grammar of Dutch' is being developed at present by Hans Broekhuis (and others at http://www.taalportaal.org). An intriguing side-effect of trying to make Dutch syntax work for interpretation is that one cannot afford not to solve problems as they occur; otherwise, it will immediately corrupt the overall result. Here is a good example. Dutch has a class of pronouns that act as left complements of prepositions, but are not necessarily

adjacent to them. So, the infamous *er* can occur in any of the marked positions in the following sentence, but only in one of them at the same time, but it can always only be interpreted as the anaphoric complement of *over*.

(23) … dat ik (*er*) mijn vader (*er*) nooit (*er*) met mijn moeder (*er*) over heb horen spreken
… that I (it) my father (it) never (it) with my mother (it) about have hear speak
… 'that I never hear my father talk to my mother about it'

None of these positions is marked, except that *er* does not intrude into 'closed' constituents. Moreover, the same pronoun *er* – or a look-alike – can also occur as an existential marker, as a place adverb and as a partitive pronoun, and in all sorts of combined functionality. Although interesting aspects of *er* are revealed by Bennis (1986) and Van Riemsdijk (1978), one has to invent syntax that is not available anywhere in order to decide for a given sentence which *er* is at stake and whether and how it can or must be interpreted – not to mention how to accommodate *er* in generation.

1.3 THE TASK FOR SYNTAX

Having established the need for a syntax of Dutch, it is wise to reflect upon its task. Syntax is not all of grammar. In our concept of a lexicalized grammar, syntax is that dynamic module that directs unification of complex symbols. Unification of complex symbols is the core business of grammar, both computationally and cognitively. In a human's brain as well in the digital lexicon, much more information is stored than can be covered by syntax. Complex symbols, feature structures, signs, templates or whatever we call our information units, are supposed to accommodate all knowledge of language to the extent that the knowledge can be attributed to phrases (localized) and is expressible. Knowledge of language extends far beyond the combinatoric mechanisms (routines, principles, parameters, ...); knowledge of language is also about meaning, about sound, about use, about style, about semantic and pragmatic fields. Basically, syntax is only arranging the concatenation of phrases, and in this respect quite different from semantics, which is governed by functional application.

Yet, syntax is far from trivial, for the simple reason that it has to be extremely discriminating while steering the powerful engine of infinity, recursion. Sim-

plistic syntax over-generates: it allows more unification to be tested than efficient processing and subtle interpretation can afford. Too restrictive syntax under-generates: it may miss meaningful combinations of signs. Good syntax is in between. It defines the combinatorial space for a natural language. To the extent that it does so adequately and transparently, it contributes to the notion of possible language, for example by excluding permutation closure.
If syntax is to direct unification, unification and the data structures it operates on have to be defined carefully. Here we discuss the major outline. In the chapter on the lexicon the complex symbols are presented more in detail.

Our data structures are feature trees: acyclic connected directed graphs in which each node labels an attribute and its outgoing edges define the value for that attribute in that graph. Technically, this concept is not essentially different from the feature structures that characterize Head-Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994). The main difference, as will be illustrated in chapter 3, is that we do not impose any regime on the nature of features, and that we allow feature values to be plain variables. More esoterically, we consider feature graphs to be *the* objects of combinatoric manipulation, rather than 'just' descriptions of those objects, which Kracht (2004:ch. 6) takes HPSG attribute-value matrices to be. Furthermore, we call a feature graph a *template*, because Delilah exploits feature trees not only for grammatical unification, but also for the dynamic construction of the lexicon. Here is a representation of a simple template, one of the lexical templates of the finite verb *werkt* 'works'.

(24) *a lexical template of* werkt *'works'*

```
ID:A+B
HEAD:CONCEPT:work
     PHON:werkt
     SLF:work
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:2
            TENSEOP:at-pres
            VTYPE:nonacc
PHON:C
PHONDATA:lijnop(werkt,A+B,[arg(left(11),wh,D)],C)
SLF:{{[E&(B+F)#G, H@some^I^and(and(quant(I,some),
    work~[I], event~[I],entails1(I,incr)), H,
    entails(I,incr))&(A+B)#J],[],[]},
    and(and(agent_of~[J,G],attime(J,K)),tense(J,pres))}
SYNSEM:CAT:s
       EVENTVAR:J
       EXTTH:agent_of~[A+B,G]
       PREDTYPE:nonerg
       SUBQMODE:L
       TENSE:tensed
TYPE:s\u~[np^wh#B+F]/u~[]

ARG:   ID:B+F
       PHON:D
       SLF:E
       SYNSEM:CASE:nom
              CAT:np
              FOCUS:focus
              NUMBER:sing
              PERSON:2
              OBJ:subject_of(A+B)
              QMODE:L
              SUBCAT:pron
              THETA:agent_of
```

The template is hanging from one (hidden) top node, *template*, denoted by TOP. From there, vertices lead to nodes ID, HEAD, PHON, PHONDATA, SLF, SYNSEM, TYPE, and ARG. There is no ordering between these vertices. The node HEAD has outgoing vertices to CONCEPT, PHON, SLF, and SYNSEM. The nodes PHON and SYNSEM under HEAD are not equal to the nodes PHON or SYNSEM under the top node: each node is uniquely defined by a path from the top node. Thus, we have three different nodes *phon* in (24): TOP:PHON, TOP:HEAD:PHON, and TOP:ARG:PHON. They are separate, distinct, unique and independent.

A node without outgoing vertices is a *leaf* or *terminal*. It represents values by definition, and may be variable or instantiated, and either complex or simple. The value for the node TOP:SLF is a complex structure itself – not a graph. The value for TOP:TYPE is a complex category constant. The value for TOP:ARG:SYNSEM:QMODE is a variable, identical to the value for TOP:SYNSEM:SUBQMODE. For each attribute, the set of values it may assume is defined and finite. Thus, a template is finite, by definition.

Templates store information about phrases, in relation (attached) to one lexical part of the phrase. From this perspective, (24) specifies a sentential phrase headed by *werkt*. Consequently, templates specify information on three levels:

(a) attributes of the phrase as a whole
(b) attributes of the (lexical) head
(c) attributes of the non-head parts of the phrase.

The properties of the head are specified as values of the node TOP. The properties of the non-head parts of the phrase are specified as values of nodes TOP:ARG. Of these nodes there may be several, but each TOP:ARG node is uniquely indexed. All other paths in the template specify values of the phrase as a whole. The information at the three levels is not necessarily disjoint. As a matter of fact, attributes sharing values at different levels of the templates specify networks of local dependencies, like agreement. In (24), all variables that occur as (part of) values occur more than once, thus specifying agreement.

Two templates unify when their specifications at relevant attributes are compatible. The unification amounts to merging the templates and is of course a template itself. Technically, the unification of two templates t1 and t2 is a template t* which subsumes both t1 and t2; a template subsumes another if it is more specific, being obtained from the other by valuation of variables. The unification result therefore is at least as specific as each of the entry templates. Unification operates point-wise. Singular attributes are compatible if they are identical. For each specified pair of attributes, there is a check whether the values are compatible. A variable is compatible with anything, and is instantiated at unification. Two constants are incompatible unless they are equal. Two structures are compatible if they match.

The syntax determines under what conditions templates – phrasal descriptions – can or must unify. As the templates are specified at different levels, the syntax is sensitive to the internal structure of the templates. In particular, the syntax determines which part of a template should unify with another

template. More precisely: if the syntax sends templates A and B to unification, it arranges unification of A with one particular subtemplate TOP:ARG of B by replacing ARG with the top of A. The other parts of B will be affected only by the possible instantiation of variables resulting from this unification.
A syntactic category, then, is a projection of the internal structure of a template. Every template is specified for a literal expressing its syntactic status; the values here are, *e.g.* S, NP, and VP. They label the phrase as a whole. If the template has attributes of type *top:arg*, these attributes may also carry syntactic labelling. If so, that label occurs in the syntactic category, exactly once. The syntactic category of a template, therefore, reflects the label of the phrase as a whole and of the labelled *arguments* in the template. It is specified as the value of the attribute TOP:TYPE. As the template is assigned to the head of the phrase, the syntactic category, being a value in that template, can also be seen as being assigned to that phrase. In that sense, the finite verb *werkt* in (24) *is of* or *belongs to* the category `s\u~[np^wh#B+F]/u~[]`.

The category relates information on the internal structure of the template to conditions under which it can be unified. Abstracting from the label B+F, which solely introduces a template internal index, the category `s\u~[np^wh#B+F]/u~[]` for *werkt* expresses the following statements:

(a) the full phrase has syntactic label S “`s `”
(b) one attribute TOP:ARG is labelled NP “`[np `”
(c) there are no other specified arguments “` ], []`”
(d) the subtemplate labelled NP can unify with a template labelled NP occurring to the left of *werkt* “`\ `”
(e) this unification is submitted to a package of conditions *wh*, specified in a syntactic rule of category merging “`^wh `”
(f) the template was not subjected to leftward or rightward unification “u”

In the same vein, a syntactic category `vp\a~[np^0]/u~[s_sub^6]` summarizes a template with phrase label VP with an argument labelled NP that can unify leftward under rule condition 0, with a completed leftward unification for another argument indicated by flag *a*, with an argument of label S_SUB that can unify rightward under rule condition 6.

The arguments that are open for unification at each side are organized as a list. An argument is open for unification only if it is in the list.

The rules of syntax specify exactly and completely the conditions under which unification can take place. They are triggered by the label of arguments and by directionality. Every rule of syntax mentions three syntactic categories:

(a) the primary category, introducing the argument to be unified and its subtemplate
(b) the secondary category, introducing a co-labelled counterpart to the argument
(c) the resulting category: the category of the merged template.

Since the rules are sensitive to labels and directionality, the syntax resulting from these rules is essentially *anti-symmetric.* Interchanging the primary and secondary categories yields no merge, or a different one. Thus, the unification process is anti-symmetric too. In this sense, our grammar complies with the fundamental propositions of Kayne (1994).

For this triple, the rule specifies a complex set of constraints:

(a) the syntactic structure of the primary template (the one with the specified argument) at unification
(b) the syntactic structure of the secondary template (the one co-labelled with the argument) at unification
(c) the syntactic structure of the unified and merged template.

The first two can be seen as preconditions of the unification. The third part of the package summarizes the output of the unification. The preconditions on the primary and secondary categories define a particular *cancellation* mode. For each argument in a syntactic category this cancellation mode is explicit. It is the main combinatory specification of a syntactic category.

Here is the general format of a syntactic rule, specifying leftward cancellation; the rightward case is similar. The operation is defined relative to the cancellation mode *i*; X⊕Y generalizes the appending of lists X and Y as XY or YX, depending on the cancellation mode.

(25)

primary category:	PRIM \ PLF~PLA / PRF~[SEC^i|PRA]
composition operator:	$\otimes_i$
secondary category:	SEC \ SLF~SLA / SRF~SRA
resulting category:	PRIM \ PLF~(SLA⊕PLA) / RF~(PRA⊕SRA)

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category);
^i: cancellation mode; *RF:* some flag determined by *i*

The constants in the formulation of a rule are these, labelled as properties of the syntactic formalism:

(a) the label of the primary category is the label of the resulting category (*output*: conservation of head)
(b) the label of the secondary category occurs as a label in an argument stack of the primary category (*input*: well-foundedness of merge)
(c) only one argument is affected by the merge (*output*: exclusiveness of merge and directedness)
(d) all non-affected arguments of the primary category and the arguments of the secondary category are assembled in the resulting category with the same labels and cancellation modes (*output*: conservation of arguments)
(e) in the resulting category, the left stacks and the right stacks of the input categories are assembled to a left stack and a right stack, respectively (*output*: conservation of direction)
(f) argument stacks are appended and the order of arguments in a stack is unaffected (*output*: stack integrity)
(g) the flag of the assembled stack for the passive direction in the resulting category equals the flag for that direction in the primary category (*output*: passivity)

Normal business in a syntactic rule implies the following properties:

(h) the label of the merged argument does not occur in the resulting category (*output*: cancellation)
(i) the target argument of the unification is at the top of its stack (*input:* linear ordering)
(j) the assembled stack for the active direction in the resulting category is marked for being affected (*output:* affectedness).

'Normal business' means that deviation from these practices marks that rule as exceptional and a candidate for reconsideration. For example, the rule han-

dling 'floating' *er* in Dutch (see (23)) may exceed normal business, as *er* placement is hardly understood.

Finally, the rules are parameterized with respect to:

(k) the labels of non-affected arguments and the label of the primary category (*input:* labelling)
(l) the arity of stacks (*input*: arity)
(m) the flag of stacks in the primary and secondary categories (*input:* flag).

The instantiation of these parameters defines a certain merge mode. So, syntax specifies which templates should be sent to unification on the base of their internal structure.

The parameters of the syntactic rule determine their variety. All parameters are finitely valued: there is a limited number of syntactic labels, templates have only finitely many arguments and stacks have a restricted number of flags. Consequently, a natural-language syntax along these lines can only have finitely many rules. As a matter of fact: syntactic labels are very restricted in number, and so is the argument complexity of templates. Therefore, the number of syntactic rules that can be specified is very limited, the main factor being the arity of argument stacks in the specification. In its present state, DELILAH's syntax comprises about ten rules *per* direction.

1.4 THE LOGIC AND THE ALGEBRA OF LISTS, FLAGS, TYPES AND MODES

1.4.1 Categories and complex symbols

A category in a categorial constructive unification grammar like the one presented here has many tasks to fulfill. First of all, it embodies an agenda according to which the rules of grammar – the modalized instances of the rule of cancellation and composition – can operate. Secondly, it predetermines linearization of phrases in a sentence: the structure of the category imposes strict conditions on linear order. Thirdly, it defines equivalence classes of phrases

with respect to the grammar. And, fourthly, it represents the structure of lexical templates with respect to unification. As an example of this multi-tasking, suppose that the grammar-in-action, say: the parser, runs into the following category:

(26) pp\0~[n^0]/1~[np^0]

It tells the parser that at least two other phrases have to be found for this category to be satisfied, that one is a noun-type phrase to occur to the left of the other, a phrase with a noun-phrase-type of distribution, that the phrase carrying the category is neither lexical nor completely saturated, that the template which it encodes has two sub-templates which are subject to two different unification acts, and that for the categorial hypothesis – if not *pp* – to be met, at least one prepositional phrase must occur as an argument. Thus, the category broadcasts information essential to grammatical dynamics.
In categorial grammar, categories carry *all* combinatory information feeding into the grammar. For this reason, the categorial formalism is the quintessence of the grammatical message. In exactly that sense, categorial grammar is mathematical: grammar amounts to the manipulation of symbols according to an established protocol, which as such may reflect fundamental theses on the nature of the game. Unfortunately, the grammar we present here deviates from more or less canonical categorial set-ups, established *e.g.* by Moortgat (1988), Van Benthem (1991), Morrill (1994) and Carpenter (1997). On the other hand, more recent developments of the categorial practice in Moortgat (1998), and Baldridge and Kruijff (2003) indicate some convergence with the proposals of Cremers (1993a), which underlie the system presented here, in particular with respect to fine-grained modalities for composition. Still, the formalistic nature of any categorial approach to natural language requires an in-depth account of its architecture. That is what we are trying to achieve in this section.

It is of some importance to see that not all distributionally relevant aspects of words and phrases can be encoded in the combinatory category, and that a certain sense of arbitrariness is unavoidable. For example, the notion of *np* underspecifies some distributional aspects of the class of nominal constituents, for example with respect to case, number and definiteness. What, then, is the difference for a phrase between being nominal and being plural? The answer is relatively clear. In all contexts, being nominal is a required property – a phrase cannot be underspecified for category and still be rightfully selected; being plural may or may not be relevant for the combinatorics, depending on contextual requirements. This difference is a good reason to leave number to

the unification process, which is context-dependent by definition, and which leaves room for all levels of specification. In the same vein, certain less classical distinctions may make it to the combinatory level. In Dutch, for example, non-finite verb phrases may occur with and without *te*, a preposition-like and meaningless particle. Selection of verb phrases is sensitive to this particle: *willen* 'to want' takes only *te*-less vps as a complement, whereas *proberen* 'to try' exclusively selects vps headed by *te*. In no situation is *te* irrelevant. As a consequence, it is combinatorially relevant to mark a vp for having *te*. In the DELILAH grammar for Dutch, we thus have two distinct atomic types: *vp* and *vpt*, for it is no use having unification decide – costly – on predictable combinatory properties.

Also, word order variation may have to be encoded at the level of categories. Again, the criterion resides in selection: if the internal word order of a constituent determines its candidacy for combinatory processes, that word order is better reflected in category labeling. Now, Dutch is a verb-second language, and the position of the finite verb determines the status of the sentence. Thus, we have distinct sentential categories for distinct positions of the finite verb.

The set of categories that one needs can only be determined empirically – by experimenting with how to get a semantic grammar to work properly, *i.e.* to interpret all and only the well-formed strings over a lexicon. Yet, we do not abandon the idea, inherent in the Polish origins of categorial grammar, that there is a class of elementary types – typically two: one for names and one for sentences – and that other categories denote functions over the domains of these elementary types. That is, we assume that all syntactic categories denote (complex or composite) functions over elementary domains, in short: that nps are of type *ett*. The (semi-)compositional semantics of our system is essentially *typed* (see chapter 2). We do not assume, however, that the syntax of Dutch must be grafted on to the internal typological structure of the categories needed. The reason for this divergence between syntactic and semantic labeling is typically pragmatic: there is no bijection between types and combinatory classes. This point was already made clear in Montague (1972) – though the motivation there was not syntactic subtlety. It is a fact of life that not every syntactic property is determined by the typological structure; many, maybe even most, are not. Often, the syntax does not exploit the semantic fine-structure of category: as was illustrated above, verb-placement is syntactically big business but it is semantically vacuous. This type of tension between combinatory and semantic interests makes the grammar of natural languages worth pursuing.

The manipulation of categories, to be pursued in this section, has one single goal: arranging the unification of complex symbols – or *signs* – in such a way that the right interpretation ensues. All interesting information – including the categories steering the process – is in these complex symbols and is treated conservatively in the unification process. The syntax as such does not add to that information. It just controls it. This control, however, is the only claim to effectivity and efficiency we can put forward. The ambiguity of natural language is overwhelming. Unification is too costly to have search space reduced while we are waiting. It is the syntax that has to select and prepare felicitous analyses; it is the syntactic routine that makes language work.

1.4.2 Basics

The syntax depends on three well-defined sets:

(a) a finite set *Types* of types
(b) a finite set *Mods* of modalities
(c) a set *Cats* of categories

The set Types consists of literals denoting syntactic and semantic equivalence classes over lexical phrases. The only restriction imposed on this set is finiteness. At least one of the types is marked for sentencehood. One may reduce the number of sentential types to one, but language calls for distinguishing propositions from questions and imperatives. A typical set of types may consist of literals like `s` for propositional sentences, `q` for questions, `np` for names and nominal quantifiers, `vp` and `ip` for tensed and untensed verb phrases, and so on. Clearly, the distribution of types over the grammar expresses our insight into the phrasal tiers, but the types themselves lack any formal content.

The set Mods of modalities is also taken to be a set of literals disjoined from Types. Modalities stand for modes of merging categories. They are defined pointwise. Since the number of merge modes will turn out to be finite, Mods is finite too.

The set Cats of categories holds the basic data structures of the grammar. Every category is a triplet <Head, LeftArguments, RightArguments>. The head is a single, bare member of Types.
LeftArguments consists of a pair <Flag, LeftArgumentList>; Flag is a label, holding information as to the present state of LeftArgumentList. RightArguments is constructed accordingly. Both the left and the right argument lists consist of

a finite number – possibly zero – of pairs <Type, Modality> in Types × Mods. The left arity of a category is the number of pairs <Type, Modality> in LeftArgumentList. Equally, the right arity of a category is the number of modalized types in the RightArgumentList. The arity of the category plain is the sum of its left and right arities. Basically, a category should be seen as an *n*-place function mapping categories onto categories, with *n* reflecting its arity.
For notational convenience, categories are written in the format

(27) Head\ LeftFlag~LeftArgumentList / RightFlag~RightArgumentList.

Non-empty argument lists are written *[Type1^Mode1, Type2^Mode2, ...]* or *[Type1^Mode1|Rest].*
This format of categories is deducible to the flat types of Buszkowski (1982). The only element added is treating all the arguments at one side of the head as one object: a list or a stack. For this defining property of our categories, we call the resulting grammar *Categorial List Grammar* or CLG.

Every phrase in the language that has to be defined is assigned to one or more categories by the lexicon or by the grammar. Here is an example from Dutch. The article *de* ('the') is lexically assigned to the category `np\u~[]/u~[n^i]`. Its left arity is zero; its right arity is one. The category expresses that its phrases are of type `np` if combined with phrases of type `n`. Combining with a phrase equals merging with a category to which that phrase is assigned. It will operate according to merge mode *i* on a category headed by *n.* If mode *i* allows operating on *e.g.* `n\u~[]/u~[]` and the noun *duivelsuitdrijver* ('exorcist') is lexically assigned to that category, the grammar may assign the phrase *de duivelsuitdrijver* to the category `np\u~[]/a~[]`. The modalized argument *n* in the original category for *de* has been cancelled, and RightFlag adapted. The category thus specifies the nature of categories to be operated on by listing heads in the argument lists. The associated modalities convey the conditions under which this merge may unfold in the modalities. Furthermore, the category stores concise information as to its merge history in the flags at the argument lists. Merging categories is therefore a complex, asymmetric operation.

Merging categories is the only combinatory operation in the syntax. It was originally introduced in Cremers (1993a). The operation is a generalization of Generalized Composition for Combinatory Categorial Grammar, as it was framed by Joshi *et al.* (1991) to capture the grammar engines constructed by Steedman (1996, *e.g.*).

In Combinatory Categorial Grammar, Generalized Composition is the main engine of analysis; here are its two instances as presented by Joshi *et. al.* (1991).

(28) $x/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(x|z_1)|z_2 ... |z_n)$
(29) $(...(y|z_1)|z_2 ... |z_n) \quad x\backslash y \Rightarrow (...(x|z_1)|z_2 ... |z_n)$

x/y and *x\y* are considered to be the primary category of the compositions (28) and (29), respectively; the other one at the left of ⇒, headed by *y*, is called the secondary category. Every occurrence of $|z_i$ is a unit, representing either $\backslash z_i$ or $/z_i$. Directionality of arguments is not affected by composition (Steedman 1990). Generalized Composition cancels the argument */y* in the primary category against the secondary category's head *y* and yields the remaining structure *x* of the primary category with all the arguments of the secondary category stacked on top of it.

Categorial List Grammar restricts Generalized Composition by requiring the cancelled type *y* to be primitive or atomic, i.e. without internal structure. This restriction is referred to as *linearity*. In the terminology of König (1990), the resulting grammars would be characterized as first-order grammars, since there is only one relevant level of type embedding. A type is either a head or an argument to that head. Hepple (1996) describes a linearization procedure as a compilation of higher-order categorial grammars for parsing. Moreover, the syntax presented here will take into consideration the full internal structure of type *x*, which is not specified in Generalized Composition. As a consequence, Categorial List Grammar extends generalized composition form (28) to two different operations:

(30) $(...(p|w_1)|w_2...|w_m)/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(...(p|z_1)|z_2 ... |z_n)|w_1|w_2... |w_m)$
(31) $(...(p|w_1)|w_2...|w_m)/y \quad (...(y|z_1)|z_2 ... |z_n) \Rightarrow (...(...(p|w_1)|w_2... |w_m)|z_1|z_2 ... |z_n)$

Of these, the second is a more explicit version of (28). The first form of composition is not covered by Generalized Composition, because there, the primary category's head and arguments cannot be separated. Furthermore, Categorial List Grammar will split up both the secondary and the primary category with respect to the directionality of arguments. CLG then collects the set of arguments in each direction. Consequently, the extension will be split up again into four different modes of composition. They are only distinguishable as to the relative orderings of the directed arguments in the consequent when compared to their sources (graphical marking is just for convenience):

(32) $\mathbf{p}\backslash pl_1...\backslash pl_n\ /\mathrm{pr}_1.../\mathrm{pr}_m/\mathbf{s}\ \ \mathbf{s}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k/sr_1.../sr_l \Rightarrow$
$\mathbf{p}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k\backslash pl_1...\backslash pl_n/sr_1.../sr_l/\mathrm{pr}_2.../\mathrm{pr}_m$

(33) $\mathbf{p}\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k\backslash pl_1...\backslash pl_n/\mathrm{pr}_1.../\mathrm{pr}_m/sr_1.../sr_l$

(34) $\mathbf{p}\backslash \mathrm{pl}_1...\backslash \mathrm{pl}_n\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k/\mathrm{sr}_1.../\mathrm{sr}_l/\mathrm{pr}_1.../\mathrm{pr}_m$

(35) $\mathbf{p}\backslash pl_1...\backslash pl_n\backslash \mathrm{sl}_1...\backslash \mathrm{sl}_k/\mathrm{pr}_1.../\mathrm{pr}_m/sr_1.../sr_l$

By simple computation, CLG extends every instance of Generalized Composition to eight patterns. The properties of these patterns are discussed in the remainder of this section. In the formatting of CLG below the directed arguments of a category are bundled into two single objects, ordered lists or stacks. In order to keep the composition or merge of categories functional, a finite number of distinct modes of composition will be distinguished.

1.4.3 Merge

Merge basically sends pairs of categories to categories, and therefore resides in the functional space $C^{C\times C}$. It consists of four sub-operations:

(36) cancelling a type in a pair <Type, Mode> at the top of an argument list of one category against the head of the other
(37) appending two pairs of argument lists
(38) reflagging the resulting argument list
(39) constructing a new category from the components

The whole operation is subjected to one out of a finite set of modalities, the one that is associated with the type to be cancelled.

Here is an example of a merge of two categories C1 = `s\u~[pp^isl]/u~[vp^rais]` and C2 = `vp\u~[np^isl]/a~[vp^cons]`. The only possible cancellation involves the type `vp` in the righthand argument list of C1 and the `vp` head of C2, under the modality `^rais`. The cancellation can be effected when C2 is the right member of the pair of input categories and the internal structure of C1 and C2 complies with the constraints defined by `^rais`. Suppose that both conditions are met. If we indicate the asymmetric merge with the infix ⊗, the merge could look like

(40) C1 ⊗ C2 → C3

```
s\u~[pp^isl]/u~[vp^rais] ⊗ vp\u~[np^isl]/u~[vp^cons] →
                                    s\u~[np^isl, pp^isl]/a~[vp^cons]
```

In this merge, the left-argument lists are appended in such a way that the argument list delivered by C2 is prefixed to the left-argument list stemming

from C1. Hereby, the order of future cancellation is fixed. At the right-hand side, only the argument list of C2 survives, as the right-argument list of C1 is emptied of its only argument after cancellation. The flag `a~` at the new list indicates that it was rightward cancelling that brought about this merge: the right-hand argument list is the affected one.

Reversing the point of view, one may now define what exactly are the constraints imposed by the `^rais` mode. In fact, two different patterns have to be stated, depending on the direction of the cancelling. The particular merge mode imposed by `^rais` is indicated by the modalized merge operator $\otimes_{\text{rais}}$. This convention is also used by Cremers (1993a) to introduce modes of application, by Moortgat (1998) to describe Multi-Modal Categorial Logics, and by Baldridge and Kruijff (2003) to represent Multi-Modal Combinatory Categorial Grammar. In terms of the last, the grammar of Cremers (1993a) might be described as multi-modal combinatoric.

Since merge is essentially asymmetric – this item is addressed in section 1.5.3 below – the process discriminates systematically between the two input categories. The category that has an argument at the top of one of its lists cancelled is dubbed the *primary* category. Its head is *Prim*, and all its others components are preceded by a *p*. The other components are identified by *f* and *a* for flags and argument lists, respectively, and by *r* and *l*, for left and right sides, respectively. The other category is doomed to be the *secondary* one, with head *Sec* and *s*'s instead of *p*'s. Finally, asymmetric append is marked by ⊕. Here then are two possible instances of the merge mode `^rais`. Capital onsets indicate variables.

(41) $(\otimes_{\text{rais}} /)$
$\text{Prim}\backslash\text{Plf}{\sim}\text{Pla}/\text{Prf}{\sim}[\text{Sec}\hat{}\text{rais}|\text{Pra}] \;\otimes_{\text{rais}}\; \text{Sec}\backslash\text{Slf}{\sim}\text{Sla}/\text{Srf}{\sim}\text{Sra} \;\rightarrow$
$\text{Prim}\backslash\text{Slf}{\sim}\text{Sla}\oplus\text{Pla}/\text{a}{\sim}\text{Pra}\oplus\text{Sra}$

(42) $(\otimes_{\text{rais}} \backslash)$
$\text{Sec}\backslash\text{u}{\sim}\text{Sla}/\text{Srf}{\sim}\text{Sra} \;\otimes_{\text{rais}}\; \text{Prim}\backslash\text{Plf}{\sim}[\text{Sec}\hat{}\text{rais}|\text{Pla}]/\text{Prf}{\sim}[\,] \;\rightarrow$
$\text{Prim}\backslash\text{a}{\sim}\text{Pla}\oplus\text{Sla}/\text{Srf}{\sim}\text{Sra}$

Prim: head of *prim*ary category; *Plf:* *p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra:* *s*econdary's category's *r*ight *a*rgument list

The first of these two merge modes does not impose any constraint on the input categories apart from the presence of cancelable types. As for the output, it specifies prefixing of the secondary argument list at the left-hand side and the reverse at the right-hand side. The flag `a~` at the right-hand side marks

affectedness of this argument list. The flag at the other side is provided by the secondary category.
Unlike ($\otimes_{rais}$ /), ($\otimes_{rais}$ \) does impose restrictions on the structure of the input categories. It requires the left argument list of the secondary category to be unaffected up to then, and specifies emptiness for the right argument list of the primary category. In its output specifications it essentially behaves like its rightward dual, *modulo* directionality.

Here is a sample instantiation of ($\otimes_{rais}$ /). The Dutch verb *willen* ('to want') is a typical verb raiser. It selects, among other things, infinitival complements and adjoins to the left of their verbal head. Consider one of its finite forms, for example the singular past tense form for embedded, *i.e.* verb-final, sentences: *wou*. This form introduces a subject argument it agrees with to its left, and an infinitival complement to its right. It would typically be assigned to the category `s_emb\u~[np]/u~[vp^rais]`. Both lists are marked zero, as the category originates from lexical assignment. Take, furthermore, the verb *gaan* ('to go'). One of its combinatorial options is to create a vp by selecting a directional pp to its left. Thus, it will be assigned lexically to the category `vp\u~[pp]/u~[]`; the directionality of the pp is expressed in the template of the verb, as a feature. The merge mode ($\otimes_{rais}$ /) specified above yields the following composition of these two categories, deriving a new category for the string *wou gaan* ('wanted to go'):

(43) `s_emb\u~[np]/u~[vp^rais]` $\otimes_{rais}$ `vp\u~[pp]/u~[]` → `s_emb\u~[pp, np]/a~[]`

In compliance with the format for $\otimes_{rais}$, the left argument list of the secondary category, here consisting only of the argument `pp`, is appended as a prefix to the left argument list of the primary category. By implication, the resulting category will have to cancel `ppdir` before `np`. This reflects the state of affairs where the subject of *wou gaan* is more peripheral to that phrase than to its directional argument.

1.4.4 Modalities

It is by no means necessary that every modality be specified both for rightward and for leftward cancellation. Given the nature of the example (43) for ($\otimes_{rais}$ /), it is even unlikely that the grammar of Dutch would give rise to ($\otimes_{rais}$ \). Merge is inevitably asymmetric. One head is cancelled, the other head persists. The argument lists in one direction are unaffected, those in the other direction lose

an argument. For each direction, the unfolding or execution of one argument list will be suspended, while the other is readily available for cancellation. Since linearity matters in natural language, directional duals for merging modes will be the exception, rather than the rule.

Plural specifications in one direction are possible. They should address disjointed sub-domains of Cats × Cats, though, as merge is functional. Nevertheless, it can be proven that the set of expressible merges, and thereby the set of merge modes, is finite. To see why, consider first this characterization of an expressible merge. A merge aims at the cancellation of exactly one type at the top of one argument list and does not refer to other types in that or other argument lists. It specifies the heads of two categories and exactly two appends of argument lists. It specifies two flags at most to these lists. Therefore, the gamma of specifications by a merge mode is rather limited. Moreover, all the components of a merge come from limited sets or classes of suboperations. First, the set Types is taken to be finite, in accordance with the standard condition in formal grammar that both the terminal and the nonterminal alphabet be so. As for input conditions, merge modes may specify for each of four argument lists whether they are empty or nonempty, and may specify flags for these lists. The number of different modes is very limited, by definition. As for output conditions, a merge mode can only specify argument lists that were part of the input. There are only two combinatory possibilities for constructing new argument lists out of these, resulting from the asymmetry of append. No other operations on argument lists are definable. The number of flags to specify for the new lists is as restricted as it was at input. All components of merge appear to stem from restricted sets. Consequently, the number of different combinations is bound to be finite, and so is the set Mods of merge modes.

1.4.5 Argument list

The only operations defined on argument lists are cancellation of an argument according to the modality that comes with it and appending. The latter creates a new list but leaves the lists that are feeding it intact. One could think of alternatives to this rigid form of list construction, like mixing or popping:

(a) mixing: [a,b] ∇ [d,e] = [a,d,b,e]
(b) popping: [a,b] ↵ [d,e] = [d,a,b] ↵ [e] = [e,d,a,b]
(c) appending: [a,b] + [d,e] = [a,b,d,e]

Mixing is sometimes referred to as list merge; for obvious reasons, the term mixing is preferred here. Lists are linearized, and thus they are ordered structures. Specifying append as the operation to be performed while merging categories implies conservation of these structures with respect to adjacency and precedence. Mixing respects linearity but not adjacency. Popping respects adjacency, but partially reverses linearity. As a matter of fact, appending is the only operation that respects both the local characteristic of adjacency and the global characteristic of precedence.

Comparing mixing, popping and appending, some analogies are straightforward. Popping reflects the way stacks combine. A stack is loaded and emptied in a regular fashion, element by element, by simple repetition of the same manoeuvre. Mixing is a context-sensitive operation. It requires keeping track of former moves and accounting for the internal structure of the lists involved. The analogy here is that of combining agendas, *i.e.* tasks that have to be performed in a certain order, feeding and bleeding each other. Appending, then, is in-between. It requires recursion, *i.e.* going back to a bottom case while suspending the shifts of individual elements. No bookkeeping of the internal structure of the lists involved is necessary, and it is context-free (cf. section 1.8). If append turns out to be the canonical mode of merging argument sequences, this operation alone would already fall in with the need for recursion in natural languages, as has been argued by Hollebrands and Roeper (2007). Although we adhere to append as the proper mode of merging, we will use both of the terms *stack* and *list*, as they both reflect the essential property of complete ordering. Moreover, taking argument lists to be the stores that govern cancellation of types, we will address argument list as the grammar's *agendas*.

The conservation of adjacency and precedence in the local environment introduced by a category is the reason for considering append as the structural operation underlying merge. Categories to which lexical items are assigned reflect partial knowledge of the combinatoric potency of that item. That knowledge concerns selection and subcategorization, but also linearization of the selected items. Moreover, it can be seen as constituting the domain governed by the phrase assigned to this category. This configuration is not accidental but defines the whereabouts of the phrase's interpretation. The merging of categories is the engine of phrase combinatorics in this grammar. As such, it accounts for the discontinuities natural language abounds in. But merging would be self-destructive if it were to come with the mutation or covering of the configurations that make the phrases which it combines meaningful and

interpretable. Grammar is supposed to make sense of the intrinsicalities of natural language. Appending the identifying domains of phrases, then, seems to be the more harmless option in merging categories.

1.4.6 Deduction schemes

The grammar outlined here can be framed into a sequential, deductional format. Here is a model. Types are indicated by lower case letters, lists of arguments by indexed L_i and R_j. Categories are marked by Roman capitals A, B, C. Potentially empty sequences of categories are marked by distinct Greek capitals.

(44) Axioms

unary axiom: $C \rightarrow C$

binary axiom: $A \otimes_i B \rightarrow C$ *for all* $\otimes_i$ *defined*

rightward: $a\backslash L_1/[b\hat{}i|R_1] \otimes_i b\backslash L_2/R_2 \rightarrow a\backslash L_1 \oplus_i L_2/R_1 \oplus_i R_2$

leftward: $b\backslash L_2/R_2 \otimes_j a\backslash [b\hat{}j|L_1]/R_1 \rightarrow a\backslash L_1 \oplus_i L_2/R_1 \oplus_i R_2$

(45) Rules

left rule $\dfrac{\Delta \rightarrow B \qquad A \otimes_i B \rightarrow C \qquad \Gamma', C, \Gamma'' \rightarrow T}{\Gamma', A, \Delta, \Gamma'' \rightarrow T}$

right rule $\dfrac{\Delta \rightarrow B \qquad B \otimes_i A \rightarrow C \qquad \Gamma', C, \Gamma'' \rightarrow T}{\Gamma', \Delta, A, \Gamma'' \rightarrow T}$

Note that the rightmost premise is shorter than the conclusion, in that its antecedent contains at least one category less than the conclusion's antecedent. Note furthermore that in all binary axioms the arity of the consequent is exactly one less than the sum of the arities in the antecedent, which is necessitated by the definition of merge as involving cancellation. It follows that the arity of the right-hand premise is smaller than the arity of the conclusion. Clearly, then, the decidability of this calculus depends on the decidability of the term $\Delta \rightarrow B$. We prove its decidability by induction on the length of Δ. $|\Delta| \leq |\Gamma' \Gamma'|$, by definition. If $|\Delta|=1$, the term instantiates identity or it is false. If $|\Delta| = 2$, its components satisfy, following the rule format, one of finitely many binary axioms, or it is false. If $|\Delta| > 2$, the rules apply to create a left-hand premise $\Delta' \rightarrow B'$ such that $|\Delta'| < |\Delta|$. Given the remarks on the decidability of the middle and right-hand premises in the rules, the derivability of each proposition $\Gamma \rightarrow t$ for some designated type t can thus be deduced in a finite number of steps.

For the type of grammar presented in (44) and (45), an important invariant can be established. The invariant is a simplified version of the count-invariancy that Van Benthem (1991) uncovered for the Lambek-calculus and related systems. This invariant states that there is a certain way of counting primitive types, such that theorems of that calculus of the form $\phi \rightarrow \psi$ invariantly have the same count at both sides of the arrow. Since categories in the Lambek-calculus are more complex than in our grammar, the metric is simpler too.

(46) For each type in a category C,
count an occurrence of t as +1 if it is the head, and as -1 if it is an argument,
determine t-count(C) as the sum of the occurrences of t in C,
for each sequence of categories, determine the t-count for that sequence as the sum of the t-count for the categories in the sequence.

Given this metric, the simple fact that cancellation is a zero-sum operation, and the observation that no types other than the one cancelled are affected in an axiom, the following proposition is evident:

(47) *Count Invariance*
For each axiom $\phi \rightarrow \psi$ in (44) and for every type t, t-count(φ) = t-count(ψ).

As an immediate consequence, the rules in (45) are conservative in the sense that all the operations above and below the deduction line respect Count Invariance. In particular, the conclusion and the major premise share t-counts. Thus, the grammar enjoys *deductive monotony:* no types appear or disappear apart from zero-sum cancellation.

1.4.7 The algebra of strings

The categories, and the operations defined on these, can be interpreted on a string model <L*, +>. L is the set of atomic phrases of a language and L* is Kleene closure: the set of non-empty strings over that language such that $L \subseteq L^*$ and for all $a, b \in L^*$, $a+b \in L^*$. The string operation + is taken to be associative and noncommutative. It gives rise neither to idempotency nor to persistency. As such, it is, as Morrill (1994) states, an operation on pieces of matter rather than an operation on pieces of knowledge. Another image that comes to mind when looking for interpretations is the flow of time. Language essentially extends in time. Phrases can be seen as decorated intervals. Although time may be dealt with under all kinds of logics, there is a standard perception of the flow of time as a noncommutative, *i.e.* irreversible, and nonpersistent, *i.e.* segmentable, process. The operation + is meant to reflect this analogy.

Consider furthermore the power set 2^{L^*}. A mapping $\mathfrak{R}$: Cats $\cup$ Types $\rightarrow 2^{L^*}$ has to be established, assigning to a category or type a set of strings *of* that category or type. This is the standard interpretation of a category as a class of strings with equivalent combinatorial behaviour. In presenting categories for this purpose we will, for the moment, abstract from flags and modalities. The operator $\bullet$ indicates asymmetric associative concatenation of categories; $\circ$ interprets this relation on categories.

(48) $\mathfrak{R}(A) = \mathfrak{R}\ (A\backslash[\ ]/[\ ])$ for all $A \in T$
$\mathfrak{R}([A_1,\ldots,A_i,\ldots,A_n]) = \mathfrak{R}(\ A_1\bullet\ldots\bullet A_i\bullet\ldots\bullet A_n\ ) =$
$\mathfrak{R}(A_1)\circ\ldots\circ\mathfrak{R}(A_j)\circ\ldots\circ\mathfrak{R}(A_n) = \{\ a_1+\ldots+a_i+\ldots+a_n \mid a_j \in \mathfrak{R}(A_j),\ 1 \leq j \leq n\ \}$
$\mathfrak{R}(A\otimes_m B) = \{\ a+b \mid a \in \mathfrak{R}(A),\ b \in \mathfrak{R}(B)\}$, for all m
$\mathfrak{R}(A\backslash x\sim L/y\sim R) = \{\ a \mid \forall l \in \mathfrak{R}(\text{reverseL}),\ \forall r \in \mathfrak{R}(B)\ l+a+r \in \mathfrak{R}(A)\ \}$ (the reverse of L is needed here as the order of the list of arguments to the left is the reverse of the order of strings associated with that list; flags of argument lists do not interfere with the denotation of categories.)

The syntax established here gives rise to a form of category merging that has been dubbed *mixed* or *disharmonic* composition. Its characteristic feature is that arguments from the secondary category not belonging to the direction that is affected by the cancellation are taken over by the new category. For example, if *SLA* below is non-empty, the indicated merge will involve this disharmonic composition. (The + operator amounts to standard *append*.)

(49) PRIM\PLF~PLA/PRF~[SEC^j|PRA] $\otimes_j$ SEC\SLF~SLA/SRF~SRA $\rightarrow$
PRIM\SLF~SLA+PLA/a~SRA+PRA

In a slightly more transparent but less general notation, disharmonic composition is exemplified in the following lines:

(50) x/y y\z $\Rightarrow$ x\z (rightward cancelling, leftward composition)
y/z x\y $\Rightarrow$ x/z (leftward cancelling, rightward composition)

The particular appending of *PRA* and *SRA* in (49), on the other hand, is called *homogeneous* or *harmonic*. Here, the arguments entered by the secondary category, are of the direction affected by the cancellation. They are placed on top of those entered at that side by the primary category. Here are the simplified representations for harmonious composition:

(51) x/y y/z $\Rightarrow$ x/z (rightward cancelling, rightward composition)
y\z x\y $\Rightarrow$ x\z (leftward cancelling, leftward composition)

The main combinatorial difference between harmony and disharmony in this respect is the following. If a string is constructed with harmonious composition, there may be an alternative derivation without transfer of arguments; if a string is constructed by disharmonious composition, there is no alternative to that derivation. Again, this is illustrated in the alternative notation, but only in one direction. In (52), there are two ways to merge three categories into one: one involving (harmonious) composition of the two leftmost categories, the other using sheer cancellation twice; for the disharmonious composition in (53), there is no alternative.

(52) x/y y/z z ⇒ x/z z ⇒ x
x/y y/z z ⇒ x/y y ⇒ x
(53) z x/y y\z ⇒ z x\z ⇒ x

Consequently, disharmonious composition allows for grouping of strings that are discontinuous to such a degree that it cannot be solved by the present categories. That is: the Lambek-calculus cannot map the categories in (53) to other categories that can be reduced to a single type without resorting to disharmonious composition. Since the Lambek-calculus has been proven to be weakly equivalent to context-free grammars (Pentus 1993), the use of disharmonious composition pushes a categorial grammar beyond context-freeness. As a matter of fact, the syntax of merging listed first-order categories under modalities induces several distinct patterns of discontinuity; these will be discussed in the sections on the syntax of Dutch. The syntax (of Dutch) is one of discontinuity, rather than a syntax of gluing constituents. The basic combinatorics that is envisaged for interpretable strings is chaining rather than appending. It is designed not so much for combining strings *w* and *z* into *wz* but for combining strings *wx* and *zy* into *wzxy* or *zwyx* in a controlled manner. Such forms of string merge are immanent to natural language: strings chain each other rather than just glue together. They bubble up between each other's edges in a variety of modes. Nevertheless, we can have resort to + as the designated operation on strings, since no operation of the grammar violates the integrity of a deduced string. Merging prompts strings for peripheral association only. It does not involve any kind of extraction from or insertion into otherwise constructed strings. The discontinuous effects are solely caused by reference to the internal structure of the category that is associated with a string, *i.e.* to which a string is assigned. Once a string is derived by merge, it is maintained in the rest of the derivation. In this respect, the syntax is conservative and monotonous.

By consequence, the general strategy for constructing *ywzx* as stemming from *wx* and *yz* is mandatory. Suppose $wx \in \mathfrak{R}(A)$ and $yz \in \mathfrak{R}(B)$ for some A and B,

but there is no C such that *wz*, *yw* or *zx* ∈ ℜ(C). That is what it means to say that *ywzx* stems from *wx* and *yz*. The only derivation in the grammar depicted above runs as follows. Construct *w* and put the construction of string *x* on the agenda. Construct *z* and put the construction of string *y* on the agenda. Merge *w* and *z* and execute the agenda.
Now consider how under this regime a string *wyzx* must be derived from the same formants and with the same labour division between *w* and *x*. The string *wz* cannot be constructed, not even if for some C *wz* ∈ ℜ(C), since no rule of grammar could split it up for *y* to be inserted. Therefore, the string *yz* has to be formed first. Next, *wyz* can be derived, while putting *x* on the agenda.

1.4.8 Disharmonic composition and categorial logic: grammar and reasoning

As indicated above, CLG comes with disharmonic composition. In the present setting, disharmony results from appending a non-empty argument list of the secondary category in the passive direction, *i.e.* the direction that is not affected by cancellation – *Sla* in merge scheme (49). It has often been argued that categorial grammars involving disharmonic composition are beyond string models. Disharmony is rejected in, *e.g.*, Carpenter (1997), Morrill (1994) and Jacobson (1991). The last states that disharmonic or mixed composition, though attractive for dealing with certain phenomena, cannot be function composition. If *f* sends *a* to *ba* and *g* sends *c* to *cd* and *cd* is in the domain of *f*, then *f*∘*g* sends *c* to *bcd*, as Jacobson correctly claims for functions *f* and *g*. If the merge of categories A and B does not have this functional effect, it cannot be functional. To put it bluntly: for every argument c, the composition applied to the argument – *f*∘*g(c)* – is bound to be equal to the subsequent application of the rightmost function to the argument and the application of the leftmost function to the result of the innermost application – *f(g(c))*. In disharmonic merge, this is typically not the case. If disharmony is called for, there is no alternative to it, as was argued in the preceding section.
This argument against the functional nature of disharmony takes the directionality of categories seriously: it defines the functionality of categories in terms of linearization of strings. On the other hand, functions are not necessarily directed objects. The functional interpretation of categorial deduction under the Curry-Howard correspondence (cf. Van Benthem 1986) abstracts from directionality. As a matter of fact, this is the main reason why the correspondence is not an isomorphism over substructural logic – intuitionistic logic with selective use of structural rules – and constitutes a particular fragment

of the lambda calculus. *Merge*, as defined above, can be seen as a complex of operations, some of which deal with linearization, while others take care of the logical aspects. The linearization operation – *append*, basically – does not contribute to functionality, although *append* itself is a homomorphism in $L^{L\times L}$, where L is the class of finite lists over a given domain. The grouping or bracketing operation that comes with *merge* is as functional as it can be.

Morrill (1994: 231-232) refutes a syntax which applies selective linearization schemes: '… then the theory of syntax is not logical, in the sense of being the reflection of an interpretation of category formulas, but … a deductive system receiving definitions in terms of non-logical axioms and rules.' The point here is not merely that one could choose, operationally, between redefining 'existing' operators and adding new ones, the latter being common practice in Multimodal Categorial Logic (Moortgat 1998). The crux is the nature of the relationship between logic and grammar. In Morrill's understanding, logic starts out with irrefutable axioms, and the grammatical combinatorics reflect this reasonable base. The basic logic here is intuistionistic conditionalization. It defines three operators (left division, right division and noncommutative multiplication) in a multiplicative fashion by residuation:

(54) $a \rightarrow c \,/ b$ *iff* $a \cdot b \rightarrow c$ *iff* $b \rightarrow c \backslash a$

One could also opt for nondirected commutative operators, according to

(55) $a \rightarrow c | b$ *iff* $a \cdot b \rightarrow c$ *iff* $b \cdot a \rightarrow c$ *iff* $b \rightarrow c | a$

Other operators can be defined, also in duals, in terms of residuation. The one-place operators ◊ and □ are famous by now (see Moortgat 1998, Morrill 1994):

(56) $\Diamond a \rightarrow b$ *iff* $a \rightarrow \Box b$

Their multiplicativity can be compared to the duals *n-root* and *n-power* defined on the positive reals: $\sqrt{a} = b$ *iff* $a = b^2$.
For deduction, division is interpreted as implication and multiplication as co-occurrence. The one-place operators add modal control, by closing and opening – basically, marking and unmarking – (sequences of) categories.

From a logical point of view, an important aspect of this system is that it lacks negation, or its algebraic counterpart, complementation. The string alge-

bra has no zero-element Zero such that concatenating any String with Zero always yields String. Suppose Zero existed. Its category should be of type x/x and x\x, for all categories x. Zero would be a homomorphism over all categories. Any string String would be combinatorially and materially equivalent to $Zero^n$+String+$Zero^m$, and identity of strings would not be decidable – note that the algebra of strings is resource-sensitive. Therefore, no combinatory object in grammar denotes the empty string.

The absence of complementation in the logical basis to the categorial analysis can be well defended under reference to the nature of language combinatorics. Language is a material system, and its grammar deals with real objects; it cannot be unwound. Its combinatorics are, in intuitionistic terms, *resource-sensitive*. It irreversibly consumes time and space, and it is hard to see what operation in grammar could cover algebraic complementation or logic negation.

In the same vein, it is worth asking what aspect of natural language makes conditionalization a suitable vehicle to guide combinatorial operations. In 'classical' flexible categorial grammar (Lambek 1958, Moortgat 1988 and subsequent work) hypothetical reasoning is the fuel for the type-changing operations that drive deduction. In the calculus, deductive hypothetical reasoning is reflected in so-called *introduction* rules. These rules increase categorial complexity and cover – among others – one of the greatest discoveries in the history of linguistics, the generalizability of quantification (Montague 1972, Barwise & Cooper 1981):

(57) e → <<et>t> *or*: whenever you run into a primitive entity, you had better take a structured set of sets of entities

Its proof in elementary categorial logic involves the assumption that *e* is brought to *t* in the presence of *<et>*, to wit: *e <et>* → *t.* The type transition (57) withdraws the assumption that *<et>* is available. It is the same hypothetical reasoning as in classic natural deduction, but with the intuitionist proviso that assumptions and withdrawals are one-to-one.

What makes hypothetical reasoning so attractive in grammar is the previously acknowledged insight that language is a material, resources-consuming system. To say that a certain element requires another or is conditioned by another is saying that somewhere in the linguistics space (*e.g.* a sentence) that other element must be available. *Somewhere*, however, is far too weak. It never means *anywhere*. Generally, the requirement is much more specific: the other element must be available in a certain well-defined subspace of the lin-

guistic space. This subspace is identified relatively, with respect to the position of the requiring element, and in linear terms: to the right or to the left of that element. Relative linear occurrence is the language instance of logical conditionalization. In natural language, linear occurrence amounts to conditionality. It is by no means accidental that direction is part of the logical basis for natural-language analysis. Directionality is the name of the language game. And it is from directional requirements that we derive hierarchy, not the other way around: such is the message of Kayne (1994). We are still far from understanding exactly how linearity induces subordination and semantic dependency, but linearization is indispensable in the grammar base.

This being the case, we have at least two data structures for encoding the linearization requirements of phrases, *i.e.* their spaced conditionalizations.
First, there are the types exploited in Lambek (1958) and subsequential related work, notably Moortgat (1988) and Van Benthem (1991). Combinatorially, they can be seen as onions. Their internal structure might be complicated, but only their outer skin determines the combinatorial potential in a given state. The related combinatorics is naturally context-free, since the internal structure of the consequent of the primary category in the directed *modus ponens* that drives the system is left out of consideration. The full proof of the context-freeness of the Lambek-calculus based on this concept of type was presented by Pentus (1993). The computations that come with it relate to a tuple <body, outer skin>, where outer skin is the specified argument *with* its direction (see Steedman 1990 for an invariant of this nature).
Secondly, there are the 'flat' categories as derived in Buszkowski (1982). Though the author may not necessarily see them this way, they introduce the full directed internal structure of the *premisse major* as an entity in the *modus ponens*. Combinatorics defined on these data structures are context-sensitive (though not necessarily in the strong sense) because the internal structure of the premise is specified, and possibly taken into consideration. The computations that go with it relate to triple <head, leftward structure, rightward structure>. These characteristics of the flat, skinned categorial data structures also apply to the very first proposals of Combinatory Categorial Grammar, in Ades & Steedman (1982).
These two types of categorial information structures induce different calculi. The differences follow from the data structures themselves. The flat types allow for a fine-grained resource management in terms of linearization and subspaces. The onion types induce calculi that exploit the bare conditionalization, in the form of a conditional hierarchy, adding additional operations for resource management, like linear structuring.

Although a grammar may exploit conditionalization, it has no use for full complementation. Yet, negation is the landmark of reasoning, as it offers an alternative to the referent of a proposition. Without negation, we would not have truth – or whatever other device we may use to express the contingency of a proposition. Negation constitutes the smallest device that one needs to generalize over situations or states-of-affairs and to tell them apart. In grammar, however, there is no place for algebraic complementation. By consequence, natural-language grammar cannot be identified with reasoning as such. It can only be depicted as an algebra forging some structure over a set of expressions. Linguistics, then, is about the components of that algebra.

1.5 THE CALCULI

The strategy for characterizing natural languages as an algebra is choosing some well-defined starting-point to see what additional power is needed to deal with the intrinsicalities of the language at hand. That starting point could be PLA, the categorial system based on commutative residuation (55) and explored by Van Benthem (1991); it amounts to the Lambek-calculus under abstraction of directionality. This system is too weak, though. It defines no (syntax of any) natural language at all, since it induces a permutation-closure over the set of accepted strings. No natural language is known to be that free, not even Warlpiri, Hungarian or Latin. Therefore, it seems reasonable to upgrade one level, and exploit the system defined by residuation (54), *i.e.* the Lambek-calculus. In doing so, we take directionality and non-commutativity as basic, just like we take the interpretation of division as conditionalization to be basic. On the other hand, directionality and non-commutativity can be added to PLA in terms of modalized duals. Here is an example, expressing the functionality of (54) in (55):

(58) $a \to c|_i b$ *iff* $a \cdot_{i,j} b \to c$ *iff* $b \to c|_j a$

Here we can see that directionality and non-commutativity can be added to PLA just as any other regime of structural management could. The expressivity of such a system is more than sufficient to cover natural languages, it seems. It leaves room for the embedding of complex analyses of natural languages, as shown in Vermaat (1999). In this approach, every computable structure can be accommodated. It is Turing-complete. Of course, it is highly interesting

and revealing to scrutinize which embeddings are possible for every particular phenomenon or analysis. In fact, the multimodal approach exemplified in (58) seems to come with the categorial alternative to the Chomsky hierarchy called for in Van Benthem (1991).
CLG is considerably less expressive than multimodal categorial grammar. It imposes heavy restrictions on the set of derivable sequents and, on top of that, on the class of accepted strings, for any finite lexical assignment.

Morrill (1994) considers grammar as applied logic, as residing on logic grounds. In this view, adding axioms or adjusting operators just to comfort language analysis is needlessly weakening the grammar-logic connection; it amounts to redefining mathematics in order to get hold of nature. In another perspective, however, logic is defining (some of) the instruments for grammatical construal. Grammar is not necessarily applied logic, but may involve the application of logical means for natural-language analysis. This perspective finds justification in the present state of ignorance about the embedding of language in the cognitive and intellectual resources of our species. As far as we can see, at this moment we do not have conclusive evidence for language as an independent faculty, or for language as an epiphenomenon of emerged cognitive capacities, or for anything in between. This should not stop us from pursuing some hypothesis or other. In any case, one should adhere to explicating in grammar all those processes that one observes in scrutinizing natural language. Logic is extremely useful for this explication. CLG is an effort to localize the interference of selection and linearization in natural languages. Take, for example, the wrapping that canonical auxiliaries in Dutch induce. Auxiliaries, except when advanced to second position, are head adjuncts: they select a verbal complement of some sort to their right, but will typically wrap this complement around themselves in such a way that the complement's head is its right neighbour:

(59)	auxiliary	*wordt* ('is', passive aux)	x/vppas
	vppas	*in de verf gezet* ('painted')	(verb: *gezet*)
	string	*in de verf wordt gezet*	x

Alternatively, they select a complement to their left. But in that case, everything occurring to the right of the complement's head has to occur, after merge, to the right of the passive auxiliary:

(60)	auxiliary	*wordt*	x\vppas
	vppas	*met geweld gedwongen te vertrekken* ('violently forced to leave')	(verb: *gedwongen*)
	string	*met geweld gedwongen wordt te vertrekken*	x

Neither for the rightward nor for the leftward selection are string-ordering alternatives available. Any other ordering essentially affects interpretation or endangers interpretability. The following ordering variations on leftward selection show this.

(61) * met geweld gedwongen te wordt vertrekken
met geweld gedwongen te vertrekken wordt
*can only mean something like "*becomes violently forced to leave"

But then the following statement appears to be true: if the direction of the complement selection is part of the auxiliary's lexical definition, the way of wrapping that complement is part of that definition too. That is what CLG specifies. It takes discontinuity seriously, in defining natural-language grammar as computation of discontinuity. Likewise, non-adjacency of strings that are to be interpreted in relation to each other is a fundamental fact of language. Language is linear. It is a time-consuming process, and the order of events in this process is crucial. Disharmonious composition is one of the instruments for dealing with this ordering. It is not favoured or disfavoured compared to other merge formats. Logic, then, is just exploited to keep the analyses tractable.

1.5.1 Merge and Move

The presentation of the syntax hitherto has left no room for a fundamental distinction between the *move* and *merge* operations, as introduced in Chomsky (1995). Stabler (1992) essentially reduces this distinction to a matter of arity. Move is operating on one category to produce another. Merge operates on two categories to produce another. In both operations, checking features is the engine of the process. Cormack and Smith (1999) hypothesize that interpretation and merge of a constituent take place at the same level of derivation. There is no need for move in this case. CLG incorporates the view that merging implies moving, as lexical material is anchored in positions where it may or must split what otherwise would have been decent chains. The famous case at stake here is dative insertion. It is generally acknowledged that a verb and its direct object are tightly related. In many languages, how-

ever, a second object – the indirect one – may or must interfere between the verb and the direct object. This object too is licensed by the verb, but its relation to it seems looser, both syntactically and semantically. There is no particular relation between the direct and the indirect object. Nor is it the case that the complex of verb and neighbouring indirect object represents a more complete syntactic or semantic frame for the direct object. We do not know of any cases where verb and indirect object bring in interpretations that are not available to the verb without an indirect object. Since the indirect object is still part of the verbal complex, the mere merge of verb and indirect object induces relative movement of the direct object some distance away from its very licenser, the verb.

Likewise, one might look at the relation between prenominal adjectives and determiners. It is quite clear that there are several very severe restrictions operating on a determiner and its noun, ranging from the morphological shape to the quantificational nature of the resulting constituent. Adjectives have no, or hardly any, relation to the determiner, and some, but not very lexical (and thus not very computable) semantic cross-links with the noun. Still, adjectives occur at that side of the noun where determiners are bound to reside. The complex formation of adjective and noun, then, puts the determiner at some distance from its closest comrade.

In many respects, therefore, merge inevitably induces discontinuity, and discontinuity amounts to movement (cf. Cremers 2004).

1.5.2 Soundness and completeness

A sequent of the form $\Gamma \rightarrow B$ is valid if the set of phrases associated with Γ is included in the set of phrases assigned to B, *i.e.* if $\mathfrak{R}(\Gamma) \subseteq \mathfrak{R}(\mathrm{B})$. Here is proof that the sequent calculus presented above is sound, and in a certain sense complete with respect to the string model.

1.5.2.1 Soundness

As for soundness, *i.e.* the proposition: all that can be derived is valid, consider the following. The unary axiom is trivially valid. The binary axioms are valid by definition. We can therefore concentrate on the left one of the only two productive derivational steps:

(62) $$\frac{\Delta \rightarrow \mathrm{B} \qquad \mathrm{A} \otimes_{\mathrm{i}} \mathrm{B} \rightarrow \mathrm{C} \qquad \Gamma', \mathrm{C}, \Gamma'' \rightarrow \mathrm{T}}{\Gamma, \mathrm{A}, \Delta, \Gamma'' \rightarrow \mathrm{T}}$$

Suppose $d \in \mathfrak{R}(\Delta)$ and $a \in \mathfrak{R}(A)$. By induction, $d \in \mathfrak{R}(B)$. By definition, $a+d \in \mathfrak{R}(C)$ and $\mathfrak{R}(A \bullet \Delta) \subseteq \mathfrak{R}(C)$. So for $g' \in \mathfrak{R}(\Gamma')$ and $g'' \in \mathfrak{R}(\Gamma'')$, $g'+a+d+g'' \in \mathfrak{R}(T)$ and thus $\mathfrak{R}(\Gamma \bullet A \bullet \Delta \bullet \Gamma'') \subseteq \mathfrak{R}(\Gamma' \bullet C \bullet \Gamma'') \subseteq \mathfrak{R}(T)$.

1.5.2.2 Completeness

As for completeness, *i.e.* the proposition: all that is valid can be derived, we have to prove that $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$ implies derivability of the sequent $\Delta \rightarrow T$. In its full strength, this probably cannot be proven for the present system, as it lacks hypothetical reasoning and, thus, structural completeness – the property that the order of deduction steps is irrelevant for well-formedness and derivability. What can be proven, however, is a weaker statement that amounts to the following proposition.

(63) There is always an assignment of phrases to categories, such that if $d \in \mathfrak{R}(\Delta)$, $d \in \mathfrak{R}(T)$, then there is a Γ such that $d \in \mathfrak{R}(\Gamma)$, $|\Delta| = |\Gamma|$ and $\Gamma \rightarrow T$.

This means that the calculus is complete *salve* assignments of phrases to categories. The proof yields a construction of Γ as a string of categories $\Delta[X_i \leftarrow Y_i]$ which is like Δ, except for a finite number of substitutions of a category X_i by Y_i. The number of substitutions has an upper limit $|\Delta|$ and consequently, $|\Delta[X_i \leftarrow Y_i]| = |\Delta|$.

1.5.2.3 Help lemmas for completeness salve *assignment*

To prove completeness *salve* assignment, we have resort to the following lemma; here *L-LL* denotes the remnant of L after taking out that which L and LL have in common.

(64) *Binary derivability lemma*

for all Γ, $|\Gamma| \geq 2$, $\Gamma \rightarrow a\backslash L/R$ *iff*

$\Gamma = \Delta' \Delta''$ *and either*

$\Delta' \rightarrow a\backslash L_1/[b\hat{}i|R_1]$, $\Delta'' \rightarrow b\backslash L\text{-}L_1/R\text{-}R_1$ and
$a\backslash L_1/[b\hat{}i|R_1] \otimes_i b\backslash L\text{-}L_1/R\text{-}R_1 \rightarrow a\backslash L/R$

or

$\Delta' \rightarrow b\backslash L\text{-}L_1/R\text{-}R_1$, $\Delta'' \rightarrow a\backslash [b\hat{}i|L_1]/R_1$ and
$b\backslash L\text{-}L_1/R\text{-}R_1 \otimes_i a\backslash [b\hat{}i|L_1]/R_1 \rightarrow a\backslash L/R$

This lemma is to play the same role in the present completeness proof as the *canonical lemma* of Buszkowski (1982) plays in the completeness proof of the product-free Lambek-calculus.

From right to left, the lemma is evident, as it reflects the deductive aspect of the calculus. From left to right, the lemma is not trivial. It induces binary branching of the deductive process. The crucial deduction step in (62) substitutes a category *C* with a string *[A,Δ]*. If $\Delta \rightarrow B$ is derivable, it is an axiom, or it is derived by the same rule. In that case, Δ can be represented as Δ'', B' or B', Δ' for some category B' with the same head as *B*. This can be repeated until we end up with single categories only. Thus the structure 'below' *C* can be seen as a binary branching construal with a category on one branch and a string of categories on the other. 'Above' *C*, we may safely assume that C itself will be a right-hand or left-hand partner of some peripheral substring of either Γ' or Γ''. This duet, then, is substituted, in the deduction, by come category C'. Consequently, *C* itself will be the product of a binary process. Since every reduction decreases the number of categories and the accumulated complexity of the sequence, the structure converges, and must be binary branching. But then, at the top of that structure there are two categories covering the whole string of categories that is being deduced. The format of the categories involved in the lemma follows from the definition of merge. Note that by the monotony property of the grammar the arity of the consequent puts an upper limit on the arity of each of the categories in the antecedent.

We furthermore need the notion of *compatibility* between categories. If $A \otimes_i B \rightarrow C$, both A and B are compatible with C. If *A* is compatible with *C*, then, by binary derivability, for some *B*, $A \otimes_i B \rightarrow C$. One can easily compute that compatibility of *A* to *C* amounts to *C*'s argument lists being embeddable in *A*'s. It is noteworthy that the number of categories X compatible with a given Y is finite: Y's argument lists are finite and the set of types is finite. In fact, the number is linear in the arity of Y. Compatibility in this sense is a derivative of count invariance, established by Van Benthem (1991) for the Lambek-calculus with permutation. Count invariance expresses the property of a rule of categorial grammar that the types at the left-hand side add up to the types at the right-hand side when types can occur either positively or negatively in categories.

Finally, it must be shown that for every category that could be the product of reduction by some merge, such a merge can be found. A category is *reduced* if exactly one of its argument lists is flagged $_a\sim$. Recall that this flag indicates that one of the argument lists from which it was appended contained an argument that was cancelled by merge. If none of the list flags of a category is $_a\sim$, the category might still be a reduction; this would, however, depend on the availability of particular instances of merge. Furthermore, we assume that there is no interesting difference in denotation between two categories which

differ only in the flagging of their arguments. Flagging is control, not semantics. This was expressed in the last statement in definition (48).
Under this *proviso*, the following has to be shown:

(65) *Reducibility*
for every reduced *T,* there are categories *A* and *B* and a merge mode *i* such that $A \otimes_i B \rightarrow T$

Since categories are constructable *ad libitum*, the presence of suitable merge modes is the bottleneck here. We may safely assume that every grammar defines at least one merge mode. Let $\otimes_i$ be this merge mode and suppose, without loss of generality, that it induces a right-hand side cancellation; let *T*'s reducibility agree with this directional feature. *T* will, then, have the format $a\backslash fl_{lt}{\sim}L/a{\sim}R$. Thus it must be proven that there are argument lists L_1, L_2, R_1 and R_2 such that:

(66) $a\backslash fl_{la}{\sim}L1/fl_{ra}{\sim}[b\hat{}i|R_1] \otimes_i b\backslash fl_{lb}{\sim}L2/fl_{rb}{\sim}R2 \rightarrow a\backslash fl_{lt}{\sim}L/a{\sim}R$

The merge mode *i* specifies some appends $L_1 \oplus_{il} L_2 = L$ and $R_1 \oplus_{ir} R_2 = R$ with appropriate flagging as output-conditions, and some restrictions $inp(L_j)$ and $inp(Rj)$ as input-conditions on the left and right argument lists of the two antecedent categories, including their flags. Now suppose that $fl_{la}{\sim}L1$ agrees with $inp(L_j)$ and that $fl_{ra}{\sim}[b\hat{}i/R_1]$ agrees with $inp(R_j)$; of course, we can always construct the first category so that it accords with the input requirements for *i*. So, the only thing left to show is that, given *T*, the operation $\otimes_i$ and an appropriate compatible left-hand side can be constructed. Recall that the appendings that come with the merges are defined conservatively. They are restricted to asymmetric linear gluing of lists. Consequently, L_1 is constructed in such a way that it is a sublist of *L*; as a matter of fact, it is a (possibly empty) prefix or a (possibly empty) suffix of *L*. Which of these options holds is defined by $\oplus_{il}$ as part of the definition of $\otimes_i$. But then, L_2 is the result of the subtraction *L-L1*, uniquely defined given *L* and L_1. The same reasoning can be carried over to R_1, R_2 and *R*. So, L_2 and R_2 are uniquely defined by $\otimes_i$ and the other argument lists. As for the flags of the argument lists of the antecedent categories, since they do not in any respect reflect the present state of these lists, they can be chosen freely in accordance with $\otimes_i$. So, *B* can be constructed. This proves reducibility (65).

1.5.2.4 Completeness salve *assignments*

Given binary derivability, completeness *salve* assignments amounts to the claim that $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$ implies derivability of the sequent $\Delta', \Delta'' \rightarrow T$ for some bipartition Δ', Δ'' of Δ and reduced T. Recall that $\mathfrak{R}(T) = \mathfrak{R}(T')$ if $T = T'$ *modulo* flagging of argument lists. Now suppose $d \in \mathfrak{R}(\Delta)$, $d = d' + d''$, $d' \in \mathfrak{R}(\Delta')$, and $d'' \in \mathfrak{R}(\Delta'')$. Then, if $\Delta' \rightarrow A$ and $\Delta'' \rightarrow B$ for some A en B such that $A \otimes_i B \rightarrow T$, $\Delta', \Delta'' \rightarrow T$ and, by soundness, $d'+d'' \in \mathfrak{R}(T)$. So it must be proven that such A and B exist. This proof is by induction on the length of Δ' and Δ''.

(67)

(a) If $|\Delta'| = 1$, we have identity and for some A, $\Delta' \rightarrow A$.

(b) Suppose A is compatible with T. By reducibility, there is a B such that $A \otimes_i B \rightarrow T$. As for Δ'', the proof now requires $\Delta''[X_i \leftarrow Y_i] \rightarrow B$ to be derivable.

(c) If $|\Delta''| = 1$ and $\Delta'' = C$ and $C \rightarrow B$, we are done. Either $\Delta \rightarrow T$, as in case (a), or $\Delta[A \leftarrow A'] \rightarrow T$.

(d) But suppose that not $C \rightarrow B$. We assign every string in $\mathfrak{R}(C)$ to $\mathfrak{R}(B)$. Then, $\mathfrak{R}(C) \subseteq \mathfrak{R}(B)$ and in particular, $d'' \in \mathfrak{R}(B)$. It is evident that substituting C for B in Δ'' amounts to identity, and assures derivability of $\Delta[C \leftarrow B] \rightarrow T$, *i.e.* reducibility of the string Δ with the rightmost occurrence of C substituted by B to T. Moreover, $d \in \mathfrak{R}(\Delta[C \leftarrow B])$ and $d \in \mathfrak{R}(T)$.

(e) Now suppose $|\Delta''| = 2$. Again, by binary derivability, $\Delta'' \rightarrow B$ if $C'\ C'' \rightarrow B$ for some C', C''. We take the same cycle (a)-(c) as above and show that either $\Delta'' \rightarrow B$ or $\Delta''[C' \leftarrow C'''] \rightarrow B$, or $\Delta''[C'' \leftarrow C''''] \rightarrow B$, or $\Delta''[C' \leftarrow C''', C'' \leftarrow C''''] \rightarrow B$. Moreover, $d'' \in \mathfrak{R}(\Delta''[\ldots])$ and $d'' \in \mathfrak{R}(B)$. Consequently, $d'+d'' \in \mathfrak{R}(A \otimes_i B) \subseteq \mathfrak{R}(T)$.

(f) If $|\Delta''| > 2$, the reasoning (a)-(d) applies to prove the derivability of some sequent $\Delta''' \rightarrow B$, where Δ''' is Δ'' except for a number of substitutions $X \leftarrow Y$ linearly bounded by $|\Delta''|$, such that $d'' \in \mathfrak{R}(\Delta''')$, $\Delta', \Delta''' \rightarrow T$ and $d \in \mathfrak{R}(\Delta'\ \Delta''')$.

(g) Now suppose that A is not compatible with T. Then substitute A by some A' that is compatible with T and ensure that $\mathfrak{R}(A) \subseteq \mathfrak{R}(A')$. Then go for B, as above. Inevitably, $\Delta[A \leftarrow A'] \rightarrow T$ or $\Delta[A \leftarrow A'|Other] \rightarrow T$, where Other is any combination of substitutions induced by inspection of Δ''.

(h) Suppose $|\Delta'| > 1$. Then, following the track (d)-(f) above for Δ'', create a derivable sequent $\Delta'''' \rightarrow A$ such that A is compatible with T, $d' \in \mathfrak{R}(\Delta'''')$ and Δ'''' is like Δ' except for a number of substitutions $X \leftarrow Y$ that is linearly upwardly bound by $|\Delta'|$. For that A, there must be a B such that $\Delta''' \rightarrow B$ where Δ''' comes from Δ'' as described above. Again, $d'+ d'' \in \mathfrak{R}(\Delta'''', \Delta''')$, $\Delta'''', \Delta''' \rightarrow T$ and Δ is like (Δ'''', Δ''') except for a finite number of substitutions $X \leftarrow Y$ linearly bounded by $|\Delta|$. Moreover, $|\Delta| = |\Delta'''', \Delta'''|$.

This ends the proof of completeness *salve* assignments for Categorial List Grammar. So, for a string Δ of power n, for which $\mathfrak{R}(\Delta) \subseteq \mathfrak{R}(T)$, one can find, in at most n substitution steps, each of which is decidable, a string $\Gamma = \Delta[X_i \leftarrow Y_i]$, $0 \leq i \leq n$, such that $\Gamma \rightarrow T$.

1.5.3 The fundamental asymmetry of merge

In section 1.3, it was put forward that append treats argument lists, originating from two antecedent categories in a merge, necessarily asymmetrically. At each side, all the arguments of one category will be on top of the arguments of the other category. This relative order correlates with the relative distance of argument strings to the string of the category: the types on top of the argument list induce strings that will be closer to the string induced by the category's head than the strings induced by lower types. Since append is the canonical operation on argument lists, this asymmetry is included in CLG.

Furthermore, the linearization of the strings in the two antecedent categories of each merge brings in another intrinsic form of asymmetry. It implies that at least at one side the strings induced by the arguments of an antecedent category will be separated from the string induced by the category's head. Here is a scheme of this pattern for a certain instance of a defined merge mode.

(68) $(\otimes_i/)$
PRIM\PLF~PLA/PRF~[SEC^i|PRA] $\otimes_i$ SEC\SLF~SLA/SRF~SRA $\rightarrow$
PRIM\SLF~(SLA⊕PLA)/a~(PRA⊕SRA)

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list

(69) s\u~[np^j]/ a~[vp^i, pp^k] $\otimes_i$ vp\ u~[ap^l] / u~[vp^m] $\rightarrow$
s\u~[ap^l, np^j]/a~[pp^k, vp^m]

(70) prim $\in \mathfrak{R}$(s\u~[np^j]/ a~[vp^i, pp^k]), sec $\in \mathfrak{R}$(vp\u~[ap^l]/u~[vp^m]),
vp $\in \mathfrak{R}$(vp^m), np $\in \mathfrak{R}$(pp^k), np $\in \mathfrak{R}$(np^j), ap $\in \mathfrak{R}$(ap^l)

(71) np + ap + prim + sec + pp + vp

Here, (71) represents a string that under assignment (70) may be the result of a derivation involving instantiation (69) of merge mode (68). The string *prim* is of the category that licensed *pp* but is separated from it by *sec*. Similarly, *sec* is of the category that brought in *ap* but is not adjacent to it in (71); yet, the type of *ap* was the top of the left argument list of the secondary antecedent category in merge (69). In this case, *sec* is not adjacent to any string induced by arguments of the category it belongs to. By definition of merge, however, the string associated with the primary category will be next to at least one string induced by the primary category's argument: the string induced by the cancelled argument and the secondary category. This asymmetric aspect of merge gives rise to an essential difference in status between strings of the primary category and strings of the secondary category. We hesitate to identify

this difference with *headness*, as made relevant to categorial grammar by Pollard (1984), Hoeksema and Janda (1988) and Jacobson (1991), among others. A head – if not empty – is the predominant string in a certain substring. Its category defines the structure immediately around it. From the asymmetry observed here, however, we can only derive the following property of *string connectedness* for strings in primary categories of a merge.

(72) *String connectedness*
A string s of category C – $s \in \Re(C)$ – is connected if it can be partitioned as *w+h+v* and *w* and *v* are induced by arguments in C, or w is induced by an argument in C and v is induced by an argument of the category w belongs to, or the other way around.

Merge imposes string connectedness on the string in its primary category. The string in the secondary category lacks this property, as one of its neighbours, namely the string associated with the primary category, is not induced, directly or indirectly, by one of the secondary category's arguments. It is precisely in this sense that the primary category defines a string beyond the string by which it is introduced.
String connectedness has a dual. If a string *w+p+s+v* is 'felt' to be defined by p then there is a category C such that $p \in \Re(C)$ and *C connects p in w+p+s+v*. It is not necessarily the case, however, that w+p+s+v is a constituent. Also, there does not have to be a sequence of categories D such that $w+p+s+v \in \Re(\Delta)$, that C is in Δ, that $\Delta \rightarrow C'$ for some C' (and that C introduces the head type of C').

Summarizing, merge imposes asymmetry in the relation between the antecedent categories in each of the following senses:

(a) precisely one category has one of its arguments cancelled, *i.e.* one category is primary, the other secondary
(b) the linear ordering of the categories in the merge induces the linear ordering of the strings
(c) append on argument lists is asymmetric, *i.e.* at each side, the arguments of one category are on top of those of the other category
(d) the strings associated with the primary category are connected in the string defined by the primary category.

Asymmetry, *i.e.* irreversibility of the (dependency) relationship between elements in natural language is the resultant of linearization. In CLG, linearization is an intrinsic component of the syntactic engine. It has been argued that in the grammar of natural language precedence and dominance relations could, or should, be separated. This has been a major topic in the develop-

ment of GPSG and HPSG. It was suggested for categorial grammar by Flynn (1983). Flynn stressed that constituency relations might be at another level of universality from linear restrictions. The topic recurs in the categorial mainstream in the format of specialized permutation modalities, meant to rearrange word order in a controlled way but independent of the combinatory, reductionist engine; see Moortgat (1998) for a detailed overview, and Hepple (1990) for the original approach. The modular view on precedence and dominance, word order and constituency, permutation and composition or any other duality that captures the two-dimensionality of natural language has been challenged by Kayne (1994). Kayne holds language to be essentially antisymmetric – and thus asymmetric – because linearization *entails* dependency. In his view, a certain linearization is not an accident of a particular language, but the prerequisite to interpretability and its reflection.

The present system is hardly reminiscent of Kayne's concept of grammar. But CLG respects the interdependency of word order and constituency by exploiting only one syntactic operation that accounts both for word order and hierarchical, interpretative dependencies. The asymmetry which CLG imposes on the linear dimension and the ordering of strings comes with an irreversible inequality of the primary and secondary categories in a merge. In this respect, it must be noted that typed logics, and typed lambda calculus in particular, incorporate antisymmetry by nature: the types of function and argument differ by definition, and whatever process can be reduced to functional application would embed antisymmetry this way.

The asymmetry of CLG is, of course, not an argument in favour of linearity as the anchoring dimension of natural-language grammar. It represents, though, a choice in this respect.

1.5.4 No manipulation of structure

An immediate property of the syntactic model advocated here is that it excludes the manipulation of structure. Once a configuration has been constructed, it is fixed by the simple fact of its being constructed. Of course the syntax must be able to handle discontinuities, but discontinuities arise from merging categories, just as continuous structures do.

The anchor of this property is the monotony of the derivation. In each and every derivational step exactly one cancellation occurs. As a consequence, the structure of the categories controls the derivation with an iron hand. No empty moves can occur. In this respect too, our syntax is rigid. The phrase structure

and the derivational structure – not necessarily the derivational process – coincide. To put it in other terms, there are no type-shifts in our syntax.
Since transformations and other manipulations of structure are not necessarily without purpose, the syntax' rigidity calls for a trade-off. In DELILAH, the burden is on the lexicon. Every variation in structure has to be introduced lexically. It leads to a massive number of distinct lexical specifications for each individual member of central syntactical classes like verbs. It makes no sense to try and express this in figures, as DELILAH's grammar is in no way complete. The specification load on the lexicon calls for a powerful lexical engine, producing all these forms and expressing two types of generalizations:

(a) all the specifications of a certain verb are *of* that verb – they live on one matrix lemma;
(b) the entire variety of each matrix lemma is induced by general properties of sentence structure, *i.e.* reflects the structure of the language.

In chapter 3, the structure of the lexicon will be addressed, and there we offer arguments with respect to the viability, constructability and operation mode of a lexicon with dimensions as sketched above. Here, it suffices to repeat that the operational rigidity of the syntax re-enters in the overall system as the tremendous complexity of the lexicon. This, however, is a choice – a radically lexicalist choice.

1.6 THE CASE FOR DUTCH

1.6.1 General Format

In the preceding section, the general properties of the standard two-place operation of merge were introduced. It was noted that every merge modality specifies a format for the two categories that go into the merge, and a format for the category that emerges from them. Several aspects of the operation are general. There is always exactly one type to be cancelled, in a designated position at the edge of one argument list. The stacks are addressed as lists. Cancellation requires typological identity of that argument and the head of the other stack. The head of the output category is invariably the head of the primary input category. The (remnants of) the argument lists are appended, pairwise

and directionwise. Argument lists are marked for affectedness, *i.e.* for having or not having undergone cancellation of one or more member types. Arguments are marked with one merging mode, indicating a specific mask on the input and output of the merge operation that is to cancel that argument.

Furthermore, the nature of the specifications that can be stated in a merge modality is fixed. They come from a limited set. Argument lists of the input may be required to be empty or non-empty; non-emptiness means that the stack contains at least, or exactly, one argument, specified or not – this difference will be ignored in the calculations. Moreover, lists are marked affected or not-affected, initially unaffected by assignment and subsequently affected or unaffected by computation or specification. Specifications on argument lists are not necessary, though. The output category is specified in terms of the input components. The way of appending the input lists is determined. So is the affectedness marking on the output lists. Here is an overview of the possible specifications on input and output for a rightward cancelling modality *i*.

(73) *phrase1:* PRIM \ PLF~PLA / PRF~[SEC^i| PRA]
$\otimes_i$
phrase2: SEC\ SLF~SLA/SRF~SRA
$\rightarrow$
phrase1+phrase2: PRIM\LF~LA/RF~RA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

specifications of PRF, SRF: *affected*, *unaffected* or none
specifications of PLF, SLF: *wh, affected* or *unaffected*
specifications of PLA, PRA, SLA, SRA: *empty*, *nonempty* or none
specifications of LF: *wh, affected, unaffected*, depending on the values of PLF and SLF and the values of PLA and SLA
specifications of RF: *affected* (normally; exceptionally *unaffected*)
specifications of LA: *PLA + SLA* or *SLA + PLA*
specifications of RA: *PRA + SRA* or *SRA + PLA*

Now we can compute the total number of different modalities that can be specified within these limits. There are twelve items that may be specified: six argument lists and six flags. For each of the input arguments lists three options are available: the list is empty, the list is non-empty with a specified type on top, or the list is unspecified. The options for the two input lists are formally independent. Each of the input flags also has three options available: the list may have been *affected* by prior cancellations, it may be affected (in

fact, emptied) by a rule involving cancellation under a *wh*-mode, and it may be *unaffected* or in its lexical state. The choices are, again, formally independent of each other, but the *wh*-flag is only realized for flags of left lists. For each of the four components of the output category only two options are available, since they must be fully determined by the rule. These options are also independent, with the possible exception of the value of Rf.

Any mix of these options will define a modality. That is: a modality is a 12-tuple <Plf, Pla, Prf, Pra, Slf, Sla, Srf, Sra, Lf, La, Rf, Ra>. As an example, a modality could be <u, a, [np | [ ]], a, [ ], u, _, a, l+r, a, r+l >. Consequently, within this format at most $3^8.2^4$ = 104976 different modalities can co-exist. Materially, however, this limit is too high. Every requirement as to the emptiness of an input argument list fixes the value (for append) of the output list in that direction, since [ ] ⊕ L = L ⊕ [ ] = L for any L. Thus, the values for La and Ra are not independent of the values for Pla and Sla and for Pra and Sra, respectively. Practically, then, the limit is $(3^2\text{-}2^2).3^4.2^2(2^4 + (3^2\text{-}2^2))$ = 34020 different modalities for rightward merge.

Moreover, the flags of the output argument lists are completely determined by the values of the flags and lists of the input categories. They are therefore not so much part of a specification but computed at each merge, with the exception of Rf's value being un*affected*. This reduces the number of modalities down to half the computed number, *i.e.* 17010 and almost to one quarter (say, 9000) if we take the exception to be exceptional. Moreover, it is likely that we could establish various kinds of clusters of specifications which might exclude or induce each other. For example, it is unlikely that a certain argument list is required to be both empty and either affected or unaffected. Affectedness indicates that the phrase bearing such an argument list is not lexical but composed. Unaffectedness has the opposite flavour. How likely is it that we will discover a merge process that imposes conditions both on the internal structure of a category's component and on its 'flattened history'?

A categorial list grammar will be any set of modalities, defined on a given set of types. This view raises all kinds of questions as to the consistency of these sets, as well as with respect to the practical and theoretical equivalence between classes of modalities, and the decidability and expressibility of certain properties. Most of these questions cannot be addressed here.

In the section on patterns of discontinuity, we will reconsider the variety of merge modalities. But, clearly, the number of possible modalities ensures that no grammar constructed from them is trivial. Each selection of modalities, even if the selection is very small, as will normally be the case, represents a very specific grammar.

In practice, and in our system, the number of modalities will be relatively small. In the fragment described below, no more than twenty modalities in each direction are addressed, despite the considerable complexity of the phenomena covered. The reason is simple: the formal independence of the twelve parameters is not maintained in the practice of natural-language grammar – grammar is a subtle network of co-occurrence restrictions on parameter values.

1.6.2 A concise syntax of Dutch

1.6.2.1 Flagging argument lists

All the modalities used in the DELILAH grammar of Dutch enjoy some kind of input specification. The single mode that does not specify any input component is not used. It is unlikely that any reasonable grammar of any language will employ that mode. It amounts to the absolute insensitivity of an element for the structure of its environment. Only extra-sentential elements, like interjections of sighing, qualify for this form of context-freeness, but, even there, constituency may play a role.

The output category of the merge is always fully specified. The flags of the output argument list, appended from a pair of input lists, are computed from the flags of the input lists that are appended, and from the lists' status. The flag in the active direction of the merge will almost invariably get the value *affected*, abbreviated *a~* or *w~*. The affectedness value *w~* is assigned if the cancelled argument required application of the *wh*-modality. Most other cancellations lead to the affectedness value *a~*. This will be explained below. The other value is marked *u~*, for *unaffected*. The other output flag, the one for the argument list in the passive direction, is computed deterministically according to chart (74), the table of possibilities for two input lists and one output list in the same (passive) direction.

(74) Table of flags for output argument list: *flag ~ list*

	input 1	*input 2*	*output flag*
a.	w~_	a~[_|_]	w~
b.	u~_	w~[_|_]	w~
c.	u~_	a~[_|_]	a~
d.	_~[ ]	_~[ ]	u~
e.	u~_	_~[ ]	u~
f.	u~_	u~_	u~
g.	a~[_|_]	a~_	a~

The situation where two input lists are flagged *w~* does not occur, because of the way the *w~* flag is embedded. Just like $a_{\sim}$, the flag is assigned only combinatorially and, by definition, to an empty list in the active direction. The flag can be transferred, however, to non-empty lists according to (74) a. Derivations involving the computations (74)a and b, however, are bound to be pathological and dead-ending: the *w~* flag is to occur in categories that are completed (no arguments left to be cancelled), because *wh*-extraction induces strong islands.
The most important aspect of the computation is the restoration of unaffectedness by (74)d and e. The moral here is that whenever an empty list is appended, the nature of its emptiness – emptied or lexical – gets out of sight. This embodies locality. If a constituent has completed its agenda at one side, the next merge renders its completion invisible or irrelevant for further merges; its history and its structure are enveloped in the new structure. Note, furthermore, that the output list can be marked *affected* only if at least one of the input lists is.

In the sections to follow, all the rules of the DELILAH grammar are presented, explained and exemplified. The modalities are indicated by short memo-subscripts on the merge-operator and as extensions to arguments. The rules are named after the modalities. The flag of the output list in the passive direction is to be computed according to the scheme above, and is not specified in the rule, but invariantly dubbed *LF*.

1.6.2.2 Rightward rules

1.6.2.2.1. */^isl* FOR SATURATED CONSTITUENTS

This mode requires that the secondary category is completed. It does not allow any part of a secondary agenda to be transferred to the output.

(75) PRIM \ _~PLA / _~[SEC^*isl*|PRA] $\otimes_{\text{isl}}$ SEC \ _~[ ] / _~[ ] → PRIM \ LF~PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Because of the emptiness of the secondary argument lists, the appends are trivial. The standard application of this merge is the consumption of a noun phrase. Noun phrases are closed domains in Dutch. They do not allow for any discontinuity in their components. All noun phrases in argument lists select the */^isl* cancelling mode.

Here is an example of */^isl*:

(76) `q\u~[]/u~[np^isl, vp^x]` $\otimes_{isl}$ `np\u~[]/a~[]` → `q\u~[]/a~[vp^x]`
wil 'wants' ⊗ *de man* 'the man' → *wil de man*

Other candidates for cancelling under */^isl* are all those constituents that have been identified as absolute islands, like (embedded) questions. One of the standard categories for a verb selecting an embedded question must be `vp\u~[]/u~[q^isl]`.

1.6.2.2.2. */^transp* FOR UNSATURATED CONSTITUENTS

This merge mode is in a certain sense the opposite of */^isl* in that it requires the secondary category to be transparent instead of closed. It specifies that the argument lists in the passive direction – leftward – must both be unaffected. Being lexically assigned as such is one of the ways to meet this condition. The left agendas of both input categories are transferred to the resulting category. On the other hand, the active list of the secondary category is bound to be empty (or emptied, for that matter). Since this is a rightward rule, this requirement provokes a rightward embedding structure (...X_1(...X_i(...X_n)..)..), where the right arguments of the secondary category must be met before the constituent itself is subject to merge. As a consequence, this merge mode induces a kind of head adjunction, in that the phrases introducing the primary and the secondary head types end up being neighbours in the string.

(77) PRIM \ u~PLA / _~[SEC^*transp*|PRA] $\otimes_{transp}$ SEC \ u~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The secondary passive argument list ends up on top of the passive arguments brought in by the primary category. The secondary arguments will therefore be first to meet at the left-hand side.

The typical case here is the cancellation of the infinitival verbal complement of a verb-raising (semi-)auxiliary. This was also the example given in (41). The particular appending of the passive argument lists under this merge accounts for the famous crossing dependencies in the Dutch verb cluster. In the following example, the *np*s are indexed to show this effect.

(78) `s\u~[np1^isl]/u~[vp^transp]` $\otimes_{\text{transp}}$
`vp\u~[np2^isl, np3^isl, np4^isl]/a~[]` →
`s\u~[np2^isl,np3^isl,np4^isl,np1^isl]/a~[]`
kan 'can' ⊗ *laten geven* 'let give' → *kan laten geven*

This mode is a good example of the combination of linearization and derivation, and of grammar and control. The emptiness condition on the secondary passive argument list forces this category to complete its right-hand agenda before getting involved in this merge. Unaffectedness-marking on the two left lists and the append specification make the string instantiations of primary left arguments peripheral to the substring formed here and make them precede their string counterparts of the secondary arguments. Any string introduced to meet the demands of the primary category's left agenda is bound to occur to the left of the 'secondary' strings. Recall that no further operation can change that state of affairs. The only operations that the new argument list can be submitted to are cancelling of its top element and/or appending. Neither of these operations changes the internal order established by /^*transp*.

There is an alternative for (77). It allows for an alternative derivation if the secondary category's left agenda is lexically empty, *i.e.* is empty and unaffected at merge. This would be its format:

(79) PRIM \ _~PLA / _~[SEC^*transpb*|PRA] $\otimes_{\text{transpb}}$ SEC \ u~[ ]/ _~SRA →
PRIM \ LF~PLA / a~SRA+PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The primary left agenda no longer has to be unaffected. The primary category may have consumed one or more of its left arguments before entering into this merge. This would not harm the 'crossing dependency' effects here, since the merge presupposes that there are no left arguments to the secondary category. Moreover, it defines an append at the right-hand side, guaranteeing that the secondary category's arguments are met first. Here, this does not amount to crossing dependencies, but to completion of the secondary agenda before that of the primary right agenda. That is, in effect, the same result as was reached more rigidly by the emptiness requirement on SRA in (77). Thus, (79) defines a derivationally liberalized sub-case of, for example, verb clustering. There is no need, however, to compute the verbal complex of Dutch in a syntactically and semantically adequate way.

1.6.2.2.3. */^open* FOR OPTIONAL DISCONTINUITY

An intriguing variety of */^transp* eliminates the unaffectedness constraint on the secondary left agenda. Consequently, it allows the secondary category to consume the left agenda or part of it without destroying its fitness for this merge. A string associated with the primary category can therefore connect to a string of which the left edge was not lexically associated with the secondary category.

(80) PRIM \ u~PLA / _~[SEC^*open*|PRA] $\otimes_{\text{open}}$ SEC \ _~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA/ a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Note that the (remaining) left arguments of the secondary category at merge have to be put on top of the primary ones, which are supposed to be unaffected. This merge does not mix up the left arguments differently, but puts fewer restrictions on the pre-merge behaviour of the secondary category. The only difference between */^open* and */^transp* is the affectedness of the left argument list of the secondary category.

The canonical example in Dutch for this merge is the complementation of verbs that induce the 'third construction' (Den Besten *et al.* 1988, among others). Prominent among them is *proberen* 'to try', which occurs in each of the following, semantically equivalent patterns. All other scramblings are ungrammatical.

(81) … dat jij probeerde Ramses 'Egyptian Nights' te laten lezen
… *that you tried Ramses 'Egyptian Nights' to have read*
… 'that you tried to have Ramses read Egyptian Nights'
(82) … dat jij Ramses probeerde Egyptian Nights te laten lezen
(83) … dat jij Ramses Egyptian Nights probeerde te laten lezen

Here is the instance of */^open* that would give rise to (82):

(84) `s\u~[np1^isl]/u~[vp^open]` $\otimes_{\text{open}}$ `vp\a~[np2^isl]/u~[]` →
`s\a~[np2^isl, np1^isl]/a~[]`
probeerde 'tried' ⊗ Egyptian Nights te laten lezen '*Egyptian Nights* to have read'
→ probeerde *Egyptian Nights* te laten lezen

The value for affectedness of the secondary right argument list results from two empty lists being appended at the previous merge, according to computation (74)d.
This merge mode is also basic to the arguments of adjunctive automorphisms. Verbal and sentential qualifiers may occur in a lot of positions inside verbal complexes. Consider the variety of positions for the instrumental adjunct *met een gedicht* in the following sentences.

(85) … dat Henk met een gedicht Jan de kast wilde laten openen
… *that Henk with a poem Jan the cupboard wanted have open*
… 'that Henk wanted to have John open the cupboard with a poem'
(86) … dat Henk Jan met een gedicht de kast wilde laten openen
(87) … dat Henk Jan de kast met een gedicht wilde laten openen

This flexibility reflects the combinatorial potency of *proberen*, and makes the merge mode */^open* to an essential ingredient of a Categorial List Grammar of Dutch.

1.6.2.2.4. */^penins* FOR NEAR ISLANDS

The next mode accounts for the merge with near islands, *i.e.* constituents that have on their passive (leftward) agenda one specified argument at most. In practice, this merge is used to account for combinations with constituents that lack fronted or leftward dislocated elements. A constituent licenses a dislocated element *iff* its left argument list contains a *type^mode* pair `xp^w`. The rule comes in two mutually exclusive formats, differing only in the specification of secondary passive (leftward) list. This double format leads to a disjunction of specifications.

(88) PRIM \ _~PLA / _~[SEC^*penins*|PRA] $\otimes_{\text{penins}}$ SEC\ _~[ ]/_~[ ] →
PRIM \ LF~PLA / a~PRA

(89) PRIM \ _~PLA / _~[SEC^*penins*|PRA] $\otimes_{\text{penins}}$ SEC\ _~[_^*w*]/_~[ ] →
PRIM \ LF~PLA+[_^*w*]/ a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

It is grammatically very important that in (89) the append in the passive direction suppresses the cancellation of the `^w` argument in favour of cancellation of the primary category's left arguments. In this way, the intended dislocated element can only be absorbed as the final step of completing the left agenda. That is also what the merge modality `\^w` will specify as a requirement. The

example below illustrates a case of long-distance dislocation, where an argument of a deeply embedded infinitival complement is 'wh-ed', as in (91).

(90) `s\u~[np^isl]/u~[vp^penins]` $\otimes_{\text{penins}}$ `vp\a~[np^w]/u~[]` →
`s\a~[np^isl,np^w]/a~[]`
dwong 'forced' ⊗ *een boek te geven* 'a book to give' → *dwong een boek te geven*

(91) Wie zei Henk dat Agnes hem dwong een boek te geven?
Who said Henk that Agnes him forced a book to give
'Who did Henk say Agnes forced him to give a book?'

In order to ensure that in this case the `^w` argument is properly transferred under composition, all appendings of the left argument list – independently of the particular merge mode they appear in – should be defined in such a way that `^w` arguments are stacked down. To see why, consider the case of */^transp* in section 1.6.2.2.2. If one of the complement's arguments is of the dislocated breed, this argument has to be suppressed in the output in favour of the arguments of the primary category. Otherwise, it would be at the top of the stack at the wrong moment in the derivation, namely when it is not the only item in the left agenda. That is, the output of the crucial */^transp* merge in (92) should be as in (93) rather than as in (94).

(92) Wie denk jij dat ik Agnes een boek kan laten geven?
Who think you that I Agnes a book can have give
'To whom do you think that I can have Agnes give a book?'

(93) `s\u~[np^isl]/u~[vp^transp]` $\otimes_{\text{transp}}$ `vp\u~[np^isl,np^isl,np^w]/a~[]` →
`s\u~[np^isl,np^isl,np^isl,np^w]/u~[]`

(94) `s\u~[np^isl]/u~[vp^transp]` $\otimes_{\text{transp}}$ `vp\u~[np^isl,np^isl,np^w]/a~[]` →
`s\u~[np^isl,np^isl,np^w,np^isl]/u~[]`

We can make sure that every `^w` starts at the bottom of its stack in the lexicon. For this purpose, we only have to change the standard append, applied to argument lists, in such a way that `^w` arguments at the bottom of either stack remain there. This is relevant to the tail of the list that is to become the upper part of the stack. An adequate reformulation of *append* for argument lists in a Prolog-like fashion will be this:

(95)
```
clg_append( L, R, LR') ←
                append( Lmin, [X^w], L),
                !,
                append( R, [X^w], R'),
                append( Lmin, R', LR').
clg_append( L, R, LR) ←
                append(L, R, LR).
```

Before gluing lists together, the left argument list is checked for the occurrence of a dislocated argument; if present, it is stacked down.
From now on, *append* at argument lists is considered to be reformulated as *clg_append*. Note that (95) does not interfere with the context-freeness of this operation (cf. section 1.8.3).

1.6.2.2.5. */^transpipp* FOR INFINITIVE *PRO PARTICIPIO*

This mode is a variation of */^transp*, deviating only in the additional specification that the secondary active list must be affected. As a consequence, no lexical category can comply with the secondary requirements of this mode. Its output conditions are the same as in (77).

(96) PRIM \ u~PLA / _~[SEC^*transpipp*|PRA] $\otimes_{\text{transpipp}}$ SEC\ u~SLA/ a~[ ] →
PRIM \ u~SLA+PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Just like */^transp,* this merge mode is mainly applied for dealing with the verb cluster. This particular variation is partly responsible for the so-called *infinitivus-pro-participio* effects. An auxiliary verb may select a participle as the head of its verbal complement, but can also require an infinitive when the complement itself contains a verbal cluster. Here are some relevant data; note the form of *willen* 'to want' in the examples.

(97) Jeroen had die baan wel gewild
Jeroen had that job wanted
'Jeroen would have wanted that job'
(98) Jeroen had die baan wel willen nemen
Jeroen had that job want accept
'Jeroen would have wanted to accept that job'
(99) * Jeroen had die baan wel willen
Jeroen had that job want
(100) * Jeroen had die baan wel gewild nemen
Jeroen had that job wanted accept

The mode */^transpipp* is assigned to the verbal arguments of auxiliaries that are sensitive to the *ipp*-effect. Unfortunately, that the secondary active argument list is affected is not the whole story here. It appears that the argument cancelled in the construction of the secondary category must be *vp*, and nothing else:

(101) Hij had gewild dat ik er bij was
'he had wanted that I was present'
(102) * Hij had willen dat ik er bij was
he had want that I was present

This particular requirement cannot be expressed in our syntactic formalism. It is handled by the specification in the template that the candidate *ipp* is a semi-auxiliary itself – a qualification that is restricted to a certain group of verbs taking infinitival complements. This group is qualified in Cremers (1983) as taking infinitival complements that can only be interpreted below propositional level. *Zeggen* 'to say' is not in that class. It may take infinitival complements, but they must be interpreted as propositions. Therefore, it should be excluded from *ipp*-effects, and it is:

(103) Merel had gezegd te blijven zingen
Merel had said to keep sing
'Merel said that she would keep on singing'
(104) * Merel had zeggen te blijven zingen
Merel had say to keep sing

This justifies the qualification of *ipp*-candidates as semi-auxiliaries.

1.6.2.2.6. */^sentop* FOR WH-ISLANDS

This mode requires the secondary passive list to be flagged *w*. This flag results from application of merge mode \^wh, to be discussed below. With this it indicates that the last left mode dealt with a cancellation under this merge. The present mode consumes that flag.

(105) PRIM \ u~PLA / _~[SEC^*sentop*|PRA] $\otimes_{\text{sentop}}$ SEC \ w~[ ]/ _~[ ] →
PRIM \ u~PLA / a~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The mode canonically applies to the first of the pair of categories to which an argument *wh*-phrase is assigned. In DELILAH's lexicon, a wh-word like *wie* 'who' is assigned to a pair of co-occurring categories, $C^{left} * C^{right}$ (see also chapter 3). The asterisk in the category disappears as soon *wie* is used combinatorially. Two adjacent categories remain. The rightmost of these is simply `np\u~[]\u~[]`, dealing with the selectional properties of the word as an argument to some other phrase in the sentence. The leftmost category that comes

with *wie* selects the sentence that has been built by the merge in which the rightmost category evaporates, and qualifies it. It may be qualified as a question, a postnominal modifier, or whatever operation *wh*-elements can perform. Here is the example of *wie* introducing a question; the operation $\otimes_{wh}$ is introduced below as part of a leftward rule.

(106) wie :: `q\u~[]/[s^sentop] * np\u~[]\u~[]`
(a) `np\u~[]\u~[]` $\otimes_{wh}$ `s\u~[np^w]/u~[]` → `s\w~[]/u~[]`
(b) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `s\w~[]/u~[]` → `q\u~[]/a~[]`
(b)*(a) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `np\u~[]\u~[]` $\otimes_{wh}$ `s\u~[np^w]/u~[]` → `q\u~[]/a~[]`
wie 'who' ⊗ *slaapt* 'sleeps' → *wie slaapt*

When the *wh*-term is not an argument, it has a single category. This category cannot expect a particularly constructed right partner. For example, *hoe* 'how' just requires its sentential argument to have an unaffected left argument list, thus enforcing that the last and decisive merge of this argument has not been leftward – compare the flag computation (74). The relevant mode is given below; it is as specific as (105) and therefore an exclusive alternative to it.

(107) Prim \u~Pla /_~[Sec^*sentop*|Pra] $\otimes_{sentop}$ Sec \u~[]/ _~[] → Prim \u~Pla /a~Pra
(108) `q\u~[]/[s^sentop]` $\otimes_{sentop}$ `s\u~[]/a~[]` → `q\u~[]/a~[]`
hoe 'how' ⊗ *vlieg jij* 'fly you' → *hoe vlieg jij*

1.6.2.2.7. */^adj*

The last right merge is a very tolerant one: the primary category does not affect the status of the secondary category. The latter imposes all its specifications on the output. Even the output flag in the active direction is dictated by the secondary category, instead of being *a~* or *w~* by default. The combinatorial and derivational effect of this merge is almost zero. The mode is particularly suited for the merges not rising from selection or subcategorization, but rather from adjunction or modification. Under these circumstances, the head types of the two categories will be equal, *Prim* = *Sec*.

(109) Prim_~Pla/_~[Sec^*adj*|Pra] $\otimes_{adj}$ Sec_~Sla/Rf~Sra → Prim\ Lf~Pla+Sla/Rf~Sra+Pra

As a typical case of this mode, consider the merge of a preposition introducing a verbal adjunct with a displaced *r*-pronoun as nominal complement and the verb:

(110) ... vp\u~[r^*wh*]/u~[vp^*adj*] $\otimes_{adj}$ vp\u~[np^*isl*]/u~[ ] →
vp\u~[np^*isl*, r^*wh*]/u~[ ]
(waar heeft hij het boek) mee 'with' ⊗ *geschreven* 'written' →
mee geschreven
'(what has he the book) with written'
what did he write the book with?

The specific order in the left output list follows from the redefinition of *append* (95).

This completes the set of rightward merge modalities, activated in the grammar of Delilah Dutch. There are seven of them, one of which, to wit */^transpipp,* is merely added to handle a peculiarity of Dutch syntax. The other six represent core modalities, in that they represent essential combinatorial processes in Dutch sentence formation. Some of the rightward merges discussed below are labelled like left-directional merges. Except for the island modalities, which represent rigid cancellation without any transfer of combinatorial agendas, they seem to be only directional images. In fact, there is little symmetry in the selection of merges for dealing with Dutch syntax. Linearization itself is the source of asymmetry.

1.6.2.3 Leftward Rules

It is attractive to call leftward cancellations backward, and rightward ones forward, following the terminology first used by Ades & Steedman (1982). It is also misleading, however, as this terminology refers to the parsing process instead of to the linearity of sequence formation. The direction of an argument may influence but will not control the parsing process.

1.6.2.3.1. \^*isl* for saturated constituents

The most rigid form of rightward cancellation (75) does have its leftward mirror image. The arguments cancelled under this mode are basically of the same types as the ones which are submitted to */^isl*: nominal phrases, determiner phrases, quantifiers. The standard case is the object arguments of verbs. The canonical position of these objects is to the left of the verb, if Dutch is an SOV lookalike.

(111) Sec_~[ ]/_~[ ] $\otimes_{isl}$ Prim_~[Sec^*isl*|Pla]/_~Pra → Prim\a~Pla/Rf~Pra

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

1.6.2.3.2. \^*transp* FOR AUXILIARY INVERSION

This mode is labelled like */^transp* since it applies to order variants of configurations in which that mode is invoked. Nevertheless, it enforces different types of input conditions. The primary category is forced to cancel its left argument first, whereas the secondary category is bound to be unaffected in both directions, and therefore lexical. The general formulation would be thus:

(112) SEC\u~SLA/u~SRA $\otimes_{\text{transp}}$ PRIM_~[SEC^*transp*|PLA]/u~PRA → PRIM\a~SLA+PLA/u~PRA+SRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The canonical application for this merge is the case of auxiliary inversion: certain auxiliaries or semi-auxiliaries may take the head of their verbal complements to the left, leaving the rest of their – or that head's – complement, if any, to the right. (For an intriguing and almost discouraging analysis of Germanic verb cluster phenomena see Wurmbrand 2006, and for encouraging efforts to formalize the phenomena in different syntactic settings see Van Dreumel and Coppen 2003 and Bouma and Van Noord 1996). Here are relevant examples with the modal auxiliary *kunnen*.

(113) … dwingen kan te werken
… *force can to work* 'can force … to make … work'
(114) … * willen kan laten werken
… *want can let work*
(115) … kan dwingen te laten werken
(116) … * dwingen te laten kan werken
(117) … * dwingen te laten werken kan

The phenomenon is not restricted to infinitival complements but also occurs with participles and the temporal and passive auxiliaries, under slightly different conditions. In particular, the auxiliary triggering the inversion can be infinitival just as well as finite.

(118) … dat Gezinus in het parlement probeerde te worden gekozen
… *that Gezinus in the parliament tried to be chosen*
… that Gezinus tried to be chosen in parliament
(119) … dat Gezinus in het parlement gekozen probeerde te worden
… *that Gezinus in the parliament chosen tried to be*

The relevant merge in constructing (113), for example, is:

(120) `vp\u~[np1^isl]/u~[vp^open]` $\otimes_{\text{transp}}$
`s_vn\u~[vp^transp,np2^isl]/u~[]` →
`s_vn\a~[np1^isl,np2^isl]/u~[vp^open]`
dwingen 'force' ⊗ *kan* 'can' → *dwingen kan*

Since primary categories introducing this mode will generally be auxiliaries or semi-auxiliaries with just one (verbal) complement, we may restrict the mode to situations in which the primary right argument list is empty, except for the verbal complement. Under these assumptions, format (112) boils down to:

(121) Sec\u~Sla/u~Sra $\otimes_{\text{transp}}$ Prim_~[Sec^*transp*|Pla]/u~[ ] →
Prim\a~Sla+Pla/u~Sra

Prim: head of *prim*ary category; *Plf:* *p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra:* *s*econdary's category's *r*ight *a*rgument list;
Lf: *l*eft flag (in resulting category); *Ra:* *r*ight argument list (of resulting category)

1.6.2.3.3. \^*open*

The most liberal of all merges does not impose any input restrictions whatsoever. As such it belongs to the small class of modes with these characteristics, differing from each other only in the output specifications.

(122) Sec_~Sla/_~Sra $\otimes_{\text{open}}$ Prim_~[Sec^*open*|Pla]/_~Pra →
Prim\a~Sla⊕Pla/Rf~Sra⊕Pra

The standard application here is to the automorphic argument of an adjunct, *e.g.* by an adverbial constituent to a VP. In that case, however, we might as well opt for some more restrictive version. For example, we could consider the primary category itself to be 'closed', in having no arguments unsaturated, except the one to be cancelled. Such a restriction would imply that the primary category itself represents an island; this, indeed, has generally been considered to be an identifying feature of adjuncts since Ross (1967). Here is this more restricted version.

(123) Sec_~Sla/Rf~Sra $\otimes_{\text{open}}$ Prim_~[Sec^*open*]/_~[ ] → Prim\a~Sla/Rf~Sra

Since the flag of an empty list hardly charges the relevant flag in the output, all output lists stem from the secondary category and the passive flag may also be equal to the passive input of the secondary category. The dominance of the secondary category, again, seems to be in accordance with the nature

of adjunctival modification. The primary category merely intervenes in the secondary structure. In the following example, *heeft* 'has' introduces the secondary category, and *waarschijnlijk* 'probably' the primary category to an \^*open* merge.

(124) Elke student heeft waarschijnlijk de boeken willen verkopen
Every student has probably the books want sell
'Every student probably wanted to sell the books'

For the sake of generality, however, we will stick with the liberal (122). It is subsumed by the more restrictive (123).

1.6.2.3.4. \^*wh* FOR THE MOTHER OF DISCONTINUITY

This is the rather basic merge mode that cancels a leftward-dislocated complex for its licensing argument. It is supposed to be the final move in the completion of an argument structure. At every previous merge, the argument with this mode was suppressed, in the sense of being kept at the bottom of the output stack, according to the revised definition of list append as clg_append in (95). Thus, when cancelling an argument under mode \^*wh*, there is no remaining active list in the primary category. The merge is designed to consume elements that in other frameworks may be considered to live in the specifier of a complementizer phrase. At output, the left argument list is flagged *w* for being affected by this particular cancellation (cf. section 1.6.2.1).

(125) Sec_~[ ]/_~[ ] $\otimes_{wh}$ Prim_~[Sec^*wh*]/_~Pra → Prim\w~[ ]/Rf~Pra

It seems mandatory for the secondary active (leftward) list to be empty. Since this category is assumed to represent a left edge element, it must be completed at that side. The condition that the secondary passive list is also empty amounts to the claim that only arguments can be cancelled in this way, not heads. In this sense, the condition reflects the percolation of a *wh*-marking to some maximal projection, inducing pied piping.
Applications of this merge mode comprise standard wh-movement phenomena, topicalization with verb-second and subject lefting. As for Dutch, we take them to be the same because they exclude each other: a classic structuralist argument. These structures are exemplified in the following set (for /^*sentop* see section 1.6.2.2.6); recall that *wh*-arguments come with double categories (star-categories; see (106)).

(126) `q\u~[]/u~[s^sentop]` $\otimes_{\text{sentop}}$ `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\u~[np^wh]/a~[]` → `s\w~[]/u~[]`
wie 'who' $\otimes^2$ *denk jij dat werkt* 'think you that works' → *wie denk jij dat werkt* 'who do you think is working?'

(127) `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\u~[np^wh]/a~[]` → `s\w~[]/u~[]`
die man 'that man' ⊗ *heb ik zien slapen* 'have I see sleep' → *die man heb ik zien slapen* 'that man I saw sleeping'

(128) `np\u~[]/u~[]` $\otimes_{\text{wh}}$ `s\a~[np^wh]/a~[]` → `s\w~[]/u~[]`
ik 'I' ⊗ *slaap* 'sleep' → *ik slaap* 'I am sleeping'

In (126), it is assumed that lexical argumentative *wh*-elements come with two categories, rather than one (see also at */^sentop* above). That is the main difference between these elements and the dislocated entities in (127) and (128). The relevant merges under *\^wh* as such are equal.

1.6.2.3.5. *\^adj* FOR FREELY OCCURRING ADJUNCTS

This is the leftward counterpart of */^adj*, introduced in section 1.6.2.2.7. It has the same motivation and application.

(129) SEC\SLF~SLA/_~SRA $\otimes_{\text{adj}}$ PRIM_~[SEC^*adj*|PLA]/_~PRA → PRIM\SLF~SLA⊕PLA/RF~PRA⊕SRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

Here is a characteristic example:

(130) … `s\u~[]/a~[]` $\otimes_{\text{adj}}$ `s\u~[s^adj,r^wh]/u~[]` → `s\u~[r^wh]/u~[]`
… *zaten zij* 'sat they' ⊗ *op* 'on' → *(daar) zaten zij op* 'on (that) they sat'

1.6.2.3.6. *\^part* FOR PARTICLES

Finally, in the leftward division, we need the inverse of an adjunct merge: a selected item the cancellation of which as a secondary category does not provoke affectedness.

(131) SEC\u~[ ]/u~[ ] $\otimes_{\text{part}}$ PRIM\u~[SEC^*part* |PLA]/_~PRA → PRIM\u~PLA/RF~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

There are all kinds of restrictions on the secondary input, as it is mainly applied to lexical particles that may occur separate from their licenser. Those particles behave as if they are part of that licensing phrase, in that they may

occur in positions in which other arguments of that licenser cannot occur. The example below has a resultative small clause predicate *groen* 'green' – such a predicate may occur with almost any transitive verb. It can take any position in a cluster where it is accessible as the left argument of the verb to be met first, independently of verb-raising configurations, though not every position is felicitous for every type of particle – but the \^*part* modality generalizes over 'real' particles and resultatives:

(132) ... dat Agnes Henk de auto groen wil laten verven
... *that Agnes Henk the car green wants have paint*
... 'that Agnes wants to have Henk paint the car green'
(133) ... dat Agnes Henk de auto wil groen laten verven
(134) ... dat Agnes Henk de auto wil laten groen verven

The relevant merge for (134) is

(135) `ap\u~[]/u~[]` $\otimes_{\text{part}}$ `vp\u~[ap^part,np^isl]/u~[]` → `vp\u~[np^isl]/u~[]`
groen 'green' ⊗ *verven* 'paint' → *groen verven* 'paint green'

Because the *u*-flag at the left argument list remains in the output, the resulting category can participate in the verb clustering directed by the */^transp* mode of the complement of *laten* (see section 1.6.2.2.2.).

Since particles of the sort targeted by this mode in a certain sense disturb the order of cancellation of the verb's left arguments, it makes sense to consider a little more liberal version of (131), where it is not required for the particle to be at the top of its stack.

(136) Sec\u~[]/u~[] $\otimes_{\text{part}}$ Prim\u~[... Sec^*part* ...]/_~Ra → Prim\u~Pla/Rf~Ra

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

It introduces an *anytime* cancellation strategy for a designated class of constituents, namely exactly those that are marked with the cancellation mode \^*part* by the lexicon. On the one hand, this complicates the derivation: every category has to be checked for containing a ^*part* argument in its left stack. On the other hand, it makes a huge set of lexical specifications redundant for every particle taking verbs in which the particle is just 'hopping' in the left argument list.

Since particles in Dutch seem to exhibit the *anywhere* property that (136) tries to account for (cf. Van Dreumel and Coppen 2003), we take this to be the canonical version of (131).

This completes the set of leftward merges. Again, the number is very restricted. One of the six, to wit (112), deals with a local inversion, the subtle auxiliary switch in the Dutch verb cluster. The other five represent merges in the core of Dutch syntax.

1.6.3 Dealing with adjuncts

In the exposition above, several modes were introduced aimed at dealing with adjunctive modification. The nature of this modification is automorphistic. An adjunct sends its argument to the same type the argument comes with. Consequently, adjunctive modification does not contribute essential ingredients to well-formedness or saturation. Moreover, adjunctive phrases are not very picky. They are usually capable of modifying different sorts of phrases, thus taking different sorts of categories as secondary category. In Dutch, they hardly impose conditions on the derivational status of their arguments. Finally, adjuncts themselves are not subject to discontinuous operations, peripheral to the sentential backbone as they are. They are rigid islands, with the exception of the extraction of so-called r-pronouns in adjunctive prepositional phrases.
In CLG, then, it is not very difficult to conceive of a category for an adjunct. For example, a lexical adjunct that can occur as the left modifier of a verb phrase – say: *snel* 'quick(ly)' – is categorizable as follows:

(137) `vp\u~[]/vp^adj`

The */^adj* mode is defined liberally enough in (109) to allow for all kinds of interference between this category and a constituent headed *vp*. Apart from (137), the word *snel* would also have to be assigned to categories taking sentences into sentences, nouns into nouns, and so on. This would yield a restricted class of assignments – recall that because of merge modes there is no need to come up with different categories for different arities of verbs, not even when immediate (left) adjunction to these verbs as heads of *vp* has to be enabled. The class would be unsatisfactory, nevertheless, since it is not very general and would have to be repeated all through the lexicon. As an alternative, adjunctive phrases may be assigned to a categorial classifier, to be instantiated at need in the combinatorial process. Such a classifier would

look like (137) but it would have variable typing. The variable is valuated in a closed finite class of types, *e.g.* {vp, s, n}.

(138) X\u~[]/u~[X^adj]

Subsequently, in a certain environment, a transfer rule is needed to give the relevant instantiation. As a matter of fact, since not all automorphic merges behave exactly alike, we could even parameterize the merge mode in assignment (138), having target type and merge mode specified in one rule, yielding the category

(139) X\u~[]/u~[X^M]

Derived merge rules will have this format:

(140) X\u~[]/u~[X^M] $\otimes_{M}$ vp\Lf~Sla/Rf~Sra → vp\Lf~Sla/Rf~Sra
only if
vp\u~[]/u~[vp^adj] $\otimes_{adj}$ vp\Lf~Sla/Rf~Sra → vp\Lf~Sla/Rf~Sra

(141) X\u~[]/u~[X^M] $\otimes_{M}$ s\Lf~Sla/Rf~Sra → s\Lf~Sla/Rf~Sra
only if
s\u~[]/u~[s^isl] $\otimes_{isl}$ s\Lf~Sla/Rf~Sra → s\Lf~Sla/Rf~Sra

It should be stressed that following such a track would not affect the outline of the grammar as given in section 1.4, but merely shorten lexical specifications. Equivalently, then, for every adjunct in the lexicon one can spell out at which types it applies and in what ways. This track is chosen in the present state of Delilah. The choice is mainly sensitive to the degree of sophistication with which the lexicon can be addressed.

As a matter of fact, the grammatical treatment of adjuncts puts a heavy burden on the grammar under any implementation. The more radical solution would be to treat adjuncts as optional arguments all the way down. This is essentially what Bouma and Van Noord (1994) suggested, and it is implicit in the cognitively inspired combinatoric model of Vosse and Kempen (2000). In our system, this proposal would amount to cancellation of arguments which would not affect the arity of the primary category: adjunctive arguments are optional as well as multiple. Here is one of the hypothetical rules for such a mode.

(142) Sec_~[]/_~[] $\otimes_{adj}$ Prim\Lf~[Sec^*adj*|Pla]/_~Pra →
Prim\Lf~[Sec^a*dj*|Pla]/Rf~Pra

Prim: head of *prim*ary category; *Plf: p*rimary category's *l*eft argument list *f*lag;
Sec: head of *sec*ondary category; *Sra: s*econdary's category's *r*ight *a*rgument list;
*Lf: l*eft flag (in resulting category); *Ra: r*ight argument list (of resulting category)

In this rule, the secondary category is the adjunct, to be eaten by a VP- or S-headed category, but without being cancelled from the left-argument list and without affecting the status of that list. The insularity of the adjunct is recognized in its empty argument lists.
There are clear grammatical advantages in treating adjuncts as optional arguments. In particular, all *wh*-types of operations on adverbial and adsentential adjuncts are easier to define if they are treated as arguments of higher functors in a (142)-type approach. Under the syntax given in section 1.6.2, it is simply not possible to handle a primary category as dislocated: all discontinuity is attributed to the way argument lists are merged. This merge cannot (re-) define or reconstruct the position of the primary category. Redefining adjuncts as secondary categories in the relevant merge would solve this problem.

The main problem for this redefinition, from a linguistic point of view, is that the argument approach to adjuncts seems more feasible for verbal functors than for nominal functors. Adjectives and relative clauses ad-modify nominal arguments, and the reversal of (142) would imply that nouns, too, become functors over optional modifiers. Such a strategy has been suggested by Janssen (1986) for relative clauses. The processing of nominal domains, however, would become more complex if nouns were optional functors.

Apart from the question whether it would be wise to have the adjunct at the top of the stack and not use the *anywhere*-proviso of section 1.6.2.3.6, (142) suffers from violating Count Invariance (47). That, however, would be very damaging to the (chart-) parsability of the system – see section 1.8. The only way to handle adjuncts as arguments, then, is to neglect them in deduction, introducing the following axiom:

(143) $\text{Adjunct}\backslash_\sim[\,]/_\sim[\,] \otimes_{\text{adj}} \textsc{Prim}\backslash\textsc{Lf}\sim\textsc{Pla}/\textsc{Rf}\sim\textsc{Pra} \rightarrow \textsc{Prim}\backslash\textsc{Lf}\sim\textsc{Pla}/\textsc{Rf}\sim\textsc{Pra}$

This introduces typological anarchy and global anywhere-ness. It violates the well-foundedness of merge (25)(b) and challenges the resource sensitivity of the grammar.
One way or another, reconstructing adjuncts as arguments involves trespassing on resource sensitivity, as can be read from the alternatives (142) and (143). Unfortunately, however, as will be made clear in section 1.8, placing adjuncts outside the frame of resource sensitivity endangers the system's computability. And computability is what we are after.
As a conclusion, then, the lexicon has to carry the load for syntax: multiple assignment of categories in a well-organized lexicon is easier to handle than

an increase of combinatorial power. We stick with adjuncts as primary categories, and pay the price of multiple typing.

This choice has one drawback that deserves special attention. Adnominal adjuncts can appear separate from their targets by the intervention of categories that eat their targets. The resulting structure is called *extraposition*. Here are two examples:

(144) Wie heeft de man zien werken met de hoed?
'Wie has the man seen work with the hat'
Who has seen the man with the hat work?
(145) Wie heeft de man zien werken die gisteren staakte?
'Who has the man seen work that yesterday striked'
Who has seen the man who was on strike yesterday work?

Under the present categorization, these sentences have to the following pattern (with simplified categories for clarity):

(146) ... np vp\np np\np

There is no way of bringing the adjunct category to its target before the target has been swallowed by a category – *vp* – that is intransparent to the adnominal adjunct. Now suppose that the *np* 'de man' would be categorized as looking for some adjunct to its right, and the adjuncts *met de hoed* and *die gisteren staakte* as primitive categories:

(147) ... np/adj vp\np adj

Clearly, then, there is an easy disharmonic composition of the two first categories (cf. (49)) bringing the negative and the positive versions of *adj* towards each other

(148) ... np/adj vp\np adj ⇒ ... vp/adj adj ⇒ ... vp

Treating adnominal adjuncts as arguments here would therefore have the major advantage of accounting for extraposition, and even of offering a theory of extraposition in terms of disharmonic composition. Following this track, however, every *np* would become ambiguous between looking for an adjunct and being saturated – a major complication in the derivation and increase of combinatorial power. The alternative of sticking with higher-order adnominal adjuncts calls for the ad hoc treatment of extraposition.

At this moment we have no decisive arguments for either strategy, and leave this question undecided.

1.7 THE GRAMMAR OF DISCONTINUITY AND COORDINATION

1.7.1 The source of discontinuity and connectedness

Natural language abounds in discontinuities: elements that depend on each other, do not necessarily come together, and are not necessarily adjacent. Grammar is the study of the nets rising from these discontinuous interdependencies. Yet, not everything goes. For a structure like

(149) A B A' B'

no grammarian of any language would offer the analysis

(150) ((A A') (B B')).

It is a heartfelt desire of grammarians to present analyses that are – in a very precise way – *connected*:

(151) *Connectedness*
For any sequence (A B A') the structure of the sequence is either (A (B A')) or ((A B) A').
There is an asymmetric relation $\mathfrak{R}$ such that (A B A') is a constituent only if either $\mathfrak{R}$(A, $\mathfrak{R}$ (B, A')) or $\mathfrak{R}$($\mathfrak{R}$ (A, B,) A')).

In Steedman (1990) this is presented as the invariance *Adjacency* for categorial combinatorics: you can combine categories only if they are adjacent.
In axiomatic calculi of categorial grammar without permutation, there are no values for types *y* and *w* such that the sequence

(152) a b y\a w\b $\rightarrow$ x

can be deduced for a product-free type *x*. The sequence is more discontinuous and less connected than these calculi can afford.

This does not mean that discontinuity and connectedness in natural language are unrelated. Our present framework, with categories that merge and have to express some linearization at the same time, discontinuity and connectedness paradoxically arise from the same source: the internal complexity of categories. It is important to see that this internal structure not only reflects linearization but also and equivalently induces semantic structure, generalizing dependency.

1.7.2 Discontinuity in CLG

In CLG, phrases are related to categories that express their selectional condition. If a phrase occurs discontinuously, *i.e.* separated from its selecting phrase, the nature of that discontinuity as well as the nature of the interveners must be expressed by the interaction of two other categories: the category of the selecting phrase and the category of the intervening phrase. The nature of the interaction of the two categories is given by the cancellation modality on the argument that complies with the head of the category of the selecting phrase. Here is an example with a right occurring selecting phrase and a left adjuncting intervener. *I* is the intervening phrase, *SD* the selected phrase, to be separated from *SG*, the selecting phrase. Cat_x stands for a category of phrase *X*. For the ease of the exposition, let us assume that *I* is the only intervening phrase.

(153) SD I SG
Cat_{SD} Cat_{I} Cat_{SG}

$Cat_{SD} = H_{SD}\backslash LF_{SD}{\sim}SLA_{D}/RF_{SD}{\sim}SRA_{D}$
$Cat_{I} = H_{I}\backslash LF_{I}{\sim}L_{I}/RF_{I}{\sim}[H_{SG}\hat{}MSG|R_{I}]$
$Cat_{SG} = H_{SG}\backslash LF_{SG}{\sim}[H_{SD}\hat{}MSD|SLA_{G}]/RF_{SG}{\sim}SRA_{G}$

(154) $Cat_{I} \otimes_{i} Cat_{SG} \rightarrow Cat_{I+SG}$
$H_{I}\backslash LF_{I}{\sim}L_{I}/RF_{I}{\sim}[H_{SG}\hat{}MSG|R_{I}] \otimes_{MSG} H_{SG}\backslash LF_{SG}{\sim}[H_{SD}\hat{}MSD|SLA_{G}]/RF_{SG}{\sim}SRA_{G} \rightarrow$
$H_{I}\backslash LF_{I+SG}{\sim}([H_{SD}\hat{}MSD|SLA_{G}]\oplus_{MSG}L_{I})/RF_{I+SG}{\sim}(R_{I}\oplus SRA_{G})$

(155) $Cat_{SD} \otimes_{j} Cat_{I+SG} \rightarrow Cat_{(I+SG)+SD}$
$H_{SD}\backslash LF_{SD}{\sim}SLA_{D}/RF_{SD}{\sim}SRA_{D} \otimes_{MSD}$
$H_{I}\backslash LF_{I+SG}{\sim}([H_{SD}\hat{}MSD|SLA_{G}]\oplus_{MSG}L_{I})/RF_{I+SG}{\sim}(R_{I}\oplus SRA_{G}) \rightarrow$
$H_{I}\backslash LF_{I+SG+SD}{\sim}((SLA_{G}\oplus_{MSG}L_{I})\oplus_{MSD}SLA_{D})/RF_{I+SG+SD}{\sim}((R_{I}\oplus SRA_{G})\oplus_{MSD}SRA_{D})$

H_X: head of Cat_X; *Mx*: cancellation mode for H_X; *PLF: p*rimary category's *l*eft argument list *f*lag; *SRA: s*econdary's category's *r*ight *a*rgument list; *LF: l*eft flag; *RA: r*ight argument list; R_X: *r*ight argument list introduced by category X

Note that the intervener types the merged phrase *SD+I+SG*. It is the primary category in the first merge, (155) cancelling the head of the selecting phrase, and determines the primary category in the second merge, cancelling the head of the selected phrase. Abstracting from direction, this is the only relationship between the phrases *SG*, *I* and *SD* that can be reconstructed by CLG and complies with the definition of connectedness (151). Connectedness is realized in CLG as generalized composition, taking over 'agendas' under merge. Because connectedness is interpreted here as composition, the category Cat_I, besides intervening, is also bridging between *SD* and *SG* and their respective categories. While intervening, it brings the two occurrences of H_{S_D} – the negative and the positive one – together in adjacent positions, after all.

In CLG, then, discontinuity is accounted for inasmuch as bridges exist or can be constructed. Connectedness has a clear-cut categorial condition, in the form of a predictable construal, within the limits of finite combinatorics, of bridging interveners. The construal of discontinuity is dictated by the specification of the merge modes involved in the bridging, M_{S_D} and M_{S_G} in the example above. In particular, they may maximize or minimize the effects of discontinuity by the way they append argument lists and by zero requirements on argument lists of the categories involved. Clearly, the merge under mode M_{S_G} may introduce additional discontinuity if L_I, SLA_G or R_I is not empty: this merge will mark the phrase associated with the primary category, or the phrase associated with the secondary category, or both as an intervener with respect to the arguments in these lists.

This construal of discontinuity in CLG is will be scrutinized in the next sections.

1.7.2.1 Managing discontinuity

To get a clear idea of what discontinuity under CLG is up to, consider two phrases *P* and *S* – for primary and secondary string, respectively – and their respective categories PRIM\PLF~PLA/PRF~[SEC^j| PRA] and SEC\SLF~SLA/SRF~SRA. Let *PLA*, *PRA*, *SLA* and *SRA* be sequences of strings with types that can be cancelled against the corresponding arguments in the categories of *P* and *S*. *PRIM* and *SEC* stand for the smallest phrases with the corresponding head categories, *i.e.* for the phrases introducing the primary and the secondary head types, respectively. Depending on the specification of the mode *j*, any of the following strings may model this family of merges. PRIM and phrases selected by it (*PLA* and *PRA*) are italicized for transparency. For each sequence, a number indicates how many of the four argument strings are separated from their head string. The dislocated argument strings are underlined.

(156) SLA *PLA* *PRIM* SEC SRA *PRA* :: 1
(157) *PLA* SLA *PRIM* SEC SRA *PRA* :: 1
(158) SLA *PLA* *PRIM* SEC *PRA* SRA :: 2
(159) *PLA* SLA *PRIM* SEC *PRA* SRA :: 3

Several aspects of this exercise are noteworthy. First, none of the four strings is fully continuous. As a consequence, every merge with four non-empty argument lists induces some form of discontinuity.
Secondly, the number of discontinuous pairs varies with the ways of appending (non-empty) argument lists.
Thirdly, all discontinuities respect connectedness, in the sense that there is a bracketing available which complies with (151). For example, (159) may be parsed as follows:

(160) (*PLA* ((SLA (*PRIM* SEC)) *PRA*) SRA)

This follows from the fact that absolute discontinuity of degree 4 is impossible: in the active direction – rightward in the examples – the primary argument string *PRA* cannot be dislocated, since it is bound to be more peripheral than the string introduced by the cancelled secondary head, its former companion. The degree of discontinuity, then, can be controlled by the grammar in either of two ways. One way is requiring one or both of the relevant input argument lists to be empty. The other way is appending the lists in such a way that arguments cannot intervene between other arguments and its selecting phrase. The latter way induces crossing dependencies. In that case, the primary category – the intervener – imposes as little discontinuity on the secondary category's combinatorics as possible.

Here are some instances of merge mode definitions to this effect, guided by the examples above. Only the elements relevant to the strategy are specified.

(161) PRIM_~[ ]/_~[SEC^j|_] $\otimes_j$ SEC_~SLA/_~_ → PRIM_~SLA/_~_
(162) PRIM_~PLA/_~[SEC^j|_] $\otimes_j$ SEC_~[ ]/_~_ → PRIM_~PLA/_~_
(163) PRIM_~PLA/_~[SEC^j|_] $\otimes_j$ SEC_~SLA/_~_ → PRIM_~SLA+PLA/_~_

The two ways of reducing discontinuity under the presupposition of connectedness are not independent of each other. Choosing the empty list strategy makes the append strategy redundant. There are two ways to avoid discontinuity by empty lists. If the secondary argument list is doomed to be empty, as in (162), that category is sealed to cover a (partial) island. If the primary argument list is marked empty, as in (161), this can only mean that the other

arguments – which are required to have been cancelled already – are more intrinsic to the primary phrase than the secondary argument.
In this vein, crossing dependencies as in (157) and (159) are the inevitable result of a close connection between primary and secondary category on one hand and linearization and antisymmetry on the other. In other words, crossing dependency is a normal configuration, induced by a sound strategy of appending non-empty passive argument lists in such a way that connectedness is assured.

The following table contains an overview of the discontinuous effects of every possible specification of argument list. The condition in that table means that the specification has the effect mentioned only if that condition is fulfilled. Otherwise, it is without any effect on linearization.

(164) *Table of merge effects for rightward cancelling*

specification	*combined with*	*linearization/bracketing*	*linearization effects*
PLA := [ ]		SLA (PRIM SEC .).	discontinuity
SLA := [ ]		PLA (PRIM SEC.).	continuity
PRA := [ ]		.(.PRIM SEC) SRA	continuity
SRA := [ ]		.(.PRIM Sec) PRA	continuity
SLA+PLA	PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC .).	crossing disc
PLA+SLA	PLA, SLA ≠ [ ]	SLA (PLA (PRIM SEC .).	discontinuity
PRA+SRA	PRA, SRA ≠ [ ]	.(. PRIM SEC) PRA) SRA	discontinuity
SRA+PRA	PRA, SRA ≠ [ ]	.(. PRIM SEC) SRA) PRA	continuity
PLF := u	SLA+PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC.).	crossing disc
PLF := a	SLA+PLA, SLA ≠ [ ]	PLA" (SLA ((PLA' PRIM) SEC.).	incoherent disc
SLF := u	SLA+PLA, SLA ≠ [ ]	PLA (SLA (PRIM SEC .).	crossing disc
SLF := a	PLA+SLA, PLA ≠ [ ]	SLA" PLA (PRIM (SLA' SEC .).	incoherent disc
SRF := u	PRA+SRA, SRA, PRA ≠ [ ]	.(. PRIM SEC) PRA) SRA	crossing disc
SRF := a	PRA+SRA, SRA, PRA ≠ [ ]	.(. PRIM (SEC SRA')) PRA SRA"	incoherent disc

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

The above scheme suggests a hierarchy in specifications of merge modes affecting linearization. The strongest effect results from emptiness conditions on input argument lists. Any such specification guarantees continuity or connected discontinuity at its side, without further specifications being necessary. Next are the ways of appending. Their effect is conditioned by

non-emptiness of any of the argument lists involved. Their effect, therefore, is dependent. Finally, the effect of affectedness specification subsists on the choice for particular appends, which were already dependent themselves. Thus, flags are the subtlest instruments for influencing linearization. Appends bring in discontinuity; flags regulate the nature of the discontinuity. Recall that flags are sensitive to derivational history in ways specified for Dutch in (74): a flag *u* encodes that the list has not hitherto been activated, or that it was not involved in the last few merges. As a rule, the conditions on argument lists model the phenomena as such, whereas the conditions on flagging correlate with conditions on phenomena.

The hierarchy renders some combinations redundant or nonsensical, or just marked. For example, it is not easy to see what any appending or flagging might add to the empty input specification. In the same vein, flagging specifications have effect only when append is discontinuous. Moreover, it may seem that there may be more options to the same effect. Compare for example the following merge modes.

(165) PRIM_~PLA/_~[SEC^*i* |PRA] $\otimes_i$ SEC_~SLA/u~SRA $\rightarrow$
PRIM_~PLA $\oplus_i$ SLA/_~SRA+PRA

(166) PRIM_~PLA/_~[SEC^*j* |PRA] $\otimes_j$ SEC_~SLA/a~[ ] $\rightarrow$
PRIM_~PLA $\oplus_j$ SLA/_~PRA

(167) PRIM_~PLA/_~[SEC^*k* |PRA] $\otimes_k$ SEC_~SLA/_~[ ] $\rightarrow$
PRIM_~PLA $\oplus_k$ SLA/_~PRA

PRIM: head of *prim*ary category; *PLF: p*rimary category's *l*eft argument list *f*lag;
SEC: head of *sec*ondary category; *SRA: s*econdary's category's *r*ight *a*rgument list;
*LF: l*eft flag (in resulting category); *RA: r*ight argument list (of resulting category)

At first glance, the linearization effects may appear similar: in each case, the relevant order of phrases will be . . *PRIM* SEC SRA *PRA*. The bracketings differ, however: ..(..*PRIM* SEC) SRA) *PRA* for (165), ..(..*PRIM* (SEC SRA)) *PRA* for the two other cases. Only the first specification ensures the merge of the primary category with a lexical secondary category. (166) excludes a lexical status for the secondary category, whereas (167) is indifferent in this respect. The options thus vary in their derivational consequences.

The line of reasoning followed above leaves no room for discriminating between processes handling continuous structure and processes dealing with discontinuity. All structure and all interpretation result from merging complex categories, and the mere merge of complex categories provokes forms and degrees of discontinuity. The opposition between *move*-type operations inducing discontinuity and *merge*-type operations inducing continu-

ous structures in generative grammar is counter-productive in CLG. If merge deals with linearization, move is redundant (Cremers 2004).

1.7.2.2 Discontinuity and minimalism

Generative grammar deserves credit for making explicit that discontinuities format sentence structure. In the minimalist emanation of the generative view, the distinction between *merge* and *move* (or *copy+merge*) highlights discontinuity as one of language's omnipresent properties. Moreover, the formulation of *move* as a one-structure-operation, handling features in one local tree, illustrates that discontinuity is neither unbounded nor random, but inherently determined by properties of the structure. The general idea is that *merge* builds structure, and *move* changes it, subsequently. This amounts to restyling the division of labour between context-free phrase structure rules and transformations, as established in earlier stages of generativism.
It is noteworthy that subsuming discontinuity under *move* does not mean that discontinuity is defined independently. As a matter of fact, there is circularity. Discontinuity is marked by applying *move*, and if something moves it is discontinuous.

Categorial grammar combinatorics does not favour a principal distinction between operations of *move* and operations of *merge*. In the first categorial approaches to *wh*-movement, for example, forms of type-raising were proposed for handling the interpretation of leftward dislocated elements. Later (*e.g.*, Moortgat 1988), special binary extract and insert operators were introduced to cover these configurations. They were subject, partially, to the same logic that governed the operators taking care of continuous sequences. From the point of view of linearization, however, these operators were of the 'anywhere' kind: *AB* was interpreted as the set of those split strings *ww'* such that *wbw'* is in *A* if *b* is in *B*, but the category gave no information as to where the string was to be invaded. In recent extensions of multimodal categorial grammar, as outlined in Moortgat (1997), the toolkit has been extended to such a degree that specialized indexed operators, exclusive to certain structural configurations, can handle discontinuous or permutative instances of linearization. Discontinuity is handled by specialized structural operations on marked structures. With these wide coverage formalisms, it is possible to translate the minimalist distinctions between *move* and *merge* into categorial terms, as was shown by Vermaat (1999).

Still, it has not been decided that *merge* and *move*, *i.e.* continuous and discontinuous linearizations, really are different processes needing distinct treatments

in a grammar. Scrutinizing the definitions of *merge* and *move* as presented in Stabler (1992), and putting technical details aside, we are led to assume the following distinction between the two operations. *Merge* deals with two trees where one tree occupies one position in the other tree. *Move* deals with two trees where one tree is related to two positions in the other. *Move* projects one tree twice, *merge* projects two trees once. Distinguishing these two processes is mandatory only to the extent that not every subtree of a tree is doubly or multiply projected in the end. That is, if every tree that is 'placed' by *merge* were to be doubly committed by *move*, one might as well come up with one single operation *mergemove* that performs the structure building and the position linking in one hit. In that case, discontinuous occurrence would be the fate of every phrase, in the exact sense that every phrase except the one associated with the top node links two or more not necessarily adjacent positions in the structure. Moreover, as *move* would be bound to *merge*, all double linking would be specified locally. It would amount to the conception that the occurrence of every phrase in a sentence is backed by a chain: to be a constituent is to be the head of a chain. Of course, this is not really far removed from having every configuration licensed by feature checking, *i.e.* by matching pairs of features which are introduced at different positions. Here is an attempt to formulate such a unified structure building operation *mergemove*.

Let every lexical element come with a three-leaf labeled stucture $[_X$ Spec [X Compl]] such that Spec and Comp have the same type. For example: an intransitive verb like *walk* would be assigned $[_{vp}$ NP [walk NP]]. Spec and Comp are supposed to be chained, having the same relevant features except for lexical content. At each lexical structure, the position in which the argument is lexicalized – either Spec or Comp – is marked; we will use italics to do so. The head is lexical *per se*. So, the category of *walk* might be $[_{vp}$ *NP* [walk NP]]. The structure $[_X$ Y [H Y]], having no lexicalizations at all, is supposed to be equivalent to just H. *Mergemove* is defined on every pair of trees $[_X$ *Y* [.X .Y]] or $[_X$ Y [.X .*Y*]] and $[_Y$ Z1 [. Y. Z2]], where either Z1 or Z2 is the lexical position and the dots mark possibly intervening right branching structures. It is assumed, however, that the head of every structure is uniquely defined as the highest position equal to the top label. This has the same effect as the directional head marking in Stabler's (1992) formalization of minimalistic combinatorics. This yields the following definition of *mergemove*:

(168) *Mergemove*
$[_X\ Y\ [.X\ .Y]] + [_Y Z1\ [.\ Y.\ Z2\ ]] \Rightarrow [_X Z1\ [_X\ Y\ [.\ X\ .[_Y.Y.\ Z2]]$
$[_X Y\ [.X\ .Y]] + [_Y Z1\ [.\ Y.\ Z2\ ]] \Rightarrow [_X Z1\ [_X Y\ [.\ X\ .[_Y.Y.\ Z2]]$

At *merge*, the lexical content of the head of the selected structure is placed in the position marked in the selecting structure. To see how things would turn out, consider the following mini-grammar of a language over the alphabet {1,2,3,4}. The lexicon would be the following set of structures:

(169) { [$_1$ *2* [1 2]], [$_1$ 2 [1 *2*]], [$_2$ *3* [2 3]], [$_2$ 3 [2 *3*]], [$_3$ *4* [3 4]], [$_3$ 4 [3 *4*]], [$_4$ e [4 e]] }

Clearly, 1 is supposed to select 2, 2 to select 3, and 3 to select 4. Below is the derivation of the number 4321.

(170) [$_1$ *2* [1 2]] + [$_2$ 3 [2 *3*]] ⇒ [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 *3*]]]]
[$_1$ 3 [$_1$ *2* [1 [$_2$ 2 *3*]]] + [$_3$ *4* [3 4]] ⇒
[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* 4]]]]]
[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* 4]]] + [$_4$ e [4 e]] ⇒
[e[$_1$ *4* [$_1$ 3 [$_1$ *2* [1 [$_2$ 2 [$_3$ *3* [$_4$ 4 e]]]]]]] (= [$_1$ *4* [$_1$ 3 [$_1$ *2* [1 2 *3* 4]]]])

The linearization of the italicized elements and the head position gives the desired result.
The choice of categories is far from trivial. Under the present lexicon, not every order of the digits is derivable, while all chains are nested around 1. In particular, no digit word can be derived in which 1 and 2 are not adjacent. This is due to the selectional restrictions implemented in the categories, *i.c.* the choice to have 2 selected by the element that invariably projects the top node. To exclude some more orders, one can limit the lexicon. For example, by taking the category [$_1$ *2* [1 2]] out, one excludes the derivation of any number representation in which 2 occurs to the left of 1, which is half of the original language. The extension and the nature of the language are thus completely localized in the lexical specification. The above implementation proves that one can consistently and non-trivially consider *merge* and *move* to be a single operation.

1.7.3 Patterns of Dutch

This section contains an overview of the specifications used in the concise syntax of Dutch of section 1.6.2. Instead of naming lists by direction, the table generalizes over directionality and names list (primary or secondary) passive or active, active being the direction in which the cancellation takes place. Only if some explicit value is required by the merge mode is this requirement specified in the relevant column. Even if modes seem to have the same specifications, they may differ as to exceptional output flagging, which is not accounted for in this table. Recall, furthermore, that appending of left lists

always implies down-stacking of a *wh*-argument in either list, according to provision (95).

(171) *Table of merge specifications for* DELILAH*'s grammar of Dutch*

PRIM\PPF~PP/_~[SEC^I|PA] $\otimes_i$ SEC\SPF~SP/SAF~SA → PRIM_~APPENDP/_~APPENDA
SEC\SAF~SA/SPF~SP $\otimes_j$ PRIM_~[SEC^J|PA]/PPF~PP → PRIM_~APPENDA/_~APPENDP

mode	*PP*	*SA*	*SP*	*PPF*	*SAF*	*SPF*	*APPENDP*	*APPENDA*
/^*isl*		[ ]	[ ]					
/^*transp*		[ ]		u		u	SP+PP	
/^*open*		[ ]		u			SP+PP	
/^*penins*		[ ]	[ ]/[x^*wh*]				SP+PP	
/^*transpipp*		[ ]		u	a	u	SP+PP	
/^*sentop*		[ ]	[ ]	u		w/u		
/^*adj*							PP+SP	SA+PA
\^*isl*		[ ]	[ ]					
\^*transp*	-/[ ]			u	u	u	PP+SP	SA+PA
\^*open*	-/[ ]						SP+PP	SA+PP
\^*wh*		[ ]	[ ]					
\^*adj*							PP+SP	SA+PA
\^*part*		[ ]	[ ]	u	u	u		

PRIM: head of *prim*ary category; *PPF, PAF: p*rimary category's *p*assive / *a*ctive argument list *f*lag;
SEC: head of *sec*ondary category; *SA, SP: s*econdary's category's *a*ctive / *p*assive *a*rgument list;
*SRA: s*econdary's category's *r*ight *a*rgument list

From this table, one can read some aspects of the merges needed for this fragment of Dutch. Before going into details, however, it should be stressed that other grammars of the same fragment may be equally efficient or more enlightening. One might even compare grammars with respect to the instruments they choose for determining the family of merge modes doing the job. Below, we will discuss the use of the discontinuity toolkit in DELILAH's grammar.

What immediately strikes the eye is that almost all input specifications of empty lists concern the secondary category. The only cases of such specification for a primary list represent an alternative formulation for a rule without such specification. These alternatives are driven by reflection, not by necessity, though. Thus, in this grammar the main load of discontinuity management is put on the secondary category. The primary category is often required to be unaffected in the passive direction and to allow the secondary passive agenda to be the first to be met, thus reducing the chance of incoherent discontinuity: the secondary arguments are close to the secondary head. It is, furthermore,

remarkable that in particular SA – the edge of the secondary category necessarily separated from the primary category – tends to be completed before the merge occurs.

None of the merges involves a requirement on non-emptiness of input argument lists. The possible specification that an input list be affected is used only once, in the mode */^transpipp*. This mode exclusively deals with *infinitivus-pro-participio* effects, which occur only in the presence of complex verb clusters. All other affectedness specifications require an input list to be unaffected.

As for the modes of append, it is noteworthy that the passive direction always complies with the side the primary category is on. Thus, from a directional point of view, the active appendings in the leftward cancellations all reflect the priority of the secondary left agenda, *i.e.* the imposition of continuity. Accordingly, the passive appendings in the rightward cancellations express the same priority, now invoking coherent discontinuity. The only exception here concerns a secondary left list that only contains a *wh*-element, this element being the only one to escape from the secondary domain.

As a whole, the grammar of Dutch according to this specification exists on only a few instruments of discontinuity management. The variety of means is rather limited.

(172) *Table of discontinuities in Dutch according to section 1.6.2*

mode	*PP*	*SA*	*SP*	*append mode*	*5*	*3*	*1*	*2*	*4*	*6*
/^isl		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^transp		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^open		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^penins		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^transpipp		[ ]		SP+PP	*PLA*	SLA	*PRIM*	SEC	*PRA*	
/^sentop		[ ]	[ ]			*PLA*	*PRIM*	SEC	*PRA*	
/^adj				PP+SP; SA+PA	SLA	*PLA*	*PRIM*	SEC	SRA	*PRA*
\^isl		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	
\^transp	-/[ ]			PP+SP; SA+PA	*PLA*	SLA	SEC	*PRIM*	*PRA*	SRA
\^open	-/[ ]			SP+PP; SA+PA	*PLA*	SLA	SEC	*PRIM*	SRA	*PRA*
\^wh		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	
\^adj				PP+SP; SA+PA	PLA	*SLA*	SEC	*PRIM*	SRA	*PRA*
\^part		[ ]	[ ]			*PLA*	SEC	*PRIM*	*PRA*	

PRIM: head of *prim*ary category; *PPF, PAF: p*rimary category's *p*assive / *a*ctive argument list *f*lag;
SEC: head of *sec*ondary category; *SA, SP: s*econdary's category's *a*ctive / passive *a*rgument list;
*SRA, SLA: s*econdary's category's *r*ight/*l*eft *a*rgument list; 1-6: peripherality indication

Table (172) shows the resulting discontinuities for the fragment. Here, the reference to active and passive direction is directionalized again as left and right lists and flags. The string positions are numbered from inside out, to indicate peripherality with respect to the kernel of primary and secondary head. Continuous structures are shaded. Italics indicate dependencies determined by the primary category.

From this table one can see that the grammar of Dutch – though infamous for its discontinuous verbal clustering – avoids many possible sources of discontinuity. With the notorious exception of the rules for left and right adjunction, all rules keep or may keep argument lists empty. Some rules do not introduce discontinuity at all – those requiring both secondary lists to be empty. Recall, furthermore, that in the active direction the primary arguments cannot occur discontinuously. As a result, discontinuity arises in one direction at most. Although we cannot deduce this result, it can hardly be accidental. Therefore, we present it as a conjecture on CLG-type grammars for natural language.

(173) *conjecture*
In a CLG grammar of a natural language, no rule imposes discontinuity in two directions.

Basically, the conjecture reduces the relevance of the grammatical formalism for 'real applications' to a relevant fragment of the formalism. Interestingly, this seems to be on a par with statements like the following:

(174) Although the set of terms of the full typed lambda calculus is *context-free*, the set of lambda terms characterizing the Lambek fragment of type logic (Lambek categorial grammar) is a *non-context-free subset* (Van Benthem 1991:109).
(175) Combinatory Categorial Grammar is mildly context-sensitive (Joshi *et al.* 1991).

The first statement refers to the fact that not every lambda term gives a recipe for a deduction in Lambek categorial grammar (see sections 1.4. and 1.8). Not every lambda term is a Lambek proof as well. The second statement says that a certain formalism transgresses the boundaries of context-freeness without falling into the full expressivity of context sensitivity (see also section 1.8). This is argued by Cremers (1999) to hold for CLG too. In both cases, powerful machinery is used only selectively when applied to natural-language grammar. Moreover, section 1.8.6 reports on Van de Woestijne's (1999) observation that Delilah's CLG does not seem to meet its *worst case complexity* when parsed; we now conjecture that this is because of (173).

1.7.4 Coordination as discontinuity

Coordinated phrases are inherently discontinuous. In the presence of coordinators, at least one phrase is separated from its selector or its selection by other phrases, among which is the non-connected coordinator. Coordinators have no special licensing relation with any phrase or category. Thus, they intrude on the combinatoric process. This is the case both for constituent and non-constituent coordination. Here are some examples; the peripheral edge of each of the conjuncts is indicated by a dot.

(176) Ik heb .haar echtgenoot. en .diens moeder. ontmoet
I have her husband and his mother met
'I met her husband and his mother'

(177) Ik heb .haar echtgenoot de fiets. en .diens moeder de auto. gegeven
I have her husband the bike and his mother the car given
'I gave the bike to her husband and the car to his mother'

(178) … dat ik .de jongens wil. en .de meisjes moet. begeleiden
… that I the boys want and the girls should supervise
… 'that I want to supervise the boys and should supervise the girls'

In the sentences above, the non-coordinated selector or selection that is related to both conjoined phrases is underlined. Each underlined phrase is separated from at least one of its relatives by at least the coordinator itself. Unless the coordinator itself is selected, this state of affairs marks non-connected discontinuity.

In the preceding section, we argued that discontinuity is a normal attribute of sentences in our grammatical analysis. Nevertheless, there are several good reasons to treat coordination as a structure of a nature different from other discontinuities. These reasons will be discussed below. They amount to the thesis that the type of discontinuity established by coordination is unconnected by necessity, in the sense of (151).
Under the present assumptions, our grammar is not suited to deal with this type of discontinuity. Therefore, DELILAH handles coordination by a distinct algorithm that is fed by the grammar and feeds onto it, but is steered by essentially different procedures and aims. The algorithm is introduced and accounted for in Cremers (1993a) and Cremers and Hijzelendoorn (1996), and will be discussed separately in section 1.8.8 as part of the DELILAH automaton. It sets coordination apart from the computation of other linguistic structures because the mechanics of these computations do not fit the nature and the scope of coordination.

The arguments for a dedicated treatment of coordinated structures are given below.

1.7.4.1 Conjunctions do not select

There are only a few, mostly focus-related restrictions on the occurrence of conjunctions and conjuncts in a sentence. The general rule is that if the conjuncts can be properly contrasted with each other at the level of information structure and are syntactically compatible, the conjunction is well-formed and interpretable. Well-formedness, then, depends on the information contour rather than on the structural properties of the sentence. A recipe for forming coordinated structures could be this. Take any well-formed sentence without coordination. Choose any position to the right of some phrase. Take some string to the left of that position. Construct another string that is contrastive to the selected one, and insert the conjunction with the constructed string in the chosen position. The result will generally be a well-formed coordinated structure.
The main challenge to this recipe is the coordination of singular filters generated by non-empty atoms (singular referential DPs) in subject position; their conjunction may violate agreement under this procedure. Elsewhere we argue that this phenomenon is neither syntactically nor semantically persuasive (Cremers 1993a: ch. 2; Cremers 2002).
There is no reason to assume that conjunctions are biased towards certain types of phrases or categories. In particular, conjunctions neither select nor license either of the conjuncts. Consider the following phrase.

(179) Geen enkele ambtenaar durfde mij een notitie over de begroting en ...
No civil servant dared me a memo on the budget and ...
'No civil servant dared ... (to) me a memo on the budget and ...'

Even when the focus structures of these phrases are given, no grammarian can tell by inspecting only this left environment of a coordinator which part of the phrase will turn out to be conjoined. As a matter of fact, every dot in the following representation of (179) may turn out to be the left edge of the conjunct, but only one – in rare cases of ambiguity: two – will be. The selection of this edge is not determined by the structure of the phrase.

(180) .Geen enkele ambtenaar. durfde .mij .een .notitie .over .de .begroting en ...

Being a conjunct, then, is not an independent configurational property of a phrase but the outcome of a process (Cremers 1993b, Harbusch and Kempen

2006, but see Houtman 1994 for a defense of the opposite view). *Conjunct* is neither a category nor a type. It is not even a function. We take the argument embodied by (180) to weigh heavily against any categorial characterization of conjunctions in languages like Dutch.
The argument is directed against proposals to consider conjunctions as types with essential variables or as multi-typed operators in categorial grammar, as put forward by Wittenburg (1986), Moortgat (1986), Steedman (1990), Houtman (1994) and others. The argument also opposes analyses of conjunctions as phrasal heads, as suggested by Johannesen (1997) and Munn (1993). It even runs counter to the 'classic' conjunction-reduction approaches, since grammatical processes of the transformational type target specified categories and/or constituents. But generative grammar seems to have left coordinative structures as a discrete object of study: none of the 77 chapters of the Companion to Syntax (Everaert and Van Riemsdijk 2006) focuses on them.

1.7.4.2 Conjunctions are not selected

A good argument for conjuncts playing a role in the semantics of certain maximality expressions was made by Postma (1995). He observes that in Dutch phrases like *met man en muis vergaan* ('with man and mouse sink'), *have en goed verliezen* ('stock and barrel loose') entail universality or totality, and that it must be the semantics of conjunction that brings about this effect. In a sense, the conjunction is selected by the verb as an expression of degree. Hoeksema (1988) and Poß (2010) put forward other lexicalised forms of coordination.
These expressions, however, are all frozen, and meaningful only to the extent that they are frozen. Notwithstanding Postma's observation that the conjunction may introduce maximality, it makes no sense to say that the conjunction is selected by the verb as a conjunction. It is selected as an expression of degree. Therefore, no phrase selects a conjunction. Conjunctions do not occur on the demand side of the grammar. Note that conjunctions and adjuncts differ in this respect. It is relatively unproblematic to reverse the normal automorphic analysis of adjuncts into an argument analysis in which adjuncts are selected, even with certain limitations, by other phrases (see section 1.6.3). Analyses along these lines have been put forward by, *e.g.* Bouma en Van Noord (1994) and Janssen (1986). Such a move is not conceivable for conjunctions. Conjunctions are that unspecific that selecting them as arguments would simply double (or more than double) the set of categorial specifications, according to the formula: if category X is selected then also *X and X* and *X or X*. No analytical gain would result from doing so, however.

1.7.4.3 Coordinations do not obey Count Invariance

Count invariance (47) – repeated below – is a major distinctive feature of resource- sensitive type logics and categorial grammars.

(181) *Count Invariance (repeated)*
For each axiom $\phi \rightarrow \psi$ in (44) and for every type t, t-count(φ) = t-count(ψ).

It identifies the combinatoric engine of these systems as establishing a relationship (*e.g.* cancellation) between two items fulfilling opposite roles in the structure. By definition, coordination entails the *multiplication* of certain roles in a configuration, *i.e*. it spreads identity rather than opposition. In the presence of coordination, the balance between selectors and selectees breaks down, not by accident, but by neccessity. Only by considering conjunctions to be unselective automorphic selectors can one overcome this imbalance. This is effected by assigning them to type $(x\backslash x)/x$, where the two negative occurrences of x restore the balance against the two positively occurring conjuncts which they aim to cover. Another strategy advocated by Dalrymple *et al.* (1995) is to weaken the resource sensitivity of their (linear) logic. In either approach, however, count invariance (a reflection of resource sensitivity) loses its edge: we cannot tell which is the balance to be restored before having parsed the sentence in order to identify the conjuncts that bring in the type doubling. And we must parse because the conjuncts themselves are not identified locally. This paradox of coordination is addressed in Cremers (1993a) and Cremers and Hijzelendoorn (1997a). In the latter it is argued that the best counts we can get in the presence of coordination are (complex) inequalities, whereas count invariance is expressed in terms of equalities.

1.7.4.4 Coordinations do not mean

In this section, we will argue that a conjunction *i.e.* the unit formed by the two coordinates and the coordinator, in general cannot be interpreted prior to the interpretation of the sentence or sentences which the coordination is part of. First, recall the statement from preceding sections that the identification of the coordination is the result of parsing, not a sub-process of it, like the identification of any other constituent. This alone already implies that the meaning of the conjunction cannot contribute independently to the meaning of the sentence that is construed from parsing it. But even if we abstract from this contradiction, and assume that coordinations contribute to the meaning construal by oracle, it is hard to determine what that contribution could be. Suppose we were, by oracle, capable of identifying any conjunction as some constituent of a certain

category with structure $[_{cat}$ *cat conj cat]*. In order to establish a meaning for it, we must be able to identify an operation executed by the conjunct on the meanings of the conjuncts, yielding one 'merged' meaning. It is tempting to interpret *conj* as a certain operation defined on the algebra D_{cat} of denotations of category *cat.* It is well-known, though, that conjunctions do not show boolean behaviour, in the sense that they denote a fixed algebraic operation on their arguments' algebra. The standard example is with quantifiers, *i.e.* denotations of type *np*. In sentence (182), the coordination is most likely interpreted as the meet over the two quantifiers it conjoins: what is predicated of the coordination is predicated of each coordinate. In (183), on the other hand, nothing is predicated of each of the coordinates. Many scholars have suggested that we need something like a group denotation to get the semantics in order here. In (184) it is rather the join over the two denotations that should be the coordination's contribution to the sentence's meaning. The reason is quite clear. The coordination of (184) does not denote the set of spots on earth that are both Norwegian and Swedish, but rather the set of spaces that are (either) Norwegian or Swedish.

(182) De regering heeft [Smit en De Vries] bevorderd tot ridder in een of andere orde
'The government has Smit and De Vries promoted to knight in one or other order'
(183) De regering heeft [Smit en De Vries] belast met het onderzoek
'The government has Smit and De Vries charged with the investigation'
(184) In Noorwegen en Zweden rookt niemand meer
'In Norway and Sweden smokes nobody anymore'

The semantic effect of coordination may depend on functional properties of its environment, as was made abundantly clear by Zwarts (1986), and on denotational properties of the predication that the coordination is involved in (see Link 1983 for the original argument and *e.g.* Winter 2001 for an extensive treatment). These interactions may be steered by laws of logic and algebra, but it is impossible to identify an independent contribution to the semantics of a sentence made by coordination. Rather, the opposite is the case: in order to define the nature of the coordinative linking we have to interpret the sentence it is part of.

Does this position endanger compositionality? Janssen (1997) considers arguments against compositionality, as brought forward by Higginbotham (1986) with respect to the interpretation of *unless*. It is observed that *unless* behaves differently in different functional domains.

(185) Every person will eat steak unless he eats lobster
(186) No person will eat steak unless he eats lobster

In an upward-entailing environment such as the nuclear domain of a universal quantifier, *unless* denotes disjunction. In a downward-entailing environment it denotes conjunction. Janssen, who considers compositionality to be a requisite of grammatical architecture rather than a decidable property of language, offers two ways out in this case: either *unless* is constructed as a function from contexts to values, or it is considered to be ambiguous. We may try to subsume the context dependency of constituent coordination under this analysis. The first solution – context valuation – would not solve our problem, because there is no definite context available for coordination (see section 1.7.4.1). *Unless* precedes or follows a proposition offering the relevant context. Coordination is part of the context that should license its interpretation, since there is no proposition independent of the coordination. Moreover, coordinations may be part of each other's contexts, yielding circular traffic under this approach.
The second solution – inherent ambiguity – is the last resort. For conjunctions it would amount to a huge number of ambiguities, none of which would be locally decidable (Cremers 1993a). If conjunctions were multiply ambiguous, one would expect languages to come up with specialized elements, reducing ambiguity. Though some languages have a specialized element for group formation (a kind of 'with' element, see Payne 1985) these languages do not have explicit sentential coordination at all. Languages that have propositional conjunction, though, use it in all other environments, introducing massive ambiguity under this approach.

The propositional (and boolean) base of coordination is reflected in the proposal in Partee and Rooth (1983) to interpret all coordinations as the meet of propositions: |[|[a]| and |[b]|]| = $\lambda\varphi$. [φ(|[a]|) $\wedge$ φ(|[b]|)], where φ's type can be derived from the coordinates' type. The argument to this function must be provided by the sentential construction surrounding the coordination. Here, the local indeterminism of coordination strikes back: such remnant environment cannot be established without parsing the whole construction and determining the coordinates. But that is the problem. Though Partee and Rooth succeed in generalizing the typological aspect of the meaning of the coordination, their proposal does not solve the conjunction paradox: in order to determine the coordination's contribution to the semantic construal of parsing, parsing must be completed including the determination of the coordinates. It is because of this paradox that we claim that there is no local semantic construal for coordination. The meaning of *John or Bill* and *John to swim and Bill to walk* does not exist independently of the semantic construal of a sentence.

The arguments above entail that conjunctions be treated *syncategorematically*, not subjected to the core categorial engines of grammar, as proposed in Grootveld (1994) from a generative perspective and in Cremers (1993a). We take Kempen's (2008) analysis that coordination is an updating construction, to be in the same spirit.

1.7.4.5 Ellipsis is coordination speciale

Everaert and Van Riemsdijk (2006) not only subsume coordination under other constructional phenomena but also the massive handbook lacks a chapter on ellipsis. That is not accidental. Coordination and ellipsis are intrinsically related, along the following lines of reasoning.

Firstly, to be elliptical is a prerogative of sentences. It does not make a lot of sense to talk about incomplete VPs or DPs, as they may occur discontinuously or underspecified in languages under normal business. Secondly, ellipsis calls for a propositional interpretation. There is no reason to call *six* an elliptical DP in the sentence *I have six*, but it is inevitable to call *John six* elliptical in the sentence *I had five and John six*. Third, ellipsis needs textual context to be interpretable. The sentences *John had five* and *I'll catch it* are perfectly interpretable as propositions without any reference to textual context. This is characteristic of certain anaphora, as was pointed out by Hankamer and Sag (1976). The sentence *John six* has no propositional interpretation outside a textual context licensing some semantic reconstruction. It is elliptical.

In short, we take ellipsis to be sentential, propositional and textual. In this vein, ellipsis is essentially defined as coordinative. It introduces a phrase of a category that is the canonical target of coordination and is interpreted as a proposition, and it has the same inferential status as the context it is licensed by – the logical function of conjunction.

This property can be illustrated by having a close look at instances of ellipsis for which coordinative analysis is not obvious, like VP ellipsis and comparatives; the elliptical phrases are in italics.

(187) Sue called John before *Bill did*
(188) Sue painted John more realistically than *Bill did*

Clearly, (187) is equivalent to the following sentence.

(189) Sue called John and Bill called John and Sue's call was earlier than Bill's call.

Bill called John is entailed, and this is because it is in 'semantic conjunction' with the main sentence: a conjunction of propositions entails each of its conjoints. In the same vein, the meaning of (188) must be represented as

(190) Sue painted John in a certain manner and Bill painted John in a certain manner, and Sue's manner of painting John was more realistic than Bill's manner of painting John.

Again, there is no difference in inferential status between the ellipsis and its licenser. The coordinative nature of comparison was pointed out by Hendriks (1995). The semantic and inferential co-ranking of ellipsis and its textual antecedent – backward ellipsis is a *contradictio in terminis* – explains why VP ellipsis cannot occur in sentences like

(191) * Sue sang so loudly that *Bill did*

VP ellipsis fails here, because the relationship between the intended propositions *Sue sang (with loudness x)* and *Bill sang (with loudness y)* is not just conjunction at top level, but (asymmetric) causation.

Elliptical constructions like gapping, sluicing and answering are explicitly coordinative, and bound to their antecedents in the sense that they cannot afford any textual intrusion.

(192) * I visited John, Sue went to Amsterdam and Mary Bill.
(193) * John visited someone, Sue went to Amsterdam but I forgot whom.
(194) * Who went to Amsterdam? Why are we here? John.

Therefore, we assume that ellipsis occurs exclusively in the (right) context of (semantic) coordination of sentences. Yet, coordination and ellipsis are not the same – coordination is basically not reductional but augmental. Under the present analysis, ellipsis is a subcase of coordination, a combination of augmenting and rearranging a preceding structure. Since we handle coordination syncategorematically, ellipsis is treated this way too.
The algorithm dealing with coordination identifies ellipsis as those instances of coordination for which categorial directionality is weakened. To illustrate the point, compare

(195) Destijds wou ik Susan en jij Marie benaderen
'in those days wanted I Susan and you Marie approach'
In those days I wanted to approach Susan and you, Mary
(196) Destijds wou ik Susan benaderen en jij Marie
'in those days wanted I Susan approach and you Marie'
(197) Destijds benaderde ik Susan en jij Marie
'in those days I approached Susan and you Mary'

The coordination algorithm deals with the first sentence straightforwardly: the components of both conjoints are adjacent at the level of the sentence-to-be. The second sentence is elliptic. To the left of the coordinator a full propositional sentence occurs; the right-hand side is not interpretable as such. The right-hand conjoint cannot be mapped onto a coherent sequence at the left. The conjunction is sentential, in all respects. (197) seems to be in between: there is a full sentence to the left of the conjunction, but the right-hand conjoint maps onto a coherent sequent of constituents to the left. But the sentence is not ambiguous in any interesting sense. Ambiguity, however, would be expected if coordination and ellipsis were different procedures. As a matter of fact, the sentences (195) - (197) share non-directional aspects of the argument structure – their *arity* and linearity. In order to parse ellipsis, then, we can restrict ourselves to applying the coordination algorithm under relaxed directionality in the presence of a full sentence. The computational nature of this algorithm will be discussed in section 1.8.8.

1.8 PARSING THE SYNTAX

The Chomsky hierarchy of formal languages (e.g. Hopcroft *et al.* 2001) connects languages, grammars and automata by stating that if languages are sets, their grammars embody programs to determine membership and that the differences in memory management between programs may reflect differences in the grammar. The hierarchy used to be the dominant frame for evaluating grammatical and procedural properties in the early days of computational linguistics, when storage was expensive and processing slow. But it is not technological progress alone that diminished the Chomsky hierarchy's importance. Firstly, theoretical linguistics itself seems to have lost its interest in the hierarchy: both for the generative enterprise and for its competition – from categorial grammar through HPSG to construction grammar – the hierarchy is too crude to catch up with the tendencies towards semantically relevant, fine-grained and construction specific grammatical analyses; there is more than context-freeness on earth and in heaven (*e.g.* Van Benthem 1991). Secondly, the parsers themselves turned out to be intrinsically more complex than the grammars or the grammatical structures which they were supposed to operate on as far as natural language was concerned (McCawley 1968): even if one describes natural-language grammar as a complex of regular structures, the overall system regulating the labour division is certainly not just finite-state.

Thirdly, of course, the hierarchy became less relevant in computational linguistics as grammars tended to be avoided in favour of non-symbolic models. Yet, the nature of the parsing process of a symbolic grammar is a central benchmark for two evaluations: first, comparison of the complexity of the natural-language recognition and interpretation problem with other problems and, second, its pretentions as a model of human language processing. We assume that these are different and even independent valuations. To say that natural-language recognition is nearly as hard as the Tower of Hanoi, chess or the Salesman problem is saying little about human language processing. To say that human language processing leans on memory routines rather than on on-line computation (*cf.* Daelemans and Van den Bosch 2005) is saying little about the problem of its mathematical complexity. Both in the domain of complexity as well as in that of modelling processing, the nature of the parsing procedure is distinctly relevant.

In this section we will reflect extensively on the DELILAH parser, by comparing it to the parsing of other grammatical frames, in particular to Combinatory Categorial Grammar (CCG) as revealed in Steedman (2000), for example. The section is largely based on previous work by Peter Zinn and Christiaan van de Woestijne (Zinn 1993, Van de Woestijne 1999); the latter author also developed the core of the chart parser. The coordination algorithm was added to the chart parser by Mathieu Fannee (Fannee 2006). Robustness strategies at the syntactic level were investigated by Poeder (1994).

1.8.1 The syntax of CCG

As stated earlier, CLG is closely related to Combinatory Categorial Grammar (CCG). The definition of CCGs below is based on Joshi *et al.* (1991).

(198) *Definition*

A CCG, G, is denoted by (V_T, V_N, S, f, R) where
V_T is a finite set of terminals (lexical items),
V_N is a finite set of non-terminals (atomic categories),
S is a distinguished member of V_N,
f is a function that maps each element of V_T to a finite set of categories,
$C(V_N)$ is the set of categories, where
$V_N \subseteq C(V_N)$ and
if $c_1, c_2 \in C(V_N)$ then $(c_1/c_2) \in C(V_N)$ and $(c_1\backslash c_2) \in C(V_N)$,
R is a finite set of combinatory rules.

CCG's central rule-scheme is generalized composition, again according to Joshi *et al.* (1991). It comes in two directions.

(199) *Generalized Composition forward*
$(x/y)\ (y|_1z_1|_2 \ldots|_mz_m) \Rightarrow (x|_1z_1|_2 \ldots|_mz_m) \qquad (m \geq 0)$

(200) *Generalized Composition backward*
$(y|_1z_1|_2 \ldots|_mz_m)\ (x\backslash y) \Rightarrow (x|_1z_1|_2 \ldots|_mz_m) \qquad (m \geq 0)$

Here, $x, y, z_1, \ldots, z_m$ are meta-variables over $C(V_N)$, and each $|_i \in \{\backslash, /\}$. For m=0, these rules correspond to function application and for $m > 0$ to function composition. The set R contains a finite subset of these possible forward and backward rules, i.e., for any given CCG only some of the combinatory rules will be available. Furthermore, restriction conditions can be put on calling combinatory rules in R. Rules can be constrained in two ways: the initial non-terminal of the category to which x is instantiated (the head) can be limited, or the entire category to which y is instantiated can be restricted. In Joshi *et al*. (1991), the mapping function f may also assign categories to the empty symbol, ε. CLG does not use this feature. Excluding the empty category makes CLG cycle-free: there is no non-terminal category $A \in C(V_N)$, such that $A \Rightarrow^+ A$, that is, no A can derive itself in one or more steps. CLG is cycle-free because each derivational step cancels exactly one category, while empty moves cannot occur (cf. section 1.5.4). Note that for m=0, these rules simplify to the Forward and Backward Application rules of classical AB grammars (Ajdukiewicz 1935, Bar-Hillel 1953).

1.8.2 Comparing the syntaxes of CCG and CLG

In CLG, rules (199) and (200) have been adapted. Rule (199) has been replaced by two instantiations, and repeated here for convenience. These two rules differ only in the order in which the lists of arguments are appended. These rule types are the only ones in CLG, as merging is the only combinatory operation in the syntax (cf. section 1.4.3).

(201) *Generalized Composition forward in CLG, discontinuous*
$(\ldots(p|w_1)|w_2\ldots.|w_m)/y\ \ (\ldots(y|z_1)|z_2 \ldots |z_n) \Rightarrow (\ldots(\ldots(p|z_1)|z_2 \ldots |z_n)|w_1|w_2\ldots.|w_m)$

(202) *Generalized Composition forward in CLG, continuous*
$(\ldots(p|w_1)|w_2\ldots.|w_m)/y\ \ (\ldots(y|z_1)|z_2 \ldots |z_n) \Rightarrow (\ldots(\ldots(p|w_1|w_2\ldots.|w_m)|z_1|z_2 \ldots |z_n)$

In the same vein, backward composition (200) has been instantiated by two CLG rules.

(203) *Generalized Composition backward in CLG, discontinuous*
$(\ldots(y|z_1)|z_2 \ldots |z_n) \quad (\ldots(p|w_1)|w_2\ldots.|w_m)\backslash y \;\Rightarrow\; (\ldots(\ldots(p|z_1)|z_2 \ldots |z_n)|w_1|w_2\ldots.|w_m)$
(204) *Generalized Composition forward in CLG, continuous*
$(\ldots(y|z_1)|z_2 \ldots |z_n) \quad (\ldots(p|w_1)|w_2\ldots.|w_m)\backslash y \;\Rightarrow\; (\ldots(\ldots(p|w_1|w_2\ldots.|w_m)|z_1|z_2 \ldots |z_n)$

The instantiations are logically not neutral. The CCG forward rule (199) is entailed by the discontinuous (201) but not by the continuous CLG forward rule (202). CLG allows for one more merge mode than CCG.

CLG defines several restrictions and alternations in comparison to CCG. The most relevant ones are listed below.

(a) in CLG, categories are required to be linear, that is, without any bracketing (Cremers 1989, 1993a). They may be represented as $y|_1z_1|_2 \ldots|_mz_m$, where y is primitive or atomic, and $m \geq 0$. A linear category is a particular instance of a CCG category. Compared to Lambek's set {n,s} of non-terminal symbols, CLG uses a larger and more fine-grained set. Yet, CLG non-terminals could be analyzed as fixed or de-activated {n,s}-complexes; the CCG category *np* represents a lambekian category *(n,(n,s)),* but is not addressed as such by the syntax. As far as we can see, nothing hinges on the cardinality of the primitives from a syntactic point of view; Roorda (1991) proved that even a one-type-categorial logic may work to distinguish what has to be distinguished, and Montague (1972) operates a two-type semantics but declares the object atom *e* (the lambekian *n*) syntactically inert. Regarding the number of non-terminals, there is no fundamental difference between one-, two- and multi-atoms approaches.
In CLG, due to categories being linear, an argument can only be cancelled against the primitive head of the secondary category. More accurately, when both the primary and the secondary category have been fully expanded and argument lists are not empty, *disharmonious composition* cannot be avoided in the sense that argument lists in the passive directory are also merged: disharmony is the standard option for composition when linear and complex categories are involved (cf. section 1.4.8). This is no different in CCG. As can be read from the composition schemes (199) and (200) where vertical slashes abstract from directionality, arguments of the secondary category pass over onto the resulting category even when their direction is opposite to the slash that induces the cancelling. In (205), you find two simple but legitimate instances of generalized composition, showing disharmony in the italicized negative type /*z*.

(205) $x/y \quad y\backslash z \;\Rightarrow\; x\backslash z$
$y/z \quad x\backslash y \;\Rightarrow\; x/z$

In standard categorial grammar of the lambekian breed, composition and application allow both for complex heads and for cancelling of complex types. This will not induce structural ambiguity, as the internal structure of complex heads is fixed by a particular bracketing. In the case of $np \otimes (s\backslash np)\backslash np$, the rightmost *np* is the only candidate for cancellation. Of course, the complex head can be abbreviated by an atomic label. However, atomic labels for complex heads hide linguistic information and will yield an uninteresting grammar (Zinn 1993). The computationally interesting property of using simplex labels as the trigger for rule invocation is applied in CLG, and formulated as *linearity*. When heads are required to be primitive (simplex) categories without any structure, calling a rule boils down to matching two basic (simplex) types – which is a cheap test.

(b) In CLG, the form of the categories has been adapted for both computational and linguistic purposes. CLG assigns categories of the format $x\backslash_1 l_1 \backslash_2 \ldots \backslash_m /_1 r_1 /_2 \ldots /_n$, where x is a simplex head, and l_i and r_i constitute lists L and R of left and right arguments, respectively, taken from V_N, and m, n ≥ 0. A category may then be written as $x\backslash L/R$. L and R may be the empty list [ ], for m=0, and n=0, resp. Non-empty lists L and R have been split further into regular (L_{reg}, R_{reg}) and special arguments lists (L_{sp}, R_{sp}). L_{sp} accommodates wh-movement or, better, any form of peripheral dislocation. Its size is one element at the most, kept at the end of the list of leftward searching arguments (L is a 'priority stack'). Its counterpart R_{sp} is currently not used (it is the empty list). In the definition of rules, CCG's general operator $|_i$ has been instantiated by the directional equivalents $\backslash_i$ and $/_i$. Moreover, the relative order of the leftward searching arguments is reversed with respect to the order displayed in traditional category notation: the leftmost element of both lists represents the 'searcher', which is to be satisfied first; it is the innermost argument. Thus a representation of the argument lists as Prolog lists – stacks, essentially – becomes straightforward.

(c) Some words – typically *wh*-words – are assigned a pair of categories, $C^{left} * C^{right}$, where C^{left} and C^{right} are single categories from $C(V_N)$. Such a pair is called a *double type* (or star-category). C^{left} and C^{right} have different selectional roles and can never be cancelled against each other, by definition (see section 1.6.2). Double types are not derived dynamically by some rule that can be applied repeatedly, but are defined statically in the lexicon.

(d) All arguments of CLG's categories are flagged by *modes*. These flags enforce certain restrictions on the applicability of reduction rules. For example, a flag

may require an argument list to be empty. Moreover, every rule instance must assign a flag value to the head of the resulting category. Modes of composition are included in CCG as well (Joshi *et al*. 1991, Baldridge and Kruijff 2003).

(e) With respect to the reduction rules used, CLG's Generalized Composition rules obey Count Invariance (see section 1.4.6), directionality and adjacency. Directionality means that if a category *c* asks for some argument category *c'* on its left, it should find the lexical element which introduces the head of *c'* to the left of the lexical constituent bearing the category *c*. The same reasoning is valid for right-searching arguments. Directionality prohibits permutation from being introduced. Steedman (1990) states that any combinatorics is to be limited by some form of adjacency. The inference engine of the grammar is supposed to instantiate the types in the antecedent of the axiom schemes only by adjacent types in the sentence, in the order given by the scheme. This restriction, too, prevents permutation from slipping in (Cremers 1993a). It can be reformulated by requiring that axioms with antecedent strings of more than two members are not allowed. This is stated by the Binary Derivability lemma (64), which excludes rules of the form $x \Rightarrow y$, where *x* and *y* are single non-identical categories, any form of type raising, i.e. $x \Rightarrow y\backslash y/x$, permutations of the form $x\,y \Rightarrow y\,x$, expansions of the form *if* $x \Rightarrow y$ *then* $x\,x \Rightarrow y$, contractions of the form *if* $x\,x \Rightarrow y$ *then* $x \Rightarrow y$, nor do they accept or derive empty symbols, ε. Van Benthem (1991) shows that (some combination of) rules like Permutation, Expansion, and Contraction even decrease the generative capacity of categorial grammars.

(f) CLG's rules are degree-decreasing: the number of basic categories on the right hand side is strictly lower (*i.e.* <) than that on the left hand side. Here, heads and arguments of the same basic type are cancelled in accordance with the Van Benthem count invariance protocol (Van Benthem 1986). Cremers and Hijzelendoorn (1996, 1997a, and 1997b) extend count invariance to free coordination and left dislocation.
Degree-decreasing rules imply that the language membership of CLG is decidable, as the height of the derivation tree is bounded upward by the length of the input string. To put it differently, for CLG, being a cycle-free grammar (cf. section 1.8.1), the length of a derivation – *i.e.* the number of steps – is linear in the length of the generated string. Exactly this theorem was proven for *context-free* cycle-free grammars by Harrison (1978:418). Stabler (1992) proves a closely related version: for any cycle-free context-free grammar, there is a linear bound to the size of the derivation tree – *i.e.* the number of nodes – as a

function of the length of the terminal string that is the yield of that tree. Both theorems illustrate the close relationship between CLG and CFG.

1.8.3 Comparing the generative capacity of CCG and CLG

The format of the categories in CLG differs from the format in CCG in several respects. We discuss the main differences, and explore the consequences for the generative capacity.

(a) In CLG, the category's arguments have been split into argument lists for leftward and for rightward searching arguments. The first element refers to the innermost argument. This gives greater freedom in choosing to reduce to the left or to the right, while in a standard categorial format this order is explicitly specified in the order of the arguments. This increases the strong recognizing capacity of CLG. That is: CLG recognizes more different structures than CCG. Maintaining more than one argument list does not affect the weak recognition capacity of CLG (Cremers 1999). CLG and CCG share the same string language, but may assign different structures to it. In CCG, using only categories in standard format, the same strong recognizing capacity can be achieved by adding categories created by mixing the left and right arguments in a different way from the categorization of the word. Because of Lambek's theorem $(a/b)\backslash c \Leftrightarrow (a\backslash c)/b$, these categories are (weakly) equivalent. The theorem is responsible for generating an exponential number of analyses for a constituent. Any derivation tree over CLG can be transcribed into a tree containing only categories in standard form. In principle, lexical entries might be assigned a large number of different categorizations. Because of rule restrictions, however, each lexical entry has only one typical categorization that is effectively taken into account. Thus, the lexicon can be organized in such a way that any category is supplied with one list of arguments (modulo the list of special arguments) in a standard order. This eliminates ambiguity on the lexical level. Nevertheless, as words can still have different categorizations, i.e. non-mixed-up equivalents, structural ambiguity is not taken out of the grammar.

(b) In CLG, argument lists are flagged for affectedness (*cf.* section 1.6.1). These flags can be seen as indices of the category's head (Van de Woestijne 1999; one could read a category $x\backslash a{\sim}L/u{\sim}R$ as having a head xau, for example, yielding a category $xau\backslash L/R$. By definition however, flags may change as a consequence of cancellation and composition; basically, therefore, one

could claim that the primary head is affected by cancellation and composition. This runs counter to the definition of head in CCG by Joshi *et al.* (1991); here, the head is essentially unaffected by composition. However, the flag on the result category does not constrain the applicability of a rule, but merely serves as an output feature. In order to stay within the CCG framework, we would have to abstract away from flags. On the other hand, flags do not contribute to the generative capacity of the grammar. This follows from the fact that both category names and flag names, hence any combination of them, come from finite sets. Affectedness flags limit the search space for the applicable rules, but do not exclude proper derivations. They are processing features, rather than syntactic markers.

(c) In CLG, *wh*-arguments come with a particular cancellation mode and are treated in a special way. In merging argument lists, they are 'suppressed' in that they always end up at the bottom of the merged list, in order to be left-peripheral when cancelled (*cf.* section 1.6.2.3.4). This special treatment is very restrictive, and affects neither count properties nor the preservation of directionality. The restrictions are the following: an argument list (whether lexical or merged) may contain one *wh*-argument at most and a *wh*-argument occurs only in a left argument list. These restrictions start out from the lexicon, and are maintained under composition. Clearly, the special treatment is context-free: in merging lists, check for the occurrence of *wh*-elements; if none is present, just append; if more than one is present, cancel the operation; if there is exactly one, put it at the bottom of the appended stack. The only effect of this treatment is on the word order: the *wh*-argument is bound to be 'consumed' at the left periphery of the categorial complex that introduced it.
The context-freeness of this special append operation of two lists L and R can be shown by constructing a pushdown automaton, or merely a pushdown transducer, that uses an input tape containing the category symbols of L, followed by those of R, and one stack, on top of which the dislocated element is pushed as soon as the category symbols of L have been written to the output tape, and that is popped as soon as the category symbols of R have been written to the output tape. Pushdown automata are equivalent to context-free grammars.

(d) Furthermore, in the grammar of Dutch certain pronouns must be allowed to be cancelled (deleted from an argument list) without being on top of the stack, *i.e.* without having a fixed position in the order of arguments. These pronouns occur (among other functions, which are not under discussion here) as left arguments or complements of prepositions. They come in three forms: *hier* 'here, this', *daar* 'there, that' and *er* 'there, it' (see also section 1.2).

By necessity, they are a distinct class of *pro-np*s. The rule concerning their cancelling differs from all other rules in grammar, in that it does not match the top of the stack but has to check a complete argument list. Still, processing it is context-free as the argument stack – not to be confused with the stack of a pushdown automaton – is simply a list, and cancelling, that is deletion of a list element, requires only regular operations. Even this dirty feature of Dutch does not unbalance the grammatical set-up, although *er*-pronouns lean heavily on *disharmonious composition*, a notorious composition mode beyond context-freeness, which we need to handle crossing dependencies, among others (*cf.* section 1.4.8). Neither the repression of *wh*-elements discussed above nor the anywhere-cancellation of *er*-type pronouns permutes argument lists.

(e) While expressing the rules of CCG, the syntax of categories is also involved. In CCG's Generalized Composition rules, the full internal structure of the secondary category is taken into account, while in CLG, composition is also sensitive to the full internal structure of the primary category. Shifting part of the load to the primary category does not increase or decrease CLG's generative capacity. In particular, 'opening' the primary category as such does not induce permutation closure, nor does it result in the so-called MIX languages (Cremers 1999).

(f) In CLG, two types of categorial assignments are used that may refer to themselves, to wit the *double type* for *wh*-elements, and the automorphism type for adjuncts. The syntax of a double type, represented by a pair of categories $C^{left} * C^{right}$, could compromise the grammar or its generative capacity when C^{left} and C^{right} are cancelled against each other. However, this is made impossible because of appropriate lexical assignments. The only disadvantage of double types, then, is having to treat one more category per sentential domain. Adding this constant number does not affect CLG's generative capacity. For double types see also section 1.6.2.2.6 and chapter 3.
Automorphism types are types that can combine with categories headed by a category from a subset from V_T to form a new category of that head. Their status is discussed in section 1.6.3. In CLG, this subset is restricted to some predicative and sentential basic types. Procedurally, this definition means that an inert type *automorphism*, when heading the primary category, is replaced by the head of one of the secondary category, while changing/constituting left or right argument lists up to the merge mode level. This is implemented by assigning a unique label to an automorphism in the lexicon and by applying a rewrite rule to the label once, and as soon as the automorphism type is involved in a derivation. The syntax of a rewrite rule takes the same form as

normal CLG rules. By its nature, a rewrite rule does not obey count invariance or reducibility, but is to be regarded as a non-recurring translation step that constructs a suitable instance of a combinatory rule as needed during the derivation process, and can be taken in polynomial time and space. This rewrite procedure prevents the lexicon from being overloaded with completely predictable forests of adjuncts. CLG's generative capacity is not affected by rewrite rules as they do not introduce new structures.

(g) CLG's Generalized Composition rules can be derived from CCG's Generalized Composition rules (199)-(200) as follows. In these rules, the primary and the secondary heads x and y may be complex categories by definition. Substituting $(p|w_1|w_2....|w_m)$ for x in (199) gives (202), taking time and space proportional to the number of categories, i.e. m+1. Thus, (202) is an ordinary rule of CCG, though written down more restrictively, to be derived from (199) in linear time and space. Such a substitution cannot be made to achieve (201), however, because separating the primary category's head and arguments cannot be done in one of CCG's Generalized Composition rules in constant time and space. However, although a category is defined recursively, each lexical category from V_N, and consequently each derived category from $C(V_N)$, has a finite internal complexity. It is easy to see that an algorithm exists splitting the head and arguments of a generalized category $(p|w_1|w_2....|w_m)$, taking time and space that are proportional to the number of arguments. In fact, Prolog's `append/3` predicate (in reverse) is such an algorithm. Specified as a Prolog list, splitting only costs constant time, as splitting a list into its head and tail is a one-step operation. By appending ('chaining') the arguments, which takes time and space proportional to the number of arguments of the primary category, the result category is achieved. Thus, (201) is an extraordinary rule of CCG, to be derived from (199) in no more than linear time and space.

The overall conclusion is that there is a polynomial mapping function from CLG's category format to CCG's more general form. CLG's rule syntax is a more practical, and slightly more liberal reformulation of CCG's rule syntax, and can be derived from it by polynomial means. CLG is maximally as complex as CCG; this means that, if the recognition task for CCG can be done in polynomial time, it can also be done for CLG in polynomial time. Although CLG's generative capacity is a little larger than CCG's, it is practical for the chapters to come to regard CLG as an instance of CCG.

1.8.4 CLG and the Chomsky hierarchy

The first categorial grammars only defined rules for Forward and Backward Application. Bar-Hillel *et al.* (1964) have shown that these grammars have the same *weak generative capacity* as context-free phrase-structure grammars. Lambek (1958) added new deduction rules to these A(jdukiewicz) B(ar-Hillel) grammars, in particular, rules deriving complex categories by hypothetical reasoning: if *a* and *b* go to *c*, *a* goes to *c* under hypothesis *b.* These rules add to the descriptive power of AB grammars, as they allow for more combinatorics, like *harmonic (function) composition* $x/y \;\; y/z \rightarrow x/z$ and $x\backslash y \;\; y\backslash z \rightarrow x\backslash z$, and *type raising* $x \rightarrow y/(y\backslash x)$ and $x \rightarrow y\backslash(y/x)$. Pentus (1993) proved that Lambek grammars are still context-free. Adding 'residuation modalities' to Lambek (1958) grammars does not extend their weak generative capacity beyond context-freeness (Jäger 2003). CCGs generalize the Forward and Backward Application rules by Generalized Forward and Backward Composition rules, respectively (Steedman 1990). This means that CCGs, including CLG, at least recognize the context-free languages. CLG's extended form of Generalized Composition cannot be exploited to recognize the permutation closure of the so-called MIX language $\{a^n b^n c^n : n > 0\}$ (Cremers 1999). The MIX language is more than context-sensitive. Adding modal operators plus structural postulates greatly increases the complexity of Categorial Grammars (Jäger 2003). The crux is what is to be understood as 'structural postulates'. CLG's rules of Generalized Composition can hardly be taken as 'structural', as they can be derived from CCG's rules in polynomial time. The same is true for the finite set of modal operators. Joshi *et al.* (1991) show that CCGs, *e.g.* CLG, are weakly equivalent to Linear Indexed Grammars, which are in the class of the Mildly Context-Sensitive grammars. Linear Indexed Grammars (LIGs) were defined by Gazdar (1985) as a restriction of the Indexed Grammars, introduced by Aho (1968) as a generalization of CFGs. Indexed Grammars may be described as CFGs in which a stack of so-called stack indices is associated with each non-terminal. Rule invocations can be limited by restrictions on the top of the stack. In a LIG, only one daughter non-terminal can inherit its parent's stack, instead of all daughters.

Joshi (1985) defined a class of formal grammars which are only slightly more powerful than CFGs, but which still allow for descriptions of natural languages in a linguistically significant way. This class, dubbed *mildly context-sensitive languages* (MCSL), is described by Joshi (1985) and Joshi *et al.* (1991) to have at least the following properties.

(1) CFLs are properly contained in MCSL;
(2) Languages in MCSL can be parsed in polynomial time;
(3) MCSGs capture only certain kinds of dependencies, such as nested dependencies and certain limited kinds of crossing dependencies;
(4) Languages in MCSL have the constant growth property.

CCGs, including CLG, are an extension of AB grammars, which are weakly equivalent to CFGs, satisfying (1). Polynomial time parsers for CCG do exist; see below. This satisfies (2). Regarding (3), Joshi *et al.* (1991) give the example of "subordinate clause constructions in Dutch or some variations on them, but perhaps not the so-called MIX (or Bach) language, which consists of equal numbers of *a*'s, *b*'s, and *c*'s in any order". CLG was designed to capture the Dutch verb cluster in subordinate clauses, including crossing dependencies. Obviously, CLG also handles various instances of nesting dependencies. As was mentioned above, CLG is not able to generate the MIX language. CLG, then, may be said to comply with (3). The constant growth property requires that the length of the sentences generated and put in order by the grammar grows linearly. This is certainly not the case for $\{a^{2^n} \mid n \geq 0\}$. In CLG there is no substantial numerical condition, like the sentence's length, on the generated structures. Look-ahead is restricted to one neighbour's category only, and by inspecting the first element of the list of leftward arguments and the first element of the list of right searching arguments, including the *wh*-position, at the same time. CLG, then, complies with (4). In conclusion we can say that the language generated by CLG is not inconsistent with MCSL.

The positioning of Categorial Grammar in the Chomsky hierarchy is not obvious. In the spirit of Van Benthem (1993), Jäger (2003: 106) states "that there is tight connection between interaction postulates and generative capacity. This may eventually lead to a taxonomy of languages that is much more fine-grained than the traditional Chomsky hierarchy." MCSL is an excellent candidate for an in-between class of 'reasonable' grammars that are linguistically relevant and semantically potent. If CLG lives in MCSL, it is in good company.

1.8.5 Parsing CCGs

For any computational problem, we are interested in algorithms that can solve the problem. We want to know how efficiently these algorithms operate in terms of running time, memory usage, or other computational resources. Most of the time, by 'most efficient' we mean 'fastest', as time considerations

are often the most important in practice. Time complexity relates the execution time of an algorithm to the size of its input. Here, the problem is deciding whether a string (a sentence) is in the language generated or recognized by a grammar of some type. A parser is an algorithm to solve this problem. For a CFL and an MCSL the problem is known to be polynomially solvable. That is: for any arbitrary input sentence, *i.e.* in the *worst case*, the largest amount of time needed by the parser to decide whether the sentence is in CFL or in MCSL is a polynomial function of the size of the input, *n*. Such a parser is called *efficient*. When the major term of the polynomial function is n^3, ignoring constants and lower orders of magnitude, the parser is said to run in cubic time, or, to put it formally: to have $O(n^3)$ time complexity. The 'Big O' notation is useful when analyzing and comparing the efficiency (or: scalability) of algorithms – especially when the size of the input becomes large – independent of properties of actual hardware. The key to efficiency is for a parser to be more rigid than the grammar. Where a grammar defines the allowed structures in a language, a parser blocks the redundant, albeit grammatical ones (Eisner 1996).

Above, it was shown that there is a relationship between CLG and CCG, which can be established in polynomial time and space. For practical purposes, we regard CLG to be an instance of CCG. A well-known parsing algorithm for CCGs, including CLG, and displaying polynomial time complexity is that of Vijay-Shanker and Weir (1990). It is based on a well-known parsing technique for context-free grammars, namely the chart parsing technique (Kay 1973, 1980), and runs in n^6 time.

Chart parsing algorithms are based on dynamic programming. This technique systematically fills in a table of solutions (or, more general: *items*) to sub-problems. When the table is complete, and spurious ambiguities have been removed (see below), it contains not only the solution to each sub-problem, but also to the problem as a whole. Because of this property, the way chart parsers handle alternative solutions is dubbed *parallel*, as opposed to *backtracking*, the latter keeping only one solution at a time in memory. Tabulation is necessary to face the inherent ambiguity of natural language. Chart parsing algorithms may differ as to the type of items they use (their data structure), and the processing order (which follows from the algorithm's control structure).

When parsing of a grammar *G* is the problem, the table is a chart (or: well-formed substring table). A chart is a two-dimensional array that relates *spans* (substrings, denoted by starting and ending position) to categories (or, more generally, constituents encoded by sub-trees). A sentence has a parse, *i.e.* it is

in $L(G)$, if the chart has an entry that relates the substring starting at position 1 and ending at position n to some designated symbol, for example s. For CFGs, a parser does not have to know how a constituent is constructed, but only that it can be constructed. As a consequence, there are Cn^2 different constituents possible to be constructed for a sentence of length n, where C is the number of different categories in the grammar. A constituent can be constructed in multiple ways. There are $O(n)$ different ways of building a constituent that is $O(n)$ in size. Building Cn^2 constituents in $O(n)$ ways takes $O(n^3)$ time, assuming adjacency of constituents. Adjacency is guaranteed by categorial combinatorics. Chart parsers, then, run in cubic time. Generated constituents that cannot be reached from a valid designated symbol, like s, are purged from the chart. A sub-tree is never computed more than once, and never stored more than once, so that it can be shared among parses, which explains the efficiency of chart parsers. However, as the chart is a compact data-structure, called a *shared forest*, containing analyses to sub-sentences, the analysis for the whole sentence must be constructed by putting all pieces together by travelling systematically through the chart. To retrieve the first analysis, this procedure takes polynomial time; to find all analyses, however, takes exponential time.

A chart, being a data-structure, is neutral with respect to parsing and search strategy. The same is true for CFG rules; they may be said to be declarative statements. In categorial grammar, however, the categories encode partial derivation trees. It is easy to see them as the items of a chart parser. From a parser's point of view, a category with an adjacent neighbour category directly triggers the calling of an appropriate reduction rule. In turn, the resulting category, together with another adjacent neighbour category, fires a reduction rule that fits their requirements, etc. until the designated symbol, *e.g. s*, is reached after a finite number of reductions. In other words: categorial grammar rules are one-way traffic, that is, directional. They are procedural. A bottom-up approach (or: data-driven search) is the natural control structure for a parsing algorithm for lexicalized grammars, like categorial grammar.

A top-down approach (or: hypothesis-driven search) for parsing our type of categorial grammar would be rather artificial. Given a hypothesis about a phrase, it is very inefficient to guess two categories that constitute the hypothesis, as categories can be fairly complex (recursive). This also means that a top-down grammar filter does not make sense. One might add statistics – *e.g.* extracted from treebanks – so as to determine a realistic hypothesis about a phrase, consisting of two possibly complex daughter categories. However, applying probabilistic means *to steer the derivation* is contra the rigidity of

our *lexicalized* syntax (cf. section 1.3): it might cut off possible analyses on non-grammatical grounds. Applying probabilistic means for selecting analyses post-deriviationally is unproblematic. Extensions to CLG benefit from a bottom-up parsing regime. The algorithm for removing spurious ambiguity operates in a bottom-up fashion. Regarding robustness, it is pointless to make any top-down hypotheses for partial derivation structures. A semantic filter will gain maximum efficiency only when it can be applied to fully specified logical forms, as early as possible and at any derivational level. Semantic structures that are created in a top-down fashion may be incomplete and therefore be of less use for filtering or constraining analyses. Top-down procedures that cannot be applied in parallel to a bottom-up parser for CLG, such as determining scopal dependencies (cf. chapter 2 on semantics), can only be applied post-derivationally, that is at the moment that all words of the input sentence have been processed and all data structures have been computed.

One of the main disadvantages of a bottom-up parsing strategy is that structures (sub-trees) are created that will never lead to the designated top symbol. As a top-down grammar filter cannot be implemented efficiently while filling the chart, because of the recursive nature of categories, nodes that are not descendants of a valid top node are filtered from the completed chart. While constructing the parse tree(s), disambiguation takes place by deploying a bottom-up semantic filter (see below). Empty categories – being another problem for bottom-up parsers – do not occur in CLG.

In addition to a bottom-up control structure, the parsing strategy needs a way to consume the input symbols. A straightforward implementation is a left-to-right approach, building one-word constituents first, starting at position *i-1,* and ending at position *i*, for *i*=1..*n*, then proceeding to two-word constituents, starting at position *i*-2, and ending at position *i*, etc. while each level builds on the results of previous levels, that is, building constituents from shorter to longer ones in a strictly bottom-up manner. Finally, a constituent is built from position 1 up to *n*. A left-to-right bottom-up processing order ensures that all analyses of shorter ranges are ready before reductions of longer ranges are tried.

Vijay-Shanker and Weir's (1990) algorithm for parsing CCGs is based on the Cocke-Kasami-Younger (CKY) algorithm (Kasami 1965, Younger 1967), which is an example of a bottom-up chart parser. The CKY algorithm requires the grammar to be in Chomsky normal form: grammar rules are of the form $A \rightarrow a$ with A a non-terminal and *a* a terminal symbol, or $A \rightarrow A_1\ A_2$, with A, A_1 and A_2 non-terminals.

1.8.6 Parsing CLG

The grammar rules in CCG, as defined by Joshi *et al.* (1991) and including CLG, can be said to be in Chomsky normal form, as they take the format of binary rewrite rules. This means that standard CFG parsing algorithms, including the standard CKY chart algorithm, but even a non-deterministic shift-reduce parser, will suffice as a parser for CCGs. Stated otherwise: standard CFG parsing algorithms, including chart parsers, are 'insensitive' to CCGs as defined by Joshi *et al.* (1991), including CLG. Obeying count invariance, directionality and adjacency (cf. section 1.4), CLG's rules can be processed locally. In general, a rule's format does not determine its generative capacity. For example, the disharmonious composition rule has a context-free rule format, but generates mildly context-sensitive structures. This is possible due to the fact that the two categories in a CCG rule entertain a relationship that can be defined by *any* function. Categories in a CFG rule are atomic and functionally independent. The categories in a CCG are *indexed*, as in Linear Indexed Grammars (cf. section 1.8.3). They are data-structures, or *complex symbols.* The locality of the context-free rule format compensates for the richness of information compiled in these data-structures. The combination of locality and rich information is reminiscent of the deterministic program of Marcus (1980), which merges grammatical specification and control.

CLG – like any grammar for natural language – includes large context-free sub-parts. In CLG, these sub-parts follow from applying only the basic case of the Generalized Composition rules, namely Application. This is enforced by using only some designated flags (to wit: /^*isl* for saturated constituents, /^*penins* for near islands, /^*sentop* for *wh*-islands, and \^*part* for particles) which require the secondary category's arguments list to be empty (cf. section 1.6.1). They reduce the grammar to a standard AB grammar, which has been proven to be context-free.

In general, however, CLG's generative capacity is beyond context-freeness. This means that there are families of sentences that a standard CKY algorithm will not parse in cubic time, but in exponential time instead. Vijay-Shanker and Weir (1990) explain this behaviour by noting that categories spanning part of the input can grow proportionally to the input size, and, as a consequence, be exponential in number. In CLG, this can happen only to categories occurring in the final verb cluster with crossing dependencies in Dutch, and which can have arbitrarily long sequences of arguments. However, these arguments can only be of type *np*. This implies that the number of possibilities for argu-

ment lists is bounded for each value of the list length, and does not exceed a boundary that is polynominal in the sentence length. By taking advantage of the fact that regardless of the length of a category only a bounded amount of information (its head and its innermost argument) is necessary for determining which rule to apply, Vijay-Shanker and Weir (1990) turn the exponential worst-case behaviour of the standard CKY algorithm on CCGs into polynomial behaviour, namely $O(n^6)$. Their chart-parsing algorithm is made sensitive to the characteristics of CCG. Komagata (1997) concludes that their method only removes a possibility that rarely occurs in realistic grammars. Experiments with a standard CKY algorithm on CLG for a typical set of Dutch sentences display an average-case behaviour of $O(n^3)$. The same behaviour is reported for experimental parsers for realistic fragments of English and Japanese (Komagata 1997, 1999). Parsing natural language, then, is just one of the problems that have bad worst-case performance, but good average-case performance.

1.8.7 Extending and restricting a parser for CLG

Natural language is ambiguous. A parsing algorithm for natural language will compute different structural descriptions for the same sentence. This is called structural ambiguity. In Lambekian categorial grammar, this ambiguity is even more apparent due to *type raising* $x \rightarrow y\backslash(y/x)$, an immediate consequence of the rule of slash introduction, and associativity of composition. For instance, *John likes Mary* might be parsed as $[_S$ *[John likes] Mary]* or as $[_S$ *John [likes Mary]]*, applying $(s/np)\backslash np$ and $(s\backslash np)/np$ for *likes*, respectively, which yields different parses for the *same* sentence with the *same* meaning. CCG's flexible notion of constituency allows for the structure *[John likes]*, which indeed must not be blocked as can be seen in coordinated sentences, like *John likes, but Harry hates Mary*. This type of ambiguity has its source in the grammar, and not in the language. Hence, it is called *spurious ambiguity* (Wittenburg 1986). On the other hand, rule-based parsers, confronted with a grammar or lexicon that is not *sound* (soundness: all word-forms are correctly defined by lexical entries) and not *complete* (completeness: the lexical entries are the only definitions of the word-forms) may not produce any analysis at all for some correct sentences. This asks for robustness.

1.8.7.1 Removing spurious ambiguity

Spurious ambiguity in Lambekian categorial grammar is caused by the property of *structural completeness*: when a sentence is grammatical, it can be derived under all binary bracketings (Moortgat 1988). This gives rise to an

exponential number of possible analyses. Approaches for eliminating spurious ambiguity can be distinguished in syntactically and semantically oriented methods. The syntactic method of Eisner (1996) only allows for normal-form parses. "A parse tree is in normal form, when the result of every rightward (leftward) [Generalized] Composition rule is not composed (or applied) over anything to *its* right (left). ... Thus, every functor will get its arguments, if possible, before it becomes an argument itself." Lexical categories are preferred to derived ones, and application is preferred to composition. This method will block the structure $[_{s/np}$ *John likes]* in *John likes Mary*: it is created by rightward composition and cannot serve as an argument of a functor to its right afterwards. This very structure will be allowed, however, in deriving *whom John likes:* it can serve as an argument of a functor to its left. Although Eisner (1996) specifies the restrictions for strictly rightward and leftward operating rules, he notes that disharmonious ('mixed') composition rules are not harmed. He proves that *all* spurious ambiguity originates in associative forward (backward) "chains", like A/B/C C/D D/E/F/G G/H, and that normal-form restrictions in fact only allow right-branching derivation trees. Any CCG parser can be easily adapted to produce only normal-form parses by marking the result of each rule invocation, and using the markings in the next rule invocation, *i.e.* by adding tags to the grammar rules. With the Dutch verb cluster – which is an instance of a backward chain – in mind (cf. section 1.2) it turns out that Eisner's tags are a bit too coarse for implementation in CLG. Instead, CLG makes use of carefully designed sets of modes and flags for a fine-grained control of the combinatorics, entertaining syntactically rigid derivations. For example, it distinguishes between islands, near-islands, and *wh*-islands. In principle, the grammar rules with modes and flags attached to them produce unique analyses to an arbitrary string of categories. This follows from the non-associativity of the CLG calculus. Thus, CLG avoids spurious ambiguity at the cost of giving up direct composition of fully specified semantics (but see the next paragraph on semantic methods for eliminating spurious ambiguity). However, in order to define CLG in a more principled way, some grammar rules are underspecified, which leaves some room for spurious ambiguity. A typical example is the attachment of adjuncts. Many cases can be removed at a local level by an adapted version of an algorithm by Hepple and Morrill (1989) that was developed in Vijay-Shanker and Weir (1990). Operating on a full chart, it inspects each triple of adjacent constituents and checks whether they have been combined in two different ways with the same result category. The only possibilities are a left-branching and a right-branching structure. We decided the right-branching parse to be the 'normal form' (cf. Kayne 1994) and removed the left-branching derivation tree (not the categories themselves).

By applying this method bottom-up and recursively, all and only the spurious ambiguity will be removed from the chart, according to Vijay-Shanker and Weir (1990). Their method is defined for spurious ambiguity that arises from the associativity of (generalized harmonious) composition. Since nothing in their method hinges on the *type* of composition, spurious ambiguity that might arise from CLG's *disharmonious composition* is removed as well. In conclusion, all possible sources of spurious ambiguity in CLG will have been removed before the chart is travelled in order to construct the parse trees.

Another main stream of methods that eliminate spurious ambiguity is semantically oriented. This approach, *e.g.* Karttunen (1986), eliminates a constituent when its logical form is equal to or subsumes a logical form that has already been derived and stored in the chart. This check might require exponential time. Note that the concept of a chart – storing computed structures only once – is exploited here for syntactic as well as for semantic descriptions. In general, if two interpretations share the same syntactic structure, it may not suffice to store this structure in the chart only once, because its semantic structure may require different variable bindings for different interpretations. Although sharing of semantic substructures is certainly possible, the concept of a chart is most efficiently exploited for syntactic structure. In CLG then, the chart is used only for syntactic purposes. After the chart has been filled and all spurious ambiguity has been removed, one or more parse trees are constructed from the chart. Semantic readings are constructed compositionally. These readings start out in the lexicon as underspecified Stored Logical Form frames which are unified in parallel to each derivation step. An SLF frame is a pre-derivational specification of the functor/argument relationships, which as such does not change during derivation. Derivationally, they will get more and more instantiated. A chart that has been freed from spurious ambiguity will only give rise to possibly plural but definitely unique logical forms. Where Eisner (1996) tries to keep only those syntactic parts together that will enable exactly one parse in each semantic equivalence class (by blocking non-normal form parses in the grammar), the CLG framework already assigns the semantic contours in the lexicon. Clearly, rigid grammar, avoiding spurious ambiguity, can only come with underspecified compositional semantics. (See also chapter 2 on semantics.)

1.8.7.2 Adding robustness

As for robustness, grammatical and lexical deficiencies need to be remedied. A chart parser that cannot assign a complete parse tree for an input sentence might assign sub-trees to parts of the input sentence. This only makes sense

when it is done in a bottom-up fashion. A *cover-over-a-chart* or *cover* for short is a sequence of sub-analyses to sub-parts of the input, together spanning the whole sentence. A cover consisting of one sub-analysis only, headed by a single sentential type, corresponds to the traditional notion of a parse tree. The definition of a chart parser guarantees that all possible sub-analyses are found and stored in the chart. Thus, the notion 'cover' does not apply to non-parallel parsing methods, such as a shift-reduce (backtracking) parser. Finding the best possible cover is a non-deterministic process, as it is an instance of an optimization problem. Possible evaluation criteria include: to what extent is the cover fragmented (the number of sub-analyses), to what extent are right-branching structures to be preferred over left-branching structures, and to what extent are the sub-analyses' top-nodes sentential and saturated. Of course, the application of these criteria depends on the type of information gathered in the parsing process. Currently, the DELILAH parser selects best parses primarily on the basis of fragmentation and secondarily, by inspecting the complexity of the fragments' logical form (see also chapter 2). But the available information is so extensive that many other selection procedures could be implemented too. Since robustness basically corrects grammatical deficiencies, fragmentation of the parse may also be repaired by relaxing those grammatical restrictions that resisted composition of combinatory categories or unification of complex symbols. These are three more or less independent sources of fragmentation:

- two adjacent fragments strand by lack of appropriate categorial typing – this results from underspecification;
- two categories resist composition because of not complying with the actual mode of composition (cf. section 1.4.4, 1.6.2 and 1.7.3) – this is due to overspecification;
- two complex symbols cannot be unified because of inconvergence of specifications – this too is due to overspecification.

Fragmentation resulting from underspecification – basically: the lack of appropriate combinatorial material – can only be repaired by adding new combinatory options, by means of a categorial hypothesis. Because the CLG operates a rigid syntax, inspection of a fragmented sequence may invoke an 'educated guess' on one or more combinatorial options that would have resolved the fragmentation. This is the general idea:

(206) *If* < C_1, ...C_j ... C_n > is a sequence of combinatory categories *and*
$C_j \rightarrow \alpha \backslash [C_{j-1}...C_1] / [C_{j+1}...C_n]$, *then* C_1... C_j....$C_n \rightarrow \alpha$

The projected category is a hypothetical extension of the specification of the phrase with category C_j. Projecting this type of hypothesis is to be seen as a learning tool. It is less suited, however, to solve on-line parsing problems.
As for fragmentation resulting from overspecification: DELILAH's modalized grammar offers the opportunity to 'neutralize' modes of composition post-derivationally and see what happens to a cover when modal combinatorial restrictions are lifted. That is: two fragments may be composed according to generalized non-modal composition rules of the following type, where the merges of argument lists are chosen so that discontinuity effects are minimalized (cf. section 1.7.2):

(207) `a\List1/[b^nonmode|List2]  b\List3/List4 →`
`a\List1+List3/List4+List2`

If this relaxation leads to unification of the associated complex symbols and to a non-fragmented interpretation, this composition seems to be acceptable: it results from basically sound combinatorics. Note, moreover, that adding this post-derivational relaxation does not introduce an *everything goes* derivation – a form of robustness that would violate the semantic regime, which lives on the idea that being meaningful is not an accident but the outcome of structural subtlety.

Concerning robustness at the lexical level, information in the lexicon may be absent, incomplete or simply wrong. A simple, but rough approximation is to regard missing information to be a *name* (of type *np*). In categorial grammar, however, there are better options, since categories are not absolute, but relative encodings of the grammar's agenda. The arguments of a category refer to one or more neighbours to the left and/or to the right. When in an input sentence exactly one word is missing in the lexicon, an estimate of its category – modulo adjunctive types – can be derived from the other categories in the sentence by applying some count protocol (cf. Van Benthem 1986, Cremers and Hijzelendoorn 1997a). Such a protocol determines supply and demand of heads and arguments in a string of categories, and applies to CLG. Moreover, because of CLG's typical rigidity, the estimate is even deterministic, given some hypothesis as to the string's ultimate goal. The strategy applied here basically follows from 'inverting' the operation of reduction rules. This is possible for CLG, as will be demonstrated in section 1.9.1, where CLG's recognizing grammar is converted into a generating grammar.

This means that there are different *learning strategies* possible for the parser to construct or reconstruct a missing type. The options include a hypothe-

sis for the result type, *e.g. np*, the direction(s) in which missing categories are searched for, and the number of them in each direction. Strategies may depend on the position of the missing category in the sentence, and do not depend on the parsing algorithm, but operate on the level of the pre-terminal analysis of the input sentence. The matter is pursued in Van 't Veer (2007).

1.8.7.3 Semantic filter

In addition to the filter that eliminates spurious ambiguity, a semantic filter has been designed that ranks semantic readings post-derivationally. All logical forms that arise from the parse trees in the chart are valid and different. They differ mainly as to their 'lexical feed': the way the lexicon has delivered building blocks to the semantic combinatorics, and therefore the level to which lexical aggregation – semantic collocates, extended lexical units, constructions – is reflected. Clearly, a logical form accounting for lexical aggregation is in some sense more adequate than a logical form that does not. Sentence (208) is more sophistically represented by logical form (209) than by (210), though the latter can hardly be seen as a wrong interpretation.

(208) John kicked the bucket
(209) died'(john')
(210) ιx. bucket'(x) & kicked(j,x)

In Delilah, different readings are ranked by evaluating the semantic complexity of their Stored Logical Form at top level, *i.e.* at the level of the whole sentence. In successful analyses, small semantic structures that correspond to large parts are favoured above large semantic structures corresponding to small parts. This distinction, however, makes sense only when comparing viable interpretations; therefore, semantic complexity is a property of the analysis as a whole, and not of its parts. The semantic complexity is calculated by measuring certain properties of an SLF, including depth, items per store and number of the (resolved) semantic variables. Since all semantic structure stems from the lexicon, the degree of involvedness of the top-level SLF reflects the lexical semantics of the analysis. The ranking of readings resulting from this measurement can be used to select readings.
There is no canonical way to measure the complexity. The first implementation of this type of post-derivational reading selection was sketched in Cremers and Reckman (2008). There, the metrics were calibrated by human judgements on the outcome of the ranking. Other approaches may be equally valid. The important point is that SLF makes this computation feasible (see chapter 2) as part of the parsing process.

1.8.8 Parsing coordination

In section 1.7.4, it was argued that coordinative structures are handled extragrammatically by a dedicated algorithm, as the coordinates as such are not syntactically marked. Such an algorithm was developed in Cremers (1993a), and implemented in a backtrack parser according to Cremers and Hijzelendoorn (1997a). Fannee (2006) converted the algorithm into a tier of the chart parser. In both implementations, parsing a coordinated sentence consists of

(a) marking the boundaries of the left and right coordinates by finding a conjunction symbol; it has a combinatorially inert category, which is not taken into account by the parser
(b) parsing the left and right coordinates separately up to some 'fully reduced' level, which is slightly different, but comparable to an analysis of a cover, revealing that the coordination algorithm in fact implements a robustness strategy on incomplete phrases
(c) finding and matching levels of corresponding linguistic structures by applying a specialized, deterministic decomposition algorithm to the level of analysis pointed at by (b)
(d) constructing a top level analysis for each conjunct, *e.g.* a sentential analysis.

In abstracto, here is what the algorithm does. Given a sequence of categories, one of which is a conjunctive, with hypothetical reduction to α

(211) $[_{\alpha}$ c1 c2 c3 & c5 c6 c7]

the algorithm determines, by inspecting the nature of the categories at both sides of the conjunction, which category has to be decomposed in order to reveal a coordinate and which other categories form a coordinate, *e.g.*

(212) $[_{\alpha}$ c1 c2 $[_{c3}$ c4 *c5*] & *c5* c6 c7]

and resolves all elliptical structure to the left and the right of the conjunction, yielding

(213) $[_{\alpha}$ c1 c2 c4 c5 c6 c7] & $[_{\alpha}$ c1 c2 c4 c5 c6 c7].

Subsequently, the parser pursues the hypothesis that both sequences <c1, ..., c7> actually reduce to α according to normal syntactical and semantical combinatorics. More concretely, it computes the well-formedness and interpretability of the structure

(214) $[_{s}\ [_{c1}$ John$]\ [_{c3}\ [_{c4}$ loves$]\ [_{c5}$ his dog$]]\ [_{\&}$ and$]\ [_{c5}$ her puppies$]\ [_{c6}$ a lot$]]$

by transforming it into

(215) $[_{s}\ [_{c1}$ John$]\ [_{c4}$ loves$]\ [_{c5}$ his dog$]]\ [_{\&}$ and$]\ [_{s}\ [_{c1}$ John$]\ [_{c4}$ loves$]\ [_{c5}$ her puppies$]\ [_{c6}$ a lot$]]$

in order to test whether the sequences of categories to the left and the right of *and* reduce properly to S.

Determining the category in the span of which an outer border of a coordinate is 'hidden' – c3 in the examples (211) and (214) – is the algorithm's major task and achievement. The algorithm subsists on CLG's rigidity: instead of pumping up – in a particular context – the conjunction to some hypothetical category with structure $\beta\backslash\beta/\beta$ and testing that hypothesis on the present partial analyses, it sticks with the normal categorial reductions, and compensates for the lack of categorial flexibility with deterministic inspection of the reductions' result. Cremers (1993a) argues that this strategy is sound and complete in the following sense:

(a) it deals with both constituent and non-constituent coordination
(b) keeping the conjunction combinatorially inert and promoting it to a position 'outside of the brackets' respects the semantic integrity of the original phrasal conjunction.

Currently, the algorithm is defined and implemented for binary conjunctive structures only. Even this restriction is proven to complicate the parsing process for coordinations when compared to the parsing of non-coordinative sentences by significantly enlarging search space (Cremers and Hijzelendoorn 1997a) – the procedure by definition operates on partial analyses. This complication is inevitable, given the inherent syntactic underspecification of coordination discussed in section 1.7.4. There can be no doubt that allowing for multiple coordinations will again increase the complexity of the parsing task. This increase, due to partiality, is bounded only by the fact that the algorithm itself does not add to the complexity of the parsing process, as it works in polynomial time for each cover (Fannee 2006). The rise of complexity follows from the nature of coordination, not from the application of the parsing algorithm.

The core of the algorithm will suit the parsing of other coordinative phenomena, like discontinuous ellipsis. We expect this to be the case because the algorithm is essentially a repair mechanism, a 'late' and high-level implementation of grammatical knowledge, applied to solid syntactic information. All phenomena that cannot be detected or unveiled by normal combinatorics for composition of adjacent phrases can be delegated to this type of repair strategies.

1.9 GENERATING BY SYNTAX: AGENDAS AND LINEARIZATION

There are many good reasons for modeling language generation. Most of them point to the potential usefulness of generating natural-language systems for information retrieval and disclosure of any kind, as is widely discussed in Reiter and Dale (1997) and the many workshops on natural-language generation nowadays. In this section, we will concentrate, however, on modelling generation as the ultimate test for the adequacy of a computational grammar: your grammar is as good as the sentences produced by it. The question one can ask a grammarian, then, is: did you construct your grammar so that it is explicit and precise enough to generate well-formed and interpretable sentences without having resort to pre-established frames, scenarios or templates? If you want to know whether a grammar leaks, and where, build a free generator for it and evaluate its yield. This is the issue in this section. In the chapter on semantics, it will be argued, in addition, that a generator needs free generating capacity to generate from logical form, since logical representations are bound to be underspecified with respect to the sentences underlying them.

1.9.1 Two-directional grammar

From a computational perspective, the main difference between parsing natural language and generating natural language is linearization. When you parse a sentence, the order of words is input; when you generate, the order of words is output.

Prolog's definite clause grammars (dcgs) are famous for being applicable in two directions: the same set of rules can be used for parsing as well as for generating (Pereira and Warren 1983). As a matter of fact, the difference between parsing and generation evaporates in such a formalism. Although DELILAH is not founded on dcg, it is instructive to study the limitations of dcg. The reversibility of dcgs hinges on a tight connection between linearity and constituency: to be a constituent in a rule is to be well-ordered and continuous in a sentence. Literally between brackets, the constituents can be related to each other in every expressible manner – including *e.g.* semantic subsumption – but the linear ordering of a phrase cannot be manipulated. Fixing linear order is the backbone of reversibility in dcg. A slightly weaker but very insightful conclusion in this vein is drawn by Van Noord (1993:62) for reversible grammars in general. Moreover, it must be noted that any indeterminism in a dcg, *e.g.* caused by alternative expansions of rules, leads to irreversibility of the grammar in the sense that certain structures – those covered by rules lower in the

parsing hierarchy than their alternatives – cannot be generated without redefining Prolog's built-in depth-first strategy for unification and recursion. Consequently, every discontinuity in dcg has to be spelled out as a sequence of well-defined and well-ordered constituents. No essential variables can be used to cover a variety of strings: dcgs operate by recursive instantiation of strings, and string variables in the rules cannot be instantiated. Of course, this makes the whole complex of non-constituent coordination indefinable as a 'normal' rule in dcg. Since we do not deal with coordination in the combinatory grammar anyway, this does not have to prevent us from using reversible dcg as the combinatory engine. There is, however, another form of discontinuity in Dutch that transgresses the one-to-one correspondence between constituency and linear ordering. The problem is adverbial adjunction, again (cf. section 1.6.3). Consider the following sentence. The adverbial operator *waarschijnlijk* 'probably' can occur in each position marked by a dot.

(216) Toen heeft . een linguist uit Ter Apel . een boek van Chomsky . op internet . te koop aangeboden
then has . a linguist from Ter Apel. a book by Chomsky . on internet. for sale offered
'Then a linguist from Ter Apel offered a book by Chomsky for sale on internet'

In all cases of *waarschijnlijk* 'insertion', the adjunct must be related to a propositional domain – it is a sentential modifier – categorially headed by the finite auxiliary. At the same time, however, its scope with respect to other operators like the indefinite quantifiers has to be established (cf. Diesing and Jelinek 1995, De Hoop 1992). In a dcg, this would mean that for each possible position of *waarschijnlijk* in this sentence, its left and its right environment must be specified at constituent level in a separate rule of grammar. Although this would probably not push the grammar of Dutch completely outside the realm of finiteness, we clearly miss a generalization if every *vp*-design introduces a set of dcg rules to account for its internal distribution of adjunctive operators.

When one takes semantic reconstruction seriously, reversible grammar like dcg is not the right – nor most enlightening, most transparent, or most economic – instrument to tackle recursive discontinuity, the essence of syntax: discontinuity arising in structures embedded in discontinuous structures, like movement out of an extraposed or otherwise dislocated constituent. DELILAH does not entertain reversible grammar but operates two related, interdependent, yet distinct syntaxes for parsing and generation. The main difference between the two grammars is that generation must account for composition of segments that may turn out not to be adjacent in the final string. To put it directly: the rule that induces the unification of a verbal structure

and one of its arguments must result in a linearization structure that leaves room for an intervening verbal adjunct. That is: the resulting structure must acknowledge a well-ordered string segment in which an adjunct (or another intervener with a suitable category) can reside – whether that intervention actually 'occurs' or not. The generating rule format has to keep track of *possible* or 'future' linearizations.

The rule format for parsing has been discussed above, and mainly dealt with merging stacks of arguments under composition. Empty stacks – unless specified otherwise – behave as empty lists under append. The rule format for generation deals with segments of strings, since linearization has to be arranged. A category in a generation rule is a quadruple of strings *<WhString, LeftString, HeadString, RightString>.* The role of each (sub)string is indicated in its name. A generation rule is associated with a mode according to which two quadruples are compiled into a new one. The basic operation is a reordering of (sub) strings. To compare the formats, we repeat the parsing rule for mode */^open* (see section 1.6.2.2.3) here, and compare it to its generating or linearizing counterpart.

(217) PRIM \ u~PLA / _~[SEC^*open*|PRA] $\otimes_{open}$ SEC \ _~SLA/ _~[ ] →
PRIM \ LF~SLA+PLA / a~PRA

(218) <WHP,LEFTP, HEADP,RIGHTP> $+_{open}$ <WHS, LEFTS1+LEFTS2, HEADS, RIGHTS> →
<WHP+WHS, LEFTP+LEFTS1, HEADP, RIGHTP+LEFTS2+HEADS+RIGHTS>

In the generating rule, + stands for string concatenation. The generating rule is invoked by the primary category, according to an agenda (see section 1.9.2). At the moment of composition, the primary category must claim four fields: one for its *wh*-arguments, one for its other left arguments, one for its head, and one for its right-hand-side arguments; these form the quadruple of the primary category in the generating rule (218). The same holds for the secondary category, addressed by the agenda. At the moment of merge, the claims on the fields are just that: the final structure of each category and the saturation of its arguments may be unknown. The */^open* merge mode leaves the possibility that some left arguments of the secondary category are saturated before this particular merge, while others are absorbed after the merge. Therefore, the generating rule has to introduce a split of LeftS, LeftS1 and LeftS2; these lists occur at different positions in the final line-up, the line-up corresponding with the resulting category in (218).
The secondary head string *HeadS* has been integrated in the right field of the resulting quadruple. This is the counterpart in the concatenation algebra to

being cancelled in the algebra of types. One can lose types by cancellation, but one cannot get rid of strings combinatorially – the grammar of natural language is resource-sensitive. For that reason, the leftmost string of the resulting category must account for the *wh*-deliveries of both the primary and the secondary category, although at least one of them must end up empty. Which one, is to be determined by the generating \^*wh*-mode.
In the same vein, the generating rules cannot control the derivation, like requiring lists – string-fill recipes – to be empty or emptied, as is done in (217). Instead, (218) positions the secondary right field in such a way that it being claimed cannot influence the ordering of the other strings: it is peripheral in the resulting category.

When one does not use a reversible grammar as such, the question arises how to travel from one grammar – *e.g.* the parsing one – to the other. Hitherto, we have not implemented a general translation from parsing rules into generating rules, but constructed the generating grammar by hand out of the parsing grammar. For the sake of the computability of Dutch, however, it would be rewarding to have an algorithm performing such a translation. In order to achieve such an algorithm, we first have to formulate a very general relationship between categories and concatenated strings *instantiating* the category:

(219) C ::= w (*C is instantiated by w*) iff w is a string of category C;
[C^1, ..., C^n] ::= w^1+...+w^n *iff* C^i:=w^i and the list of categories is a left argument list;
[C^1, ..., C^n] ::= w^n+...+w^i *iff* C^i:=w^i and the list of categories is a right argument list;
[ ] ::= ε, where ε is the empty string and for all strings *w*, *w+ε = ε+w = w*;
H_~La / _~Ra ::= w^{Wh}+ $w^{LA\text{-}}$+w^H+w^{RA}, where for some *XP^wh* in La, XP^wh ::= w^{Wh}, and $w^{LA\text{-}}$ instantiates the remainder of *LA.* The string w^{Wh}+ $w^{LA\text{-}}$+w^H+w^{RA} is represented above and below as a quadruple <*WhString, LeftString, HeadString, RightString*>, with appropriate re-labeling.

In this instantiation, a category is associated with a quadruple of strings w^{Wh}+w^{LA}+ w^H+w^{RA}. Only the string instantiating the head w^H is real, representing the phonological phrase to which that category is assigned. The other three strings are virtual, in the very same sense as the types in the argument list occur negatively until cancellation. Every category constructs a linearly ordered space of strings. Given this mapping from (lists of) categories to (concatenation of) strings, we can conjecture the declarative outline of a composition-to-concatenation translation for, *e.g.,* the rightward cancelling sort of rule, like (217). It must be noted, though, that CLG rules give rise to some degree of non-determinism, allocated in the input conditions of the cancellation modes, the flags (cf. section 1.6). Since flags are declared only at negative

occurrences of types (*i.e.* in argument lists) and not on heads, a certain context may comply with more than one cancellation mode – one cannot exclude this possibility. For that reason, the generating, concatenative rules must follow the pattern of the cancellation modes. With this starting point, here is a translation procedure for the class of rightward cancelling rules; the case for leftward rules is comparable. The result of the translation is a purely concatenative grammar.

(220) *Translation of recognizing grammar into producing grammar*

Given a rightward cancelling rule

PRIM \ Flag1~PLA / Flag2~[SEC^*mode*|PRA] $\otimes_{mode}$
SEC \ Flag3~SLA/ Flag4~SRA $\rightarrow$
PRIM \ Flag13~SLA$\oplus_{mode}$PLA / Flag24~PRA$\oplus_{mode}$SRA

create a generating concatenative rule
<WHP, LEFTP, HEADP, RIGHTP> $+_{mode}$
<WHS, LEFTS, HEADS, RIGHTS > $\rightarrow$
<WHPS, LEFTPS, HEADP, RIGHTPS>

where all the elements of the quadruples in the concatenative rule represent strings such that

HEADP represents the string to which the category
PRIM \ Flag1~PLA / Flag2~[SEC^*mode*|PRA] is assigned
HEADS represents the string to which the category
SEC \ Flag3~SLA/ Flag4~SRA is assigned
WHP is the string such that for some XP^wh in PLA, XP^wh ::= WHP and LEFTP instantiates the remainder of PLA
WHS is the string such that for some XP^wh in SLA, XP^wh ::= WHS and LEFTS instantiates the remainder of SLA
RIGHTP represents the string to be formed by concatenating the strings that introduce the types in PRA, and the empty string if PRA = [ ]
RIGHTS represents the string to be formed by concatenating the strings that introduce the types in SRA, and the empty string if SRA = [ ]
RIGHTFIELDS = RIGHTP + RIGHTS if PRA is the prefix of PRA$\oplus_{mode}$SRA
RIGHTFIELDS = RIGHTS + RIGHTP otherwise
LEFTFIELDS = LEFTP + LEFTS if PLA is the prefix of SLA$\oplus_{mode}$PLA
LEFTFIELDS = LEFTS + LEFTP otherwise

where + is simple concatenation, and furthermore

a. *if* SLA = [ ] *then* RIGHTPS = LEFTS+HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS
else if SLA ≠ [ ] and Flag3 = u *then*
RIGHTPS = HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS
otherwise for some suffix LEFTS2 of LEFTS,
RIGHTPS = LEFTS2+HEADS+RIGHTFIELDS *and*
LEFTPS = LEFTFIELDS

b. *if* FLAG1 = wh *then* WHPS = WHP
else if FLAG3 = wh *then* WHPS = WHS
otherwise WHPS = WHS+ WHP

Basically, the mapping, for each string-to-come defines in which region the string is bound to dwell; this region is defined by the given strings of the primary and secondary categories. A rightward cancelling rule *generates* if there is a generating concatenative rule that simultaneously meets the constraints above. This representation of a generating grammar is operationalized by specifying in each lexical template (complex symbol) not only a triple compositional category *a\b/c* but also a quadruple concatenative category *<w,x,y,z>*, and by having both categories related to proper parts of the template. The nature of this linking will be discussed in the chapter on the lexicon and unification. The way the quadruple is specified differs quintessentially from the way the triple is represented: the quadruple is derived post-derivationally by an operation *makestring/3* that is parametrized inside the template by unification of the strings and the modes to operate on. *makestring/*3 is defined in (221) as a Prolog predicate, operating on derivationally given quadruples to determine a new one – the quadruple holding the linearization of the template itself. The definition is followed by a scheme for a template in which it is functional; here, in quadruples *e* stands for the empty string.

(221)
```
makestring( OutString, [], OutString ) :- !.

makestring( In, [(FirstMode, FirstArgumentString)|Rest], OutString :-
       generating_rule(In ⊗FirstMode FirstArgumentString → Result ),
       makestring( Result, Rest, OutString ).
```

(222) *Template-scheme for triple and quadruple categories*

```
...
triple-category: a\_~[b^mode1] / _~[c^mode2]
quadruple-category: A
string: makestring( <e,e,Headstring,e>,  [(mode1, <W1,X1,Y1,Z1>),
                                          (mode2, <W2,X2,Y2,Z2>)],  A)
...
head: string: Headstring
...
argument(1):   type:b
               mode:mode1
               string:<W1,X1,Y1,Z1>
...
argument(2):   type:c
               mode:mode2
               string:<W2,X2,Y2,Z2>
```

Under such formalisms, the generator can be adapted into an excellent test apparatus for the parser if every cancelling rule generates. In Functional Grammar, exploring two-way grammars has been good practice in the computational enterprise from the very start (e.g. Kay 1981). This, however, does not imply that parsing and generation are themselves reversible processes. (222) shows that there is a homomorphism from the cancellation algebra into concatenation, but the mapping is not bijective. The cancellation algebra essentially computes types to get the lambda conversions right, checking linearization in order to avoid ambiguity or invalid unification. The concatenation algebra focuses on linearization only. Because linearization comes with discontinuity – dismantling constituenthood and adjacency, the prerequisites for categorial composition according to Steedman (1990) – strings cannot be the sole basis for appropriate lambda conversions. Thus, there is no homomorphism from concatenation into cancellation. The grammar is two-sided, translatable and computable, but not reversible. As a matter of fact, the triple categories steer generation in a way that is described in the next section.

When, for the sake of parsing, one needs cancellation rules that do not generate, these rules are at least suspicious. They endanger the testability of the grammar, and question the idea that we have a homogeneous grammar for each language. One such rule might be the one dealing with the recovery of so-called *r*-pronouns in Dutch (see also (23)). It allows for an 'anywhere' appearance of the category *rnp* and has to be defined recursively.

(223) *rnp-special rule*

```
PRIM\ L / _RF1~R ⊗rnp rnp \u~[] /u~[] → PRIM\ L / a~RR if
      append( PRA, [pp^6|SRA], R ) and
      rnp\u~[]/u~[] ⊗isl pp\_LF1~[rnp^isl]/PRA →
                  pp\a~[]/PRAP and
      append( PRA, [pp^rnp| SRA], RR )
```

The rule states that an *rnp* can be cancelled in any position if the primary category gives rise to a *pp*-argument that licenses the *rnp*, and that the *pp* has to be marked for being saturated with respect to that *rnp*. This rule is meant to be exceptional as a parsing rule, and it does not generate, because its context-free rule format is conditioned. We are not aware of any grammatical analysis for the phenomenon that could help us out of this impasse. Interestingly, the situation is less complex from a linearization point of view: whenever a *pp* looking for an *rnp* is absorbed, the *rnp*-string is integrated with the left field of the absorbing category. The problem with the linearization rule, though, is that it cannot be translated from (223). As a consequence, we are forced to leave open the possibility that some 'abnormal' parsing rules must be manually turned into linearizing and generating rules. Because this linearization procedure is completely opportunistic and *ad hoc*, we will not try to specify it here.

1.9.2 Categories as agendas

Reiter and Dale (1995) define generation for applications by means of a layered architecture involving at least three stages: text planning, sentence planning and linguistic realization. The generation we describe here is, of course, not an application. Yet, it may be useful to locate the strategies of the DELILAH generator as living in the interface of sentence planning and linguistic realization. Our generator creates a meaningful sentence by following a syntactical strategy. It may have as a goal to produce a sentence with certain detailed semantic properties or even expressing a certain proposition. This goal, however, is achieved by piecemeal, unification-oriented construal. In chapter 2, on semantics, a procedure will be given to generate from fully-specified logic in this manner.

Here, we concentrate on the use of combinatory categories in the generation process, for any purpose and for any input. The internal structure of these categories sets off a sequence of lexical lookup, unification of complex symbols and update of the agenda.

In the remainder of this section, we will offer a shortcut through a generation process by numbered statements. Below is the generation scheme, focusing

on agenda management. The templates are represented by suggestion, rather than by detail. The indexation of arguments is considerably simplified.

(224) *Concise overview of a generation procedure*

(a) initialization: determine the categorial hypothesis: `s`
(b) determine a lexical template with type: `s\_/_`
(c) found (randomly): *wenst* 'wish'

```
T1 = [...HEAD:CONCEPT:wish...SEM:f(..wish..)...
      TYPE: s\u~[np^wh#1]/u~[np^isl#2]...
      ARG(1):...
      ARG(2):...PHON:makestring(..wenst..)]
```

(d) fix agenda (by cancellation and merge):

```
given: [s]
to_find: [np^wh#2, np^isl#1]
```

(e) retrieve a lexical template with type `np\_/_` and properties as determined by the value for the feature `ARG(2)` in the `s` template
(f) found (randomly): *wettig* 'lawful'

```
T2 = [...HEAD:CONCEPT:lawful...SEM:g(..lawful..)...
      TYPE: np\u~[]/u~[n^isl#3]...ARG(3):...]
```

(g) fix agenda (by cancellation, merge and unification T1 ⊔ T2):

```
given: [s]
to_find: [n^isl#3, np^isl#1]
```

(h) retrieve a lexical template with type `n\_/_` and properties as determined by the value for the feature `ARG(3)` in the `s` template
(i) found (randomly): *goud* 'gold'

```
T3 = [...HEAD:CONCEPT:gold ...SEM:h(..gold..)...
      TYPE: n\u~[]/u~[] ...]
```

(j) fix agenda (by cancellation, merge and unification (T1 ⊔ T2) ⊔ T3):

```
given: [s]
to_find: [np^isl#1]
```

(k) retrieve a lexical template with type `np\_/_` and properties as determined by the value for the feature `ARG(1)` in the `s` template
(l) found (randomly): *langzaam* 'slow(ly)'

```
T4 = [...HEAD:CONCEPT:slow...SEM:k(..slow..)...
      TYPE: np\u~[]/u~[n^isl#3]...ARG(4):...]
```

(m) fix agenda (by cancellation, merge and ((T1 ⊔ T2) ⊔ T3) ⊔ T4):

```
given: [s]
to_find: [n^isl#4]
```

(n) retrieve a lexical template with type `n\_/_` and properties as determined by the value for the feature `ARG(4)` in the `s` template
(o) found (randomly): *water* 'water'

```
T5 = [...HEAD:CONCEPT:water ...SEM:m(..water..)...
      TYPE: n\u~[]/u~[] ...]
```

(p) fix agenda (by cancellation, merge and (((T1 ⊔ T2) ⊔ T3) ⊔ T4) ⊔ T5:

```
given: [s]
to_find: []
```

(q) apply `makestring/3`: `wettig goud wenst langzaam water`
'lawful gold wishes slow water'

apply **f∘g∘h∘k∘m**

T = `[... HEAD:CONCEPT:`*`wish`*`, HEAD:PHON:`*`wenst`*`...`
`SEM:` *`quant(A,gen).[water(A) & slow(A) &`*
`quant(B,some).[gold(B) & lawful(B) &`
`quant(C,some).[wish(C) & state(C) &`
`experiencer_of(C,B) & theme_of(C,A) &`
`attime(C,D) & tense(C,pres)]]]``...`
`TYPE:`*`s\a~[]/a~[]`*`...`
`ARG(1):PHON:`*`langzaam`*`...`
`ARG(2):PHON:`*`wettig`*`...`
`ARG(3):PHON:`*`goud`*`...`
`ARG(4):PHON:`*`water`*`... ]`

The yield of the generation procedure is a template that contains at least linearization of phonological units and composition of semantic functions. The linearization and the composition are the main results, and they are fed by incremental unification of templates during the procedure. Chapter 3 elaborates on this unification process. It is important to stress here that the incremental unification is steered by the agenda which is cast in terms of the output categories of the syntactic composition rules. The 'agendafication' of an output category is straightforward. For any category (225) that is the result of a cancel-and-merge rule called for in the derivation, the agenda it imposes on the subsequent generation procedure is (226).

(225) `Head\_~L1` $\oplus_i$ `L2/_~R1` $\oplus_i$ `R2`
(226) given: `Head`
to_find: `L1` $\oplus_i$ `L2 and R1` $\oplus_i$ `R2`

Technically, the agenda is added to the existing one under cancellation of the *given*-value from that agenda, if possible. If not, both the head and the appended argument lists are added to the *given* and *to find* values, respectively. Below is the generating algorithm, amounting to an agenda regime; first, the well-known notion of logical *backtrack* is defined procedurally.

(227) *Backtrack*
the statement *backtrack* p_1, ...p_{n-1} *until* p_n means:

find a valuation V *such that* for all i, V(pi) is true
if such V exists *then* proceed to the next statement
else regress to previous statement

(228) *Generating algorithm*

a. *input* **hypothesis** H
set **agenda** *to* <given: [ ], to_find: [H]>
set **structure** *to* [ ]
b. *backtrack*
c. *if* **agenda** = <given: [ ], to_find: [Only]> *then*
select a template T^{only} *with* category(T^{only}) = `Only\L/R`
set **structure** *to* [T^{only}]
set **agenda** *to* <given: [Only], to_find: L+R >
endif
d. *if* **agenda** = <given: Given, to_find: ToFind> *then*
e. *backtrack*
f. *if* Given = [First | Rest] *then*
extract a T^{First} *from* **structure**
select a template T *such that for some rule*
`category(T) ⊗ First\L/R → New\LNew/RNew`
set T^{New} *to* T ⊔ T^{First}
endif
g. *if* Rest =/= [ ] *then*
for some R ∈ Rest *determine some rule*
`New\LNew/RNew ⊗ R\L7/R7 → New\LN2/RN2`
extract a T^{R} *from* **structure**
set T^{New} *to* T^{New} ⊔ T^{R}
replace in **structure** T^{First} *by* T^{New}
remove T^{R} *from* **structure**
set **agenda** *to* <given: Rest-[R]+[New],
to_find: LN2+RN2+ToFind>
else
replace in **structure** T^{First} *by* T^{New}
set **agenda** *to* <given: [New],
to_find: LNew+RNew+ToFind>
endif
until Given = [H] % end backtrack (e)
h. *backtrack*
if ToFind = [First | Rest] *then*
select a template T *with* category(T) = `First\L/R`
% by now, structure is a singleton set
extract T^{Only} *from* **structure**
set **structure** *to* [T^{Only} ⊔ T]
set **agenda** *to* <given: Given,
to_find: L+R+Rest>
endif
until ToFind = [ ] % end backtrack (h)
i. *endif* % end *if* (d)
j. *until* **agenda** = <given: [H], to_find: [ ]> % end backtrack (b)
k. *output* **structure**

Basically, the algorithm tries to find templates and to unify them according to an agenda which is set by an initial hypothesis and updated by applying combinatory categorial rules. The agenda consists of two parts: *given*, corresponding with complex symbols already adopted, and *to_find*, corresponding with structures still to be checked. The result of successfully unifying complex symbols according to the agenda is accumulated in a store *structure*. The procedure succeeds if the *to_find* part of the agenda can be discarded, and the *given* section contains only the initial hypothesis. The complex symbol in *structure* is the yield of the procedure. The repeat-loop (228)b-j does the job. It checks the agenda and acts accordingly. Firstly, it tries to reduce the *given* part to a single occurrence of the hypothesis, in (228)e-g. After that, it looks for complex symbols matching the *to_find* agenda, in (228)h. In practice, every step of the algorithm can be sugared with additional directives. For example, in steps (228)f and (228)g it might be desirable to have the algorithm select templates with categories headed by the hypothesis. In general, the *select* subprocedure can be regimented by shortcuts respecting the agenda. As a matter of fact, the procedure we propose here amounts to posing additional restrictions on *select*. In particular, *select* will be restricted by conceptual conditions derived from logical form. These conditions are discussed in chapter 2.

Halting is not an intrinsic property of the procedure: the length and the complexity of the expression to be produced are not fixed. For practical purposes, however, we may impose restrictions on the length of the sentences to be produced. Suppose that is the case. Given such a restriction, we must investigate the soundness and completeness of the procedure in the following sense:

(229) *Soundness*
for every template T used in the production of a sentence S, there is a parse of S using T

(230) *Completeness*
for every template T used in the parse of a sentence S, there is a production of S using T

The soundness of (228) depends on the soundness of the translation (220) and the completeness of the parser. The generator cannot be better than the parser. We assume that by the translation and unification procedure underlying both parsing and generation, the generated sentences are among the parsable ones. This is an empirical matter, however. The generator can always be tuned or even made sound in this respect by having its output tested on the parser. Note that this does not invalidate the idea that the generator can be used to test and cure the parsing grammar.

The completeness of (228) is a much more complex affair. The question is whether we can prove that every grammatical expression – within the limits of the parsing grammar – can be produced, *i.e.* can be part of a successful production. The burden of proof is on the design of the selection procedure. In favour of completeness are the following considerations:

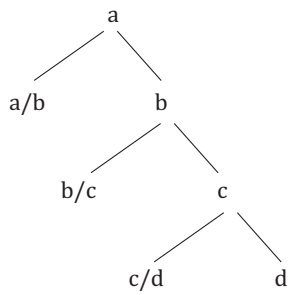
- the selection for a particular agenda can be organized as trial-and-error by pop-up without return from a randomized stack;
- the stack on which selection operates is finite by definition and thus backtracking is enhanced;
- there are no inhibitions for the introduction of complex categories, including recursion.

The last argument *pro* completeness, however, is dangerous. Under the assumptions of limited length, the complexity of categories must be controlled somehow. To make this work, we have to assume that for every type X (with a possible exception for the hypotheses) representatives with zero categorial complexity are available, *i.e.* templates of the category X\u~[ ]/u~[ ], for only selection of these arguments reduces the arity of the *to find*-agenda. This requirement is met by the parsing grammar, however. Under the principles of count invariancy (47) and antisymmetry (cf. section 1.5.3) of the combinatorial rules, the parser can recognize sentences only if a few of these 'zero' categories exist lexically. If we – additionally but perfectly reasonably – assume that every instance of a complex category can be reduced by some cancellation, the requirement that zero argument categories are available is met – actually, by virtue of some normality condition on the grammar. At the same time, categories of a certain complexity may be the only solution to the agenda problem. The situation of (228)g can be solved only if there are categories which have at least two arguments (complexity ^{3} 2). The existence of such categories cannot be assumed trivially. Consider sequences like

(231) a/b d b/c c\d → a
a/b b/c c/d d → a

These sequences are designed perfectly, without containing any category of a higher complexity than 1. Is there any reason to believe that these sequences do not or could not occur in natural language? There is nothing wrong in natural-language analysis with a binary branching left-headed tree like (232).

(232)



It might be the format of a simple transitive sentence with object *c* and subject *a/b*, for example. It might even be the dreamt configuration for antisymmetrical syntax with lambda annotation. Moreover, if the complexity of categories were below 2, agenda management at the *to_find*-side would be considerably simplified: this part of the agenda could never increase in complexity – it would be at most 1, by definition. Therefore, it is ill-advised to require the generating syntax to be such that it contains categories of complexity 2 or higher. Rather, we would have the algorithm enriched with a check on the maximal complexity of categories meeting certain conditions, and have these checks abort the present search and force backtracking if the required complexity is no longer available in the lexicon.

Alternatively, we may consider how to avoid the situation covered by (228)g that the *given* agenda contains more than one type. In our experience, this situation occurs when a set of unrelated side-conditions to the generation is processed initially. If one of them cannot be passed to the actual *to_find*-agenda by cancellation, the category associated with the side-condition is by necessity passed on to the *given* agenda, which is already inhabited by the hypothesis, at least. It takes brute force at the control side to avoid such a situation, passing the account of side-conditions on to the *to_find* agenda under the regime of (228)h. To the extent that one succeeds in this task, backtracking and termination for the generation process are enhanced, independently of accidental features of the lexicon and the grammar. By the general design of selection sketched above, randomization and backtracking ensure the completeness of the generation procedure.

Of course, completeness (230) does not imply that every parsable sentence will be generated at some moment by the procedure. It ensures, however, that the procedure can be adopted in an enumeration automaton to this effect. Thus, DELILAH's grammar embodies a generator that mirrors the parser.

2. SEMANTICS: the game of scope and intensionality

2.1 THE WAYS OF MEANING

Language refers. It cannot help doing so. As soon as it hits the mind – and is experienced as language – it is *about something other* than its forms. Language refers to something outside of itself. It is the only system with this property. Therefore, no other system needs to be interpreted the way language is interpreted. Language refers by encoding propositions – statements expressing the way of being of, or the state of affairs in, another system. This system we may call a *model*. A system is a model because it is described by a proposition, to the extent to which it is described by that proposition. For semantic analysis, it is irrelevant whether the model is independent of the proposition – that is a matter of philosophy, and has been intensively discussed in medieval logic. A proposition refers to a model, by definition. This way of *being about a model* we call meaning.

The interesting thing about propositional meaning is that it is not conventional. It is not given by some canonical prescription. Therefore, it is not a genuine topic of debate among language users. The meaning of a proposition is something on which those that know a language converge – on penalty of failed communication. In this respect, the meaning of a proposition is something completely different from the meaning of a word or a phrase. A proposition does not mean something *because* its phrases are point-wise meaningful: it means something *notwithstanding* the fact that we do not know or do not agree what the words or the phrases mean. That is: we do not know whether our interpretation of words is the same as someone else's – the speaker's, for example – and we do not need to know. To paraphrase Frege's *Nur im Zusammenhang eines Satzes bedeuten die Wörter etwas*: phrases mean something if

and only if they occur in meaningful propositions. The proposition's meaning is primary; the meaning of a phrase is derived.

In this respect, natural language essentially differs from formal languages. There we find finite, fixed and of course conventional interpretations for all meaningful phrases in a sentence. It would be almost perverse to come up with artificial languages the lexicon of which is semantically flexible. In natural languages, however, only functional elements – those that many speakers consider to be meaningless or indefinable – can boast that they are well-defined, to a certain degree. The compositionality of natural languages is so impressive that it obscures lexical indeterminism. Words do not mean a thing, but they are saved by the proposition; the proposition assigns types to the words, semantic roles to be played in a functional space. From this type, the nature of a phrase's contribution to the overall meaning is induced. This is how we discover, and never stop discovering, (aspects of) words and phrases, according to Bloom (2002). We are able to use and understand words to the extent to which we are able to use and understand sentences.

In this vein, it is intriguing that in Wierzbicka's Natural Semantic Metalanguage (NSM, *e.g.* Goddard 2002) words represent extremely complex concepts, the definition of which almost amounts to propositions – which we would expect for meanings of types <α, t>. Here follows, as an example, Goddard's (2002) analysis of a phrase of type *X is watching Y*:

(233) for some time X was doing something because X thought: when something happens in this place I want to see it; because X was doing this, X could see Y during this time

The idea that this meaning is built from universal conceptual atoms is less intriguing for the present exposal than the definitely propositional format which Goddard resorts to. Unfortunately, NSM hardly deals with the logical structure of phrasal meanings. In the logical framework of DELILAH to be discussed below, however, (233) would have to be converted into an extensive spell-out of logical units and structure:

(234) at some period *p* before *now*, there was an event *e* with agent X and place *pl* and there is a state of thinking *s* with agent X and theme *ts*
and
s was the reason for *e*
and

t is the proposition
(there is a state *w* of wanting with agent *X* and theme t*w*
and
tw is the property of X such that if there is an event *ev* at *pl* there is a state *si* of seeing with patient X and theme *ev*)
and
there is a state *se* of seeing with patient X and theme Y during *p*
and
there is a state *sp of* being possible with theme *se*
and
sp had reason *e*.

Not everyone will be convinced that (233) or (234) represents the canonical meaning of the verb *to watch* in its progressive form. One may even consider these to be defective, or overdone. Yet, the proposition-like structure of lexical meanings in this universalistic approach of NSM can be seen as a statement that we need propositions to register, encode and store lexical meanings, and consequently, that propositions underlie lexical meaningfulness. The meaning of a proposition is not an agglomeration of simpler or atomic meanings: it is generated by structure.

These remarks concern neither the organization of grammar nor the process of parsing and generation. In the production and processing of language, human beings rely on compiled routines rather than on philosophical hierarchies. In particular, the lexicon is the place where knowledge (experience, skills, insights, generalizations, guesses) of language aggregates to complex symbolizations – not necessarily elegant, not necessarily principled but extremely efficient. These very complex symbolizations reflect our versatile competence, including its semantic creativity.

The role of a lexicon in the *computation* of natural language is addressed in chapter 3. In the present chapter we are concerned with the way Dutch propositions are meaningful. There is not much that is particular about the meaning of Dutch. We do not have reason to assume that ways of being meaningful vary with the language. We must assume, however, that the propositional meaning is conveyed by the structure of the language, and thus by its particular grammar. This assumption is known as *compositionality* – according to Janssen (1986) wrongly attributed to Frege, but underlying all formal languages and as such effectuated for the interpretation of natural language by Richard Montague (*e.g.* Montague 1972) and many others. Compositionality implies that propositional meaning results from the way a proposition is

built. Since grammar is not debated among those who know a language, the construal of propositional meaning in that language is not either. Thus, the fundamental meaningfulness of a language depends both on the generality and on the particularity of grammar.

As it stands, we do not claim that bare structure is meaningful. We assume that the grammar both structures and labels sentences, that this combined effort induces types, and that these types – semantic functions – are sensitive to specification. It is not the semantic specification that is computed but the typological, functional semantic tier. That is where meaning grows and lives. Here is what we consider to be the backbone of meaning.

(235) labeled structure

$[[_{np}\ no_{det}\ dutchman_{n}]\ [_{ip}\ can_{aux}\ [_{vp}\ be_{aux}\ fooled]_{vp}\ always_{adj}]]]$

(extensional) types

<<et> <et>t> et <<et><et>> <<et><et>> et <<et><et>>,

<<et> <et>t> et <<et><et>> <<et><et>> et <t, t> ...

meanings

$f_{no}(g_{dutchman})\ (k_{always}\ (h_{can}(\ i_{be}\ (j_{fooled}))),$

$f_{no}(g_{dutchman})\ (h_{can}\ (k_{always}\ (\ i_{be}\ (j_{fooled}))))$ ...

Our understanding of compositionality is not that there be a *function*al relation between labels, types and meanings. Compositionality in the montegovian sense sees to a well-defined relationship between the specifications of form and the specifications of meaning. In Montague (1972), this relationship was conceived of as a mapping between syntactic and semantic rules of construal. The rules of construal, however, were produced neither by mathematics nor by aesthetics or economy – they were opportunistic but operational. Though it is wise to look for principles and restrictions governing the rules of construal, compositionality is not a derivative of these constraints on grammar. In particular, it is not the case that syntactic labeling and semantic typing must be homomorphic for compositionality to be guaranteed. A functional relation between syntactic labeling and semantic typing is a valid anchor for compositionality, but it is neither necessary nor sufficient to entertain a computable relation between form and meaning. Approaches to natural-language semantics like Discourse Representation Theory (Kamp and Reyle 1993) and Head-Driven Phrase Structure Theory (*e.g.* Sag 2003) exploit all kinds of mechanisms, strategies and principles to assure compositionality. These instruments are beyond form-meaning homomorphisms of the sort elaborated in Hendriks (1993) and in type-logical grammar more generally (*cf.* Carpenter 1997, Morrill 1994).

This chapter offers a quite eclectic view on the construal of meaning. It defines three different but related layers of propositional semantics. It marks unification as the process feeding into semantic construal. It takes no position whatsoever on the meaning of particular items. In particular, it leaves open the possibility that lexical items have complex, proposition-like meanings – like (233) or (234) – which also contribute to the overall propositional semantics by unification. The abducted representations live on a much richer language than first-order predicate logic, in order to account for the variety of quantification, for event structure, for tense, mood, aspect and other issues beyond the borders of classical logic. The chapter also leaves room for post- and extra-derivational algorithms dealing with semantically defined constructions like anaphora and ellipsis. Moreover, the following sections will focus on formal entailment such as the litmus test for semantic representations. In short, the syntax-semantics interface may be more like a jungle than like a French garden.

The jungle enjoys some ordering, however, by a tight and intrinsic relationship between entailment and logical form. Entailment is a convergence of natural-language speakers on the necessity of the relationship between two sentences. An entailment is a converging judgement, and can be measured among language users. Logical form is a linguistic artefact, meant to model linguistic meaning. The following definition reflects the relationship between the entailment judgements and the calculus of logical form.

(236) *Logical Form*
[[X]] is *the* logical form of a sentence X if for some logic L,
S entails P *iff* $[\![S]\!] \vdash_L [\![P]\!]$ and $[\![S]\!] \models_L [\![P]\!]$.

The definition stresses the need for a well-defined deductive mechanism defined on logical forms that calculates entailments. We take the development of that logic to be a major task for semantic theory.

2.2 THE FORMS OF MEANING

Montague (1972) – hereafter: *PTQ* – introduced a package for meaning representation that we take as a basis for our efforts to interpret Dutch mechanically. The package includes

(237) higher-order functional terms with application and composition
semantic typing
intensionality and intensional embedding
full scoping
post-derivational meaning postulates.

The DELILAH semantics implements this package in the following way:

(238) higher-order functional terms with application and composition
implicit extensional semantic typing
intensional embedding
underspecified representation at derivation (but)
post-derivational, syntactically constrained unfolding of full representation.

In this sense, DELILAH is tributary of PTQ. By extending the 'fragment', however, with an account of peculiarities of Dutch, by focusing on inference and entailment and by adding many additional features, our system comes up with representations which are hardly recognizable as montegovian derivatives. Here is an example, reconstructing the diversity of modules in PTQ, with its intended restyling in DELILAH's various semantic formats; the comparison is enhanced by the fact that DELILAH uses English words to represent lexical concepts – just like PTQ.

(239) *Every man seeks a unicorn*
$\lambda\mathcal{P}.\ \forall z.\ [\textbf{man'}\{z\} \rightarrow \mathcal{P}\{z\}]$
$(\hat{}\lambda y.\ \textbf{try-to'}(y, \lambda\mathcal{Q}.\exists x.\ [\textbf{unicorn'}\{x\} \ \&\ \mathcal{Q}\{x\}](\hat{}\lambda z.\textbf{find'}(y, z)]))$

This formula is the semantic yield of a particular derivation of the sentence *every man seeks a unicorn* in PTQ. It says, roughly, that the function from intensional properties to propositions corresponding to *every man* is to be applied to the function from individual concepts to propositions that interprets *seeks a unicorn* as the relation *try-to* between individuals and the property of finding a unicorn. The (restyled) derivational product in DELILAH that corresponds to this formula and its derivation reads like (241); we abstracted away from all 'book-keeping' information which occurs in the machine, and just give the mere functional structure. It is formatted as a storage of lambda terms, called *Stored Logical Form*; its main structure is indicated in (240), by abstracting over all operators and lexical specification.

(240) *Stored Logical Form* < Store, Body> *abstract*

```
< [store0
<<[store1 man 1erots] every >>,
<<[store2 <<<[store3 <<<<[store4 unicorn 4erots], some >>>> 3erots],
find >>> 2erots], agent_of(U, F) & theme_of(U, P) & attime(U, W)
> 1erots] property(X(L))>>, try 0erots],
agent_of(D, I) & theme_of(D, Z) & attime(D, A) & tense(D, pres)
>
```

(241) *Stored Logical Form* < Store, Body> *full detail*

```
< [store0
<<[store1 λI.state(I, man) 1erots] λJ.λK. quant(I, every) & J(I) &
entails1(I, decr) & K(I) & entails(I,incr)>>,
<<[store2 <<<[store3 <<<<[store4 λP.state(P, unicorn) 4erots], λQ.λR.
quant(P, some) & Q(P) & entails1(P, incr) & and(R(P) & entails(P,
incr) >>>> 3erots],
λT.quant(U, some) & event(U, find) & entails1(U, incr) & T(U) &
entails(U,  incr)>>> 2erots],
λW.λU.λP.λF. agent_of(U, I) & theme_of(U, P) & attime(U, A) >
1erots],
λX.λY.quant(Z, the) & property(X(L)), Z) & entails1(Z, decr) &
Y(Z), entails(Z, incr)>>,
 λC. quant(D, some) & event( D, try) & entails1(D, incr) & C(D)
& entails(D, incr)
0erots],
λI.λZ.λD.λA. agent_of(D, I) & theme_of(D, Z) & attime(D, A) &
tense(D, pres) >
```

The structure establishes the following. There is a finite thematic signature of an agent and a theme to a *try*-event: the second element in the main structure <α,β>. The agent is a universal quantifier over *man:* the first element of *store0*. The theme is a quantifier over a property marking an intensional domain – there are no intensional types. This property is a thematic signature to an event *find*. This event is a signature over a free (but controlled) agent and a theme introduced by an existential quantifier over *unicorn.* In all cases, the target variables for quantification are identified with the bound variable, to compensate for the lost book-keeping of conversion. The specification of temporal objects is left unresolved, but they are identified in the two event-signatures. The structure of SLF is further discussed in section 2.6.3.

An immediate difference between (239) and (241) is that in (241) the scopal relation between the quantifiers involved is still underdetermined. The PTQ formula may, however, be subject to meaning postulates to the effect of reordering the dependencies between the existential and the universal quantifier and the intensional domain of the complement of *try.* Moreover, the lambda terms in the Stored Logical Form abound in analytical specifications from the

lexicon, whereas PTQ is very reticent in this respect. Partially, these specifications follow from adopting an event-structure analysis. Other specifications like *entail*, *entail1* and *property* are meta-predicates indicating semantic domains relevant to the spell-out of full scopal semantics. Here are the three fully-scoped analyses, called *Applied Logical Form,* which DELILAH derives from (241) by a post-derivational algorithm; they correspond to the scope orderings *every-intension-a*, *every-a-intension* and *a-every-intension*, respectively. The event-related existential quantifiers are not scoped, for reasons to be discussed in section 2.5.

(242) Applied Logical Form

(a)

```
quant(B,every).[man(B) ⟹ quant(C,some).[event(C, try) &
quant(D,the).[property(D).[ quant(A,some).[unicorn(A) &
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]
```

(b)

```
quant(B,every).[man(B) ⟹ quant(C,some).[event(C, try) &
quant(A,some).[unicorn(A) & quant(D,the).[property(D).[
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]]
```

(c)

```
quant(A,some).[unicorn(A) & quant(B,every).[man(B) ⟹
quant(C,some).[event(C, try) & quant(D,the).[property(D).[
quant(E,some).[event(E, beat) & agent_of(E,B) & theme_of(E,A) &
attime(E,F)]] & agent_of(C,B) & theme_of(C,D) & attime(C,F) &
tense(C,pres)]]]]
```

In this representation, the event structure is naturally maintained, but the meta-predicates are mostly resolved. The quantification over *property* is preserved, though, as it marks intensional embedding in an extensionally typed logic. Apart from the lexical decomposition, this representation is on a par with the result of applying β-conversion and meaning postulates to (239):

(243) ∀x. [**man'**{x} → **try'**{x,^∃z. [**unicorn'**{z} & **find'**{x, z}]})

Apart from the 'classical' analysis (242), DELILAH derives another representation from (241), the *Flat Logical Form*. In this logical form, all scopal dependencies induced by quantification and intensionality are compiled onto every occurrence of every bound variable. Moreover, the scopal operators them-

selves are reduced to indices on the variables, with their main inferential attribute. For example: the clause `event(E+↑+some+[D], beat)` is to be read as follows.

(244) `event(E+↑+some+[D], beat) ::`
the (meta-)relation *event* holds between the bound variable *E* and the concept *beat*,
E is referentially (as for its valuation) dependent on the (bound) variable *D*,
E is bound by the quantifier *some* and here allows for *increasing* inferences with respect to the predicate of which it is an argument.

The remaining structure is essentially a flat conjunction of fully-specified small clauses of this nature, each of which is entailed under a certain regime accounting for referential dependency and quantification (see section 2.6.5). Below is the *Flat Logical Form* derived from (241) and equivalent to the family of formulas (242).

(245) Flat Logical Form

```
state(B+↓+every+[], man) &
event(C+↑+some+[B], try) &
property(D+↓+the+[]) &
event(E+↑+some+[D], find) &
agent_of(E+↑+some+[D],B+↑+every+[]) &
       (theme_of(E+↑+some+[D],A+↑+some+[D]);
        theme_of(E+↑+some+[D],A+↑+some+[B]);
        theme_of(E+↑+some+[D],A+↑+some+[]))
&
       (state(A+↑+some+[D], unicorn);
        state(A+↑+some+[B], unicorn);
        state(A+↑+some+[], unicorn) ) &
attime(E+↑+some+[D],F) &
agent_of(C+↑+some+[B],B+↑+every+[]) &
theme_of(C+↑+some+[B],D+↑+the+[]) &
attime(C+↑+some+[B],F) &
tense(C+↑+some+[B],pres)
```

It spells out all possible semantic dependencies in the sentence, as represented by the applied logical forms in (242). In Flat Logical Form, distinct readings accumulate as a disjunction of those clauses for which scopal alternatives are available. Thus, in (245) the scopal variation shows up as a disjunction of small clauses `state(A+↑+some+[...], unicorn)`, differing from each other only in the fourth term of the variable specification – the dependency marking. The conjunctive frame of Flat Logical Form can be seen as an index – and a meta-representation – of Applied Logical Form.

Clearly, the relationship between PTQ and DELILAH's semantics is not straightforward. In order to deal with the extremely rich semantic profile of natural languages and to make automated interpretation 'inference proof', we need more extended and more versatile formalisms than the one Montague applied. Yet, we consider our way of dealing with automated interpretation as anchoring in Montague's program of computing meanings for forms.

2.3 SCOPE AND SPECIFICATION

2.3.1 Quantifiers

Basically, sentences express predication, modification, mood and quantification. Take the sentence

(246) Enkele klassieke teksten waren niet goed bestudeerd
some classical texts were not well studied
'Some classical texts were not studied well'

The semantic structure may be pictured as follows, with predicates in standard font, mood in bold, modifiers in italics and the quantifier in small capitals

(247) SOME (text, *past*(**not**(*well*(studied))))

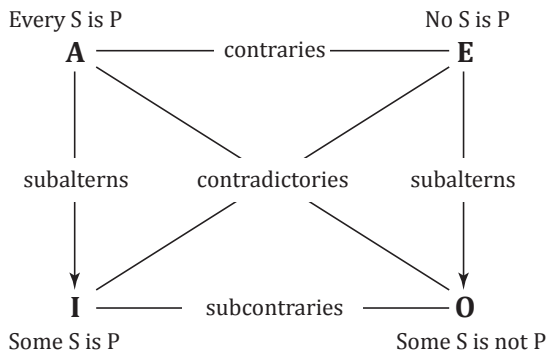
In our approach, modification reduces to predication. By reification of predicates in event structures (see section 2.5), all modification of basic predicates can be cast as properties of the object associated with that predicate: if *walks* is interpreted as an event *e*, the modifier in *walks fast* predicates over that object *e*. Mood certainly is an operator over predication: negation has to be expressed on predicates, and is always introduced as a distinct lexical unit. Quantification, however, is a distinct phenomenon. It relates predicates independently of their nature, and cannot be reduced to predication by enriching that notion. This was Aristotle's insight in the *Prior Analytics*. From these insights, Boethius constructed the square of opposition that governed medieval logic. He defined four quantifiers: A and I (from Lat. *AffI*rmo 'I affirm') and E and O (from Lat. n*EgO* 'I deny'). A, I and E correspond to *all*, *some* and *no*, respectively. O is best approximated to *not all* or *some...not* – no language

has a lexicalized determiner for this quantifier, and that is not an accident (Jaspers 2005). The four quantifiers were related as follows:

(248) by necessity, the propositions $A\varphi$ and $O\varphi$ differ in truth
by necessity, the propositions $N\varphi$ and $I\varphi$ differ in truth
by necessity, the propositions $A\varphi$ and $N\varphi$ cannot be both true
by necessity, the propositions $O\varphi$ and $I\varphi$ cannot be both false
by necessity, $A\varphi$ entails $I\varphi$
by necessity, $N\varphi$ entails $O\varphi$.

Usually, this family of relations is pictured as the Square of Opposition (source: http://plato.stanford.edu/entries/square/):

(249)

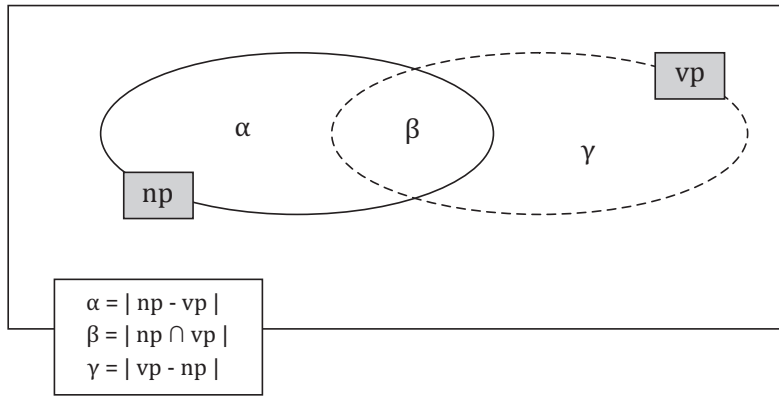


In this diagram, the counterparts to the quantifiers in natural language are added. From these, one can read that the I and the E corners express symmetric quantifications in that the S and P predicates can be inverted *salve veritate.* In the course of the centuries, it has been debated *passim* whether the O corner has existential import and is proper, and, in the same vein, whether S is true of at least one entity for the A corner to be true, or cannot be empty generally. The relation in the square between the two conjectures is revealed in Kneale and Kneale (1962). Suppose there are no Ss. Then I over S and P is false. By contradiction, E is true. By entailment (sub-alternation) O is true. But then, I is not necessarily false, as was assumed. So either S cannot be empty or O has no existential import. Jaspers (2005) argues convincingly that the O corner is linguistically, psychologically and computationally exceptional and must be abandoned. Seuren (2006) reverts to boethian logic in stressing that natural logic for natural language does not deal with empty sets and properties that no object has.

Part of this discussion on the logic of quantification can be projected to the need felt in Generalized Quantifier Theory to restrict the set of possible quan-

tifiers. Particularly illuminating in this respect is the numerical approach of Van Benthem (1986), illustrated by the following figure.

(250)



The open oval indicates the extension of the nominal predicate (S in the Square) in a quantified sentence; the grey one is the extension of the verbal predicate (P in the Square). Van Benthem observes that most quantifiers in natural language can be defined as restrictions on α and β, the cardinality of their respective subsets. That is, these quantifiers are 'about' the *NP* and in that sense *conservative*. For their evaluation, the only domain needed is the *NP*. Moreover, the restrictions are numerical. Below are two alternative definitions of the classical quantifiers.

(251)

A:	$\alpha = 0$	;	$\alpha < 1$
E:	$\beta = 0$	;	$\beta < 1$
I:	$\beta \neq 0$	;	$\alpha > 0$
O:	$\alpha \neq 0$	;	$\beta > 0$

One could choose, in the light of the discussion above, to add numerical statements in order to guarantee existential import.

(252)

A:	$\alpha < 1, \alpha + \beta > 0$
E:	$\beta < 1, \alpha + \beta > 0$
I:	$\beta > 0$
O:	$\alpha > 0, \alpha + \beta > \alpha$

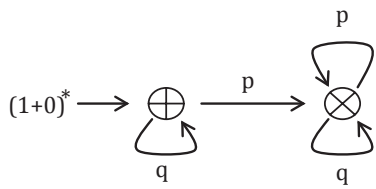
The γ value in (250) is only needed for certain classes of quantifiers, called *non-conservative*, like *only*.

(253)

only:	$\gamma = 0$	$(\beta + \gamma > 0)$
not only:	$\gamma > 0$	$(\beta + \gamma > 0)$

One of the problems with the Square of Opposition and the logics built on it for linguistic analysis is that there are many more quantifiers in natural language than these four, and that not all of them can be reduced to some finite combination of these four. They are called *higher-order quantifiers*, and Van Benthem (1986) proves them to be those quantifiers that cannot be modelled by finite state automata. *Many* and *most*, for example, are among them, but *only* is not. To start with, we give the definitions of finite state automata for the classical quantifiers and for *only.* They are taken to operate on sequences over {1,0}, where, for the classical quantifiers, 1 indicates an object in the β section of (250) and 0 an object in the α section, and for *only* 0 is an object in the γ section. As a matter of fact, the automata for A and for *only* are the same, under this proviso.

(254)



A: + = accepting, x = rejecting,
p = 1, q = 0
E: + = accepting, x = rejecting,
p = 0, q = 1
I: + = rejecting, x = accepting,
p = 1, q = 0
O: + = rejecting, x = accepting,
p = 0, q = 1

It follows that all quantifiers of the type *at least n* for finite integers *n* can be processed by automata with *n*+1 states, and quantifiers of the types *at most n* and *precisely n* by automata with *n*+2 states. The particular status of the O quantifier (*some ... not* or *not all)* can be read from the circumstance that its automaton's initial state rejects 'positive' evidence: the canonical 'models' for the predication – the members of β, represented by 1s in the input of the automaton – are irrelevant to the accepting state.
Clearly, the processing of quantifiers like *most* and *many* needs memory, since they express relations between numbers; in terms of (250):

(255) *most:* $\beta > \alpha$
many: $(\alpha + \beta) / \beta > k$ *or*: $(\gamma + \beta) / \beta > k$, for some *k*, $0 < k < 1$ or $0 \leq k \leq 1$

Most can be processed with simple stack memory or a counter. *Many* and its like require more sophisticated but not very complicated, memory management beyond context-freeness. Notice, though, that *many*, according to (255), has both a conservative and a non-conservative interpretation.

Quantifiers are relations between predicates, and predicates can be complex. In particular the predicates related by quantifier X may give rise to a quanti-

fication Y. Typically, in that case X and Y share at least one of their predicates, while at least one of the four predicates involved is independent. Here are some classical examples, with an informal analysis where *smthing* indicates the locus of dependency.

(256) Every man loves a woman

P^{every}:	*love_someone*	S^{every}:	*man*
P^{a}:	*to_be_loved_by_smthing*	S^{a}:	*woman*

(257) Every farmer who owns a donkey, beats it

P^{every}:	*beat_smthing*	S^{every}:	*farmer_who_owns_smthing*
P^{a}:	*be_beaten_by_smthing*	S^{a}:	*donkey*

(258) Every gentleman who meets her will salute a lady he respects

P^{every}:	*salute_smthing*	S^{every}:	*man_who_meets_smthing*
P^{a}:	*to_be_saluted_by_smthing*	S^{a}:	*lady_who_is_respected_by_smthing*

(257) has been recorded as the donkey-anaphora phenomenon, and (258) as the Bach-Peters paradox (Bach 1970). The occurrence of anaphora here, however, and its syntax should not obscure the underlying phenomenon of semantic dependency. When quantifiers share a predicate, their valuation gets intertwined. For some quantifiers, this intertwining leads to dependency: the relation they express is governed by an instantiation controlled by the other quantifier. Some quantifiers, typically I and its derivatives, are sensitive to this government: their valuation may vary with the valuation of the other quantifier. Other quantifiers, like A and O, are unaffected. In (256) we have the choice to have the evaluation of I vary or not vary with the evaluation of A; the sentence is ambiguous in this respect. In the first case, when the evaluation of I varies with the instantiation controlled by A, we say that I is *in the scope* of A. It is not very useful – though common practice – to say that in the other case, A is in the scope of I, since the evaluation of A will never vary with the instantiation controlled by I. In general, we can say that only those quantifiers that have a definition in terms of (250), like

(259) Q: $\beta \geq n$

for some independent integer $n > 0$ are sensitive to scope. That is, quantificational ambiguity is relevant neither in sentences with only A, E or O quantifiers nor in any of the following complex quantifications.

(260) Every man loves *at most three / many / most / the / only* women

The source of the sensitivity of I quantifiers must lie in the fact that they may be in competition with other quantifiers as for *aboutness* of the sentence. By

definition, they introduce entities that are both S and P. No other type of quantifier does. The sentence is *about* these entities, by definition. The *aboutness* claim of another quantifier, however, may interfere with I's one, if that quantifier introduces a different focus – as A does, for example, by not introducing entities in the intersection of the predicates but requiring one difference between them, *NP-VP*, to be zero. In that situation, the *aboutness* claims have to be ordered with respect to each other. Thus, we would expect that *only* and *not all* may also have an *aboutness* conflict with I. And they do: the next sentences are as ambiguous as (256).

(261) Only men loved a favourite actress of Hitchcock's
Not every man loved a favourite actress of Hitchcock's

No conflict arises when two Is meet. Since they share at least one predicate and target at the intersection of the predicates only, they focus on the same subset of the universe. In that case, both I quantifiers are *dynamic*, in the sense that they can both occur referentially and license anaphora outside their syntactic domain – the Dekker (1993) litmus test to tell dynamic from static quantifiers:

(262) Some_i men read a_j book by Chomsky. They_i had bought it_j for 10 bucks.
(263) All_i men read a_j book by Chomsky. * They_i had bought it_j for 10 bucks.

The determination of semantic dependencies between quantifiers must be a target of linguistic analysis. In DELILAH, it is taken care of by a complex post-derivational procedure dubbed *apply_store* that inspects Stored Logical form and – among other tasks – determines for each quantifier whether or not it might be in the scope of one or more others. It spells out possible ambiguity, and thus maps the Stored Logical Form on a family of disambiguated readings. In doing so, it takes into account syntactic as well as semantic information accumulated in the derivation. It will not mechanically produce representations of all possible scopal variation; it restricts its output to semantically distinctive readings. In Flat Logical Form, these readings are assembled as local disjunctions of the relevant predicates.

2.3.2 The structures of quantification

In all languages, proper names and quantificational *NPs* share most paradigms. In particular, argument positions at predicates are open to both. Yet, they are semantically different. Names denote rigidly, as Kripke (1972)

argues, but quantifiers denote contingently, varying with situations and the extension of predicates. It took linguistics some millennia to reconcile the distribution and the typing of the different sorts of nominal constituents, and it took a logician to find the key. Montague (1972) typed both proper names and quantificational *NPs* as characteristic functions over predicates, and thus puts them in the same semantic league as higher order sets, in compliance with their equal distribution. The general format of a one-place predication in natural language, then, is that a sentence $[_s$ NP VP] is to be interpreted as the proposition that the meaning of the NP applies to the meaning of the VP, in this case: ⟦VP⟧ ∈ ⟦NP⟧. Barwise and Cooper (1981) argued that the rich algebraic structure of these characteristic sets can, and indeed should, be exploited for linguistic analysis. They coined the term *Generalized Quantifier* for the algebraic objects that could be identified in the realm of the power-set of the universe (the universal predicate) as interpretations for natural-language NPs. The general definition boils down to this:

(264) *Generalized quantifier*
A set of subsets Q of the universal predicate U is a generalized quantifier *iff* there is a subset A of U and a well-defined relation R, such that X is in Q iff R(A,X) applies.

That is: a generalized quantifier requires a property to *live on* and a relation to generate the set from this property. For normal quantificational *NPs*, the property-to-live-on comes with the nominal predicate. In the case of proper names, it is the singleton of being a particular individual, like {j}. This property limits the *aboutness* of the quantifier. The class of relations is given with the algebra of sets < ∪, ∩, ⊆, − > and numerical conditions on sets. To find the restrictions limiting this class of relations is a main target of modern semantics. The classical quantifiers A, E, I and O and the proper name Socrates are defined below as generalized quantifiers.

(265) ⟦every N⟧ = { X ⊆ U | ⟦N⟧ ⊆ X }
⟦some N⟧ = { X ⊆ U | ⟦N⟧ ∩ X ≠ ∅ }
⟦no N⟧ = { X ⊆ U | ⟦N⟧ ∩ X = ∅ }
⟦not every N⟧ = { X ⊆ U | ⟦N⟧ ⊆ X }
⟦Socrates⟧ = { X ⊆ U | {s} ⊆ X }

The structures that these quantifiers establish are far from random. Barwise and Cooper (1981) and Zwarts (1982, 1986) identify numerous typical algebraic objects like filters and ideals among them, and relate these structures to distributional characteristics of the corresponding *NPs*. This, then, is the real

revolution of generalized quantifiers: language users apparently recognize the algebraic characteristics of the quantifiers when constructing or parsing sentences. In this vein, it is not remarkable that Emon Bach is reported to have said that generalized quantification is the major development in linguistics in the past millennia.
Another very valuable feature of generalized quantifiers is that literally any quantifier can be captured in these terms. Here are a few more.

(266) $[\![\text{only N}]\!] = \{ X \subseteq U \mid X \subseteq [\![N]\!] \}$
$[\![\text{most N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| > |[\![N]\!] - X| \}$
$[\![\text{many N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| > k,$ for some k depending on $|[\![N]\!]|$ or $|X| \}$
$[\![\text{at most n N}]\!] = \{ X \subseteq U \mid |[\![N]\!] \cap X| < n \}$

Quantifiers are of type <<et>t> in a (t, e)-based type-logic. As extensional predicates (like nouns) are of type <et>, determiners are of type <<et><<et>t>> or, equivalently, of type <<<et><et>>t>. Although Montague (1972) characterized determiners *syncategorematically*, consequent typing identifies the relevant determiners as relations between sets. The definitions are fairly straightforward, given (265).

(267) $[\![\text{every}]\!] = \{ \langle X,Y \rangle \mid X \subseteq Y \}$
$[\![\text{some}]\!] = \{ \langle X,Y \rangle \mid Y \cap X \neq \emptyset \}$
$[\![\text{no}]\!] = \{ \langle X,Y \rangle \mid Y \cap X = \emptyset \}$
$[\![\text{not every}]\!] = \{ \langle X,Y \rangle \mid X \not\subseteq Y \}$
$[\![\text{many}]\!] = \{ \langle X,Y \rangle \mid |Y \cap X| > k \}$

This amounts to an analysis of determiners as relations between predicates, almost in the medieval sense, but now with access to all the concepts coming with modern algebra, as practiced in Zwarts (1983). Among the fruits of this approach, we have the insight that certain determiners cannot occur in a language but on the penalty of violating contingency. For example, if a language were to have a determiner Q to validate the syllogism in (268), this determiner would only produce trivially true sentences.

(268) Q *A B*, Q *A C* ⊨ Q *B C*

Thus, Q *X Y* would be true for any *X* and *Y*. As languages may be assumed not to lexicalize triviality, such a quantifier (dubbed 'Euclidean') cannot exist. For details of the proof, see Zwarts (1983).

In the same relational vein, determiners can be projected on a Pascal triangle of pairs of numbers <α, β> following (250), with α standing for the cardinality of NP-VP and β standing for the cardinality of NP ∩ VP,

(269)

<0,0>
<1,0> <0,1>
<2,0> <1,1> <0,2>
<3,0> <2,1> <1,2> <0,3>
....

Clearly then, these pairs represent the way a determiner may divide an NP with cardinality α+β. For example, the quantifier A selects the rightmost edge of the triangle, and E the leftmost. This way, one can construct a geometry of determiners and try to find out which planes do, or do not, qualify as representing (lexical) natural-language determiners (cf. Van Benthem 1984).

2.3.3 Compositionality and underspecification

Generalized quantification opens an algebraic window on semantic structures. It assumes – or rather: hypothesizes – that language entertains higher-order objects, the structure of which is understood and/or computed in action. It adds structure to meaning – it lends a face to meaning. For the theory of grammar, it poses the question to what extent Montague's idea of synchronizing syntactic and semantic structure can or must be preserved. This strict concept of so-called *compositionality* is expressed in the categorial grammar emanating from Lambek (1958), by identifying syntactic and semantic types, and by correlating logic and derivation. This strategy has been explored in-depth in Moortgat (1988), Hendriks (1993), Morrill (1994) and more work by these and other authors. The bottom line in this approach is that interpretation and derivation are parallel, and that semantic variation comes with derivational variation. In particular, variation in semantic dependency – like scope – is (partially) accounted for by syntactic processes. In this sense, compositional categorial grammar is montegovian by nature.

Delilah does not observe strict compositionality. In particular, scopal relationships – semantic dependencies between variable binding or other operators – are not computed during the derivation, but afterwards. The derivation itself delivers an underspecified storage of functional terms, assembled by sheer unification. This structure is subject to a kind of spell-out algorithm,

establishing full scopal relations in a fine-grained propositional frame. The ideology and benefits of this post-derivational after-burn will be discussed in section 2.6. What matters here is the proper balance between compositionality and semantic specificity.

It is not a secret that syntax – or form in general – underdetermines the subtleties of interpretation. It is simply not the case that scopal ambiguities are completely resolved by syntactic or prosodic marking, or that intensional embedding always comes with specific morphology. The reverse is true, too. If you have an interpretation for a sentence, not every formal aspect of a sentence carrying that interpretation can be read from it. This is established in section 2.7 on generation. In strictly compositional approaches to parsing, derivational flexibility plays the role of marking by form. If a sentence S has several meanings P, then for each P there is a different way to construe S. These different construals may even yield the same configuration: it is the procedure, the *order* of combining terms that makes the difference. There is a serious problem with this approach. Although structural ambiguity is reflected in variation of interpretation, no common representational layer for this variation can be computed: nothing is *the* (core of the) interpretation(s) of a sentence that is fully interpreted by derivation.
Of course, for interpretation we do not need a semantic object that is shared by the readings; the relationship between readings is defined logically and not syntactically. For processing purposes, however, a computable object capturing the core of a family of readings might be very welcome. This idea has become the heart of the approach to the computation of semantics that is simply dubbed *underspecification.* It rejects the idea that every aspect of sentential semantics must be computed by derivation. Bunt (2008) identifies lexical ambiguity, syntactic ambiguity, structural semantic ambiguity, imprecision and missing information as the main motives for underspecification. At the same time, he observes that the different techniques for underspecification also fulfill different needs, and that no existing technique meets all semantic requirements – a conclusion formalized in Ebert (2005) as *expressive incompleteness*. The conclusion affects approaches like Cooper storage (Cooper 1983, Frank and Reyle 1994), Hole Semantics (Bos 2002), Minimal recursion Semantics (Copestake *et al*. 2005) and Glue Semantics (Dalrymple *et al.* 1995, Crouch and Van Genabith 1999).

With Delilah we pretend to live in two worlds: the world of underspecification, and the world of full logic. Underspecification is necessary when meaning is to be computed by derivation and no information on the deriva-

tion is retained elsewhere. Full logic requires distance from the derivational processes, as many of them do not convey any information relevant to full logic. DELILAH stores all relevant derivational and structural information in an accessible manner, in the very same data structures. This information is accessible afterwards to provide full logic out of underspecification. The underspecified semantic object, generalizing over a family of readings, is just one of the yields of the derivation. *Information management* provides the algorithm spelling out that family with all the pieces of information that it needs and the availability of which would interfere with strict compositional and derivational interpretation. Under this strategy, fully-specified semantics like (242) or (245) can be derived from a well-defined derivational product, Stored Logical Form. A comparable strategy is proposed in Koller and Thater (2006). Consequently, the family of analyses ALF and FLF is protected against spurious redundancy while it is being constructed by incorporating grammatical knowledge.
Below, an outline is presented of the algorithm that takes care of the translation from SLF into full specification.

(270) APPLY(f, g) ⇒ f(g) iff *g* is of type α and *f* of type <α, β>
APPLY(f, g) ⇒ g(f) iff *f* is of type α and *g* of type <α, β>
undefined otherwise

(271) ORDEN(ListOfTerms, Template) ⇒ the list of terms Ordered such that for every a and b in ListOfTerms, a and b occur in Ordered and a precedes b *iff* according to Template, b is scope-sensitive and a is not.

(272) APPLYSTORE(Store, Body, Template) ⇒ APP(LIED)LOGI(CAL)FORM
(1) APPLYSTORE([], Body, Template) ⇒ Body;
(2) *otherwise* Store = [First|Rest] *and*
(2.1) *if* First = <S, B> *then*
(2.1.1) *if* B is not an island according to Template *then*
(2.1.1.1) APPLY(B, Body) ⇒ $Body^B$ *and*
(2.1.1.2) APPLYSTORE(SURest, $Body^B$, T) ⇒ APPLOGFORM *and*
(2.1.1.3) APPLYSTORE(S, B, Template) ⇒ B^S *and*
(2.1.1.4) APPLYSTORE([B^S|Rest], Body) ⇒ APPLOGFORM
(2.1.2) *otherwise*
(2.1.2.1) APPLYSTORE(S, B, Template) ⇒ B^S *and*
(2.1.2.2) APPLYSTORE([B^S|Rest], Body, Template) ⇒ APPLOGFORM
(2.2) *otherwise*
(2.2.1) *if* First is scope-sensitive according to Template *then*
(2.2.1.1) *if* Rest contains structures <S', B'> *then*
(2.2.1.1.1) APPLYSTORE(Rest, Body, Template) ⇒
<NStore, NBody> *and*

(2.2.1.1.2) APPLYSTORE(NStore∪[First], NBody, Template) ⇒ APPLOGFORM
(2.2.1.2) *otherwise*
(2.2.1.2.1) ORDEN(Rest∪[First], Template) ⇒ OrderedStore *and*
(2.2.1.2.2) APPLYSTORE(OrderedStore, Body, Template) ⇒ APPLOGFORM
(2.2.2) *otherwise*
(2.2.2.1) APPLY(First, Body) ⇒ B^{First} *and*
(2.2.2.2) APPLYSTORE(Rest, $Body^{First}$, Template) ⇒ APPLOGFORM

Below is an example of the algorithm's effect. Let (273) stand for the derivation of a simple transitive sentence, which marks *Det2* as scope-sensitive in a non-island. Let (274) be the Stored Logical Form produced by the derivation, with the lambda terms related to the constituents italicized in a <Store, Body> format. The application of algorithm (272) to this storage yields, in some logical formalism, the family of readings in (275).

(273) [[Det1 NP1] [Vfin [Det2 NP2]]]
(274) <<*NP1*, *Det1*> <[*NP2*] *Det2*><[ ],*V*>> *FIN*>
(275) *Det1(NP1)(Det2(NP2)(FIN(V)))*
Det2(NP2)(Det1(NP1)(FIN(V)))

In this example, *FIN* represents the term associated to finiteness, and V holds the event structure. As such, *V* contains a quantifier but the derivational template excludes this quantifier from taking scope.
In general, two scope-sensitive operators are unordered with respect to each other. So is each pair of non-scopers. The cardinality of the readings created by (272), therefore, is less than the faculty of the number of operators. This can be deduced as follows. In each semantic island (see Szabolcsi and Den Dikken 1999, Honcoop 1998) the derivation *de facto* distinguishes between scope-sensitive and scoping, independent, non-sensitive operators. A reading is any particular configuration of the members of these two classes. Let a domain contain three scope-sensitive operators {x, y, z} and two independent operators {a, b}. In general, indefinite operators are scope-sensitive, or dynamic, while definite operators are scope-insensitive, or static. The order of the members of the second class is irrelevant. A scope-sensitive operator is in the scope of an independent operator if it occurs to the right of it. The algorithm (272) ends up with the following set of readings

(276)

x y z a b	x y a z b	x y a b z	y z a x b	y z a b x
x z a y b	x z a b y	x a y z b	x a y b z	x a z b y
x a b y z	y a x z b	y a x b z	y a z b x	y a b x z
z a x y b	z a x b y	z a y b x	z a b x y	a x y z b
a x y b z	a x z b y	a y z b x	a x b y z	a y b x z
a z b x y	a b x y z			

If *s* is the number of scope-sensitive operators and *i* the number of independent scope-taking operators, the number of different readings is s^{i+1}. In this case there are 27 readings. Naive permutation would yield 5! = 120 readings. In general, we compare *n!* to k^{n-k+1} for $k < n$.

In Flat Logical Form (see (245)) different readings are clustered into one single formula of local disjunctions of small clauses. Thus, the backbone of the final representation for the sentence with the operators {x, y, z} and {a, b} will be a conjunction of small clauses. The different readings occur in the following way: every small clause in which a scope-sensitive operator *x* binds is replaced by a disjunction of small clauses each of which represents exactly one scopal dependency under which *x* can occur. In a scheme: instead of having a conjunction of small clauses (277), we have a conjunction of disjunctions of small clauses – the left arrow marks semantic dependency:

(277) x *and* y *and* z

(278) (x *or* x↤a *or* x↤a,b)
and
(y *or* y↤a *or* y↤a,b)
and
(z *or* z↤a *or* z↤a,b)

This disjunction is all that remains of the 27 readings in Delilah's semantic set-up. The whole process from Stored Logical Form to Flat Logical Form is a transition from under-specification to full specification, steered by derivational information, amounting to a kind of *logarithmic* interpretation: it decreases the order of semantic multiplicity. This implements the idea that one single semantic object should be assigned to every sentence that is lexically disambiguated, at every level of interpretation – its interpretation. Scopal ambiguity is brought back to sentence structure, and that is compositionality.

In section 2.7, finally, we will consider the use of Flat Logical Form as a program for generation. Although (278) is a conjunction of disjunctions, one can still read it as an instruction to a generator to produce a family of sentences.

2.4 INTENSIONALITY AND SEMANTIC DEPENDENCY

2.4.1 Intensionality

Phrases in a sentence may not directly contribute to determining its reference. They are not meant to refer directly to semantic objects that can be identified in the situation or state-of-affairs the sentence aims to *be about.* The crucial configuration is an embedding clause. If a sentence φ properly contains a sentence ψ, they cannot refer to the same semantic object. If φ refers to the situation-at-hand, ψ can't. In that case, ψ refers either to a sub-situation of the coordinates proposed by the main sentence, or it introduces new interpretative coordinates or indices. By now, we are referring to the latter situation as intensional embedding. It is typical for sentences or situation-type expressions because we assume that interpretation of a complex sentence amounts to its interpretation in one particular situation – that is what model theory tells us. Sentences can be about many objects, relations and moods, but about only one situation. So, the initial definition of intensionality or, better, intensional embedding is like this.

(279) *Intensional embedding*
where $[\![\psi]\!]^W$ is the interpretation of ψ with respect to situation W

A proposition φ is intensionally embedded in a context $[_s$ X φ Y] *iff*
$[\![[_s$ X φ Y] $]\!]^W$ is not a function of $[\![\varphi]\!]^W$.
In that case, φ is an intensional domain.

(280) *Extensional embedding*
A proposition φ is extensionally embedded in a context $[_s$ X φ Y] *iff*
$[\![[_s$ X φ Y] $]\!]^W = f([\![\varphi]\!]^W)$ for some *f* associated with $[_s$ X _ Y].

As was said before, Montague (1972) considers *every* embedding to be intensional, unless explicitly denunciated as such post-derivationally. This view is the same as the one expressed here since Montague's lambda terms, which were lifted to intensional functions under embedding, are all of type <α,t>, mapping onto the canonical type for propositions.

For DELILAH, we assume that every sentential embedding amounts to the creation of an explicit intensional domain. This domain, however, is not reflected at the level of types, as in Montague (1972), but by semantic operators that may or may not lead other variables into semantic dependency. Embedding

comes with the introduction of definite quantifiers over restrictions like *proposition* or *property* and scoping over or below the content of the embedded sentential or infinitival construction. The relevant distinction between properties and propositions is made within the class of infinitival complements, following Cremers (1983): all finite sentences map onto propositions, and some but not all infinitival complements do. For reasons explained there, the internal semantic construal of a property is much more restricted than the construal of a proposition, and verbs select for that construal: *proberen* 'to try' selects a property, *beloven* 'to promise' selects a proposition.

This strategy leads to the following Applied and Flat Logical Forms; in FLF, the arguments of predicates occurring in an intensional domain are marked for dependency on the proposition or property that establishes the domain and scopes over them at ALF.

(281) Jan probeert te slapen
'Jan tries to sleep'

NLF

```
exists(A) & quant(A,henk).[quant(B,some). event(B, try) &
quant(C,the).[property(C).[quant(D,some).[event(D, work) &
agent_of(D,A) & attime(D,E)]] & agent_of(B,A) & theme_of(B,C) &
attime(B,E) & tense(B,pres)]]]
```

FLF

```
exists(A+↑+henk+[]) &
event(B+↑+some+[], try) &
property(C+↓+the+[]).[
work(D+↑+some+[C]) &
event(D+↑+some+[C], work) &
agent_of(D+↑+some+[C],A+↑+henk+[]) &
attime(D+↑+some+[C],E)] &
agent_of(B+↑+some+[],A+↑+henk+[]) &
theme_of(B+↑+some+[],C+↑+the+[]) &
attime(B+↑+some+[],F) &
tense(B+↑+some+[],pres)
```

(282) Henk zei elke jongen te slaan
Henk said every boy to beat
'Henk said that he beats every boy'

ALF

```
exists(A) & quant(A,henk).[quant(B,some).[event(B, promise) &
quant(C,the).[proposition(C).[quant(D,every).[man(D) & young(D) ->
quant(E,some).[beat(E) & event(E) & agent_of(E,A) & theme_of(E,D) &
attime(E,F)]] & agent_of(B,A) & theme_of(B,C) & attime(B,F) &
tense(B,past)]]]
```

```
exists(A) & quant(A,henk).[quant(B,some).[event(B, promise) &
quant(D,every).[man(D) & young(D) -> quant(C,the).[proposition(C).[
quant(E,some).[beat(E) & event(E) & agent_of(E,A) & theme_of(E,D) &
attime(E,F)] & agent_of(B,A) & theme_of(B,C) & attime(B,F) &
tense(B,past)]]]]
```

FLF

```
exists(A+↑+henk+[]) &
event(B+↑+some+[], say) &
proposition(C+↓+the+[]).[
      (man(D+↓+every+[C]); man(D+↓+every+[] )) &
      (young(D+↓+every+[C]; young(D+↓+every+[])) &
event(E+↑+some+[C,D], beat) &
agent_of(E+↑+some+[C,D],A+↑+henk+[]) &
      (theme_of(E+↑+some+[C,D],D+↑+every+[C]);
       theme_of(E+↑+some+[C,D],D+↑+every+[]))&
attime(E+↑+some+[C,D],F)] &
agent_of(B+↑+some+[],A+↑+henk+[]) &
theme_of(B+↑+some+[],C+↑+the+[]) &
attime(B+↑+some+[],G) &
tense(B+↑+some+[],past)
```

The logical forms of (282) reflect the classical ambiguity between the *de dicto* and the *de re* readings of *(every) boy* – the genuine target of research into intensionality. There are two important questions to consider, though. First, it is not clear whether intensionality affects the interpretation of the embedded quantifier as a whole, or just of its restrictions, or both. Second, one can wonder whether the *de dicto / de re* ambiguity also affects the event of *beat*.
As for the first question: does it make sense to assume that (282) is three-way ambiguous?

(283) ... *say*...[... for every *boy(x)* ... beat(...x)
for every *boy(x)* ... *say* ...[... beat(...x)
for every x ... *say* ...[... if *boy(x)* then beat(...x)

As far as we can see, no interesting truth-functional distinction can be constructed between the second and the third readings. Yet, it is certainly reasonable to consider the 'responsibility' for the qualification *boy*; it is with the speaker or the utterer *(de re)* in the second and with the alleged 'sayer' in the third. That distinction becomes quite clear when a less neutral qualification for the restrictor is chosen:

(284) ... *say*...[... for every *traitor(x)* ... beat(...x)
for every *traitor(x)* ... *say* ...[... beat(...x)
for every x ... *say* ...[... if *traitor(x)* then beat(...x)

Now, it certainly matters whether one is quoting or summarizing the proposition made by the sayer. The second, but not the third, reading entails that those who are said to be beaten are traitors. The quantifier, however, is irrelevant for this opposition, in this case. If the quantifier were existential or indefinite, however, truth-functional considerations would make a difference independently of your modal model: *there is someone who is said to be a traitor and who is said to have been beaten* vs. *there is a traitor said to have been beaten*. In our view, the first reading amounts to the referential reading of indefinites pointed at by Fodor and Sag (1982:380): '... a referential indefinite can be used for the purpose of making an assertion *about* an individual, even though the individual in question is not identified by the speaker'.
Ruys (2006) proposes to deal with certain scope phenomena around indefinites precisely by distracting the quantificational aspect from the referential aspect. He assumes, however, an analysis of indefinites in terms of existentially quantified *choice functions* – functions from a non-empty set to members of that set – of type <<et>e>. One can reconstruct the interpretation scheme of his example (64) *Every country's security will be threatened if some building is attacked by terrorists* as follows.

(285) for all x, *if* Country(x*)*
then there is a choice function *f* and ThreatenedIfAttacked (x, *f*(Q))
for all x, there is a choice function *f* such that *if* Country(x)
then ThreatenedIfAttacked(x, *f*(Q))
there is a choice function *f* and for all x, *if* Country(x)
then ThreatenedIfAttacked(x, *f*(Q))

Clearly, the quantification over the choice function is made dependent on the universal quantifier in the first two, but not in the last reading. The second one is relevant here: the existential quantifier is in the scope of the universal quantifier but not in its restriction. Quantified choice functions, however, do not come with a labour division between restriction and nuclear scope, which is characteristic for natural-language quantifiers (see section 2.3.1). Moreover, choice functions themselves are hardly referential – the axiom of choice is non-constructive – and thus not subject to *de dicto/de re* alternations. A choice function is a way to identify members of a set. By the axiom of choice, a choice function exists for every set (Jech 1977; cf. section 2.4.2). The function is not subject to interference with other quantifiers or operators. The introduction of entities by existential closure of choice functions imposes the referential dependency on the selection of entities, but is not scoped itself. In this respect, we do not follow Ruys (2006), Winter (2001) and Reinhart (1997). Consequently, as for (285), we take the third reading to be the crucial one: the selec-

tion and identification of set members is postulated by existential closure, but the ordering of the relevant set is parameterized by other terms.
Although it is not impossible to account for three-way ambiguity in Applied Logical Form, Flat Logical Form does not qualify for this representation: all quantification is compiled into the predications. We must conclude, therefore, that this representational strategy may be defective in this sense. But this imperfection hardly disqualifies the logical form with respect to other approaches.

The second question raised by (282) was whether this intensional ambiguity would also arise in the case of the event *beat*. This question seems to be provoked by our choice to quantify over events, but even if in non-event semantics, the question who is responsible for the embedded predicate is relevant. For independent reasons to be addressed in section 2.5, events do not participate in scopal dependencies – reference to events is not influenced by their embedding. As a matter of fact, we take events to be introduced by the ultimate choice functions: assuming – philosophically rather than linguistically – that on every index of interpretation or in every situation there is always at least one event, labelling the event chosen is always possible and is not an act of reference as such. The event is identified by its roles and their performers, rather than by labelling it. Thus, in order to lift the event out of the intensional domain and still have it participate in meaningful scopal dependencies, it would be necessary to bring its full class of related propositions to the main clause level.

(286) for some event x of sort A there is an agent z and a patient y such that … INTENSIONAL OPERATOR …[…x …y … z

This would render the proposition, the theme of *say*, for example, vacuous as to its content: at best, the remaining proposition could be interpreted repeating the lifted constituent, namely that there is an event so and so. To put it boldly: it amounts to lifting the proposition out of itself.
It is difficult to see the gain here. Therefore, we assume that events are dwelling inside their intensional domain, being semantically dependent by definition. In general, we will assume that events have narrow scope in the following sense: the interpretation of no variable in a sentence ever depends on the valuation of the event term. Or, in different phrasing: the verb may be the queen of syntax, but she surely is the beggar of reference.
The domains of intensionality are introduced as a lexical feature of the embedding. Verbs taking infinitival complements – but not the standard auxiliaries – subsume the semantics of that complement under a *property* or a *proposition.* Overt complementizers come with a propositional domain. An

infinite verbal form is not intensional by definition, but a sentence with an overt complementizer is.
Intensional dependency, then, is represented by semantic dependence. If φ is to be interpreted in an intensional domain, it is marked for being dependent on the valuation of a variable of type *s* – the intensional basic type in Montague's logic. φ can be inferred only with respect to this valuation. And this is the main goal of an intensional logic: to shield certain semantic objects from unconditioned inference and from interpretation at the default indices.

2.4.2 Skolemization of dependent events

Indefinites may be referential or quantificational, specific or non-specific, with wide or narrow scope. Their proper interpretation has received strong focus in modern semantic reflection. In order to deal with their magic, Reinhart (1997) proposes to interpret them partially by choice functions, for 'assigning wide scope to existentials without moving their restriction', *i.e.* in order to avoid manipulations on logical form by quantifier raising and the like. She targets on those indefinites that come with bare numerals and that were classified by Kamp and Reyle (1993) as introducing a discourse referent.
Events are candidates for introduction by choice functions because they are indefinite – their existence can be denied consistently – and because they introduce discourse referents – they can easily be referred to by definite anaphora. Moreover, the phrases restricting the events – all sorts of predicates – hardly qualify for quantificational manipulation. It seems as if choice functions are the only but also the optimal option to handle the semantic behaviour of events. To see why, we have to project the particular nature of quantification over events on the concept of choice function.
Choice functions are assumed to exist as executors of the *axiom of choice*. In set theory, this axiom states that in every non-empty set a member can be identified (chosen), even if that set is infinite and no selection rule is given. The axiom amounts to the claim that every set is well ordered so that its members can be distinguished from each other. This is not trivial for infinite sets or for sets the construction of which is unknown. As a matter of fact, the axiom is independent of set theory but has by now become accepted as an important non-constructive but intuitively valid tool. The axiom was introduced by Zermelo in 1904 and is defined as follows in Jech (1977):

(287) For every family $\mathcal{F}$ of nonempty-sets, there exists a function *f* such that $f(S) \in S$ for each set S in $\mathcal{F}$.

The axiom reduces the identification of members of a set to the function by which they are selected. More precisely, the identification of a member of a set is made relative to the identification of a member of another set. This means that we can identify the member of a set the ordering of which is not evident in the same way as we identify the member of a set the ordering of which is less problematic and in which identification of individual members is less problematic: we can always pick the first, the second or the *n*-th member by refering to a set with an established well-ordering.

As Winter (2001) notes, choice functions can be generalized to Skolem functions: functions that identify an object with reference to independently identified other objects. Skolem functions are applied in the valuation of sentences in a predicate calculus, *e.g.* by the programming language Prolog (cf. Clocksin and Mellish 1984). A Skolem function f_φ for a formula $\varphi(v_0, ...,v_n)$ is defined by

(288) $\forall v_0 ... \forall v_{n-1}[\exists v_n.\varphi(v_0, ...v_{n-1}, v_n) \rightarrow \varphi(v_0, ...v_{n-1}, f_\varphi(v_0, ...,v_{n-1}))]$ (*cf.* Morley 1977)

The function replaces the existential quantifier and its (dependent) variable v_n by a function of type $<<e^nt>e>$ that is essentially a choice function over the set $[\![\lambda v_n.\ \varphi(v_0, ...v_{n-1}, v_n)]\!]$ with respect to some assignment of values to each v_i, $0 \leq i \leq n\text{-}1$. The Skolem function identifies a dependent object by using parameters from other sets (valuations of variables into those sets) and by getting rid of the related (indefinite) quantifier. Thus, it turns the object scope-independent, frees it from quantification and makes it addressable; and it does so by leaning on the axiom of choice. Basically, the Skolem function is the procedural counterpart of scopal dependency.

We need Skolem functions to deal with events, for at least three reasons. Firstly, events are existential and semantically dependent, intuitively. The use of a predicate asserts the existence of an event or state but the event is identified by refering to the arguments and adjuncts, like the specifications of time and place. From the point of view of modelling, it does not make sense to claim that there is an event of a certain sort without refering to its thematic, temporal and, possibly, spatial correlates (cf. Verkuyl 1993, ch. 13). There are no events outside the gates of Eden. Consequently, Skolemization of the interpretation of events reflects their real but secondary referentiality.

Secondly, the algebraic structure of the set of events and states is such that identification of events by simple quantification is not enough. Events are mass-like, rather than discrete and countable (cf. Bach 1986). We have to guarantee the selection and identification of an event and its labelling with predicates by more fundamental means, like contextual fixation.

Thirdly, the algebra of events is such that manipulating scope does not buy us much – except in the case of truly collective events or states like the *kolchoz collectivity* or *the angles of a triangle are 180 degrees* – the predicative and semantic structure of which is difficult anyway. Except in the *kolchoz* cases, one event or state for all can always be split into sub-events or sub-states per individual, just as events per individual easily aggregate to just one; since events combine in many ways, their scope is referentially not distinctive.

The three reasons amount to just one from an ontological point of view: events are entities that differ essentially – *i.e.* algebraically – from other entities living in the realm of natural-language semantics, and therefore deserve some logical respect.

In the following excursions, we will assume that every predicate comes with a Skolem function, identifying a state or event by refering to its parameters, without assigning scope to its value. Logically, however, the interpretation of predicates is supposed to be existentially quantified, though the quantifier is redundant in the valuation. In the sections to come, events and states are marked as such. They do not enter into exciting interactions with other operators, because their valuation is handled by Skolem functions. In Flat Logical Form, however, quantifiers do not return as scopal objects anyway.

2.5 EVENTS AND STATES: REIFICATION OF PREDICATION

2.5.1 Reference to events

In natural language, predicates come in some variety: nouns, verbs, adjectives, adverbs, prepositions may all introduce properties of entities introduced by other phrases in a sentence. For reasons of inference, all these predications have to be made explicit. *John put the wood in the garden* entails, among others, that after a certain moment the wood is in the garden, and this predication must be made explicit in the sentence's representation in order to make that inference viable. In the same vein, something is predicated to be wood and to have been put in the garden. Is there also something of which the sentence predicates that it is putting-wood-in-the-garden, or putting-something-by-John, or putting-wood-in-the-garden-by-John? Yes, there is. In both sequences of (289) the italicized phrase in the second sentence refers

to a "something" that is not addressed or introduced by any phrase in the sentence other than the one headed by *put.*

(289) John put the wood in the garden. But Mary did not approve of *it*.
John put the wood in the garden. He did not do *it* alone, though.
John put the wood in the garden. He was all alone*, then*.

Davidson (1967) was the first to propagate that the predicative "something" of action verbs should be addressable in natural-language semantics. The strategy of assigning a referential backbone to real predicates is named after him: '(neo-)Davidsonian'. Here is an example from Reckman (2009).

(290) I flew my spaceship to the Morning star
∃x. Flew(i, my_spaceship, x) & to(morningstar, x)

Here the predicate *flew* is expressed as a three-place relation, one of the arguments of which is a bound variable that occurs as an argument in a typical adverbial modification of the predicate.

At least three questions immediately arise: (1) what is the nature of the predicative entity? (2) should or can *all* natural-language predicates be *reified* this way and (3) how does it get quantified over?

The first question can be answered as: that is a matter of philosophical taste. If you prefer a sharp ontology, you can introduce a particular type or sort for the predicative entity: apart from entities and/or propositions, there are properties representing a 'basic' type, as has been proposed by Turner (1989). Alternatively, you have the predicative entity's nature defined by some predicate, e.g. the predicate itself. This is more or less the strategy undertaken in Delilah. But there we introduce reified predicates by a metapredicate like *event*, relating the predicative thing to a name and introducing explicit role labels for the predicate's arguments. Delilah's version of (290) looks like this:

(291) I flew my spaceship to the Morning star
∃x. Event(x, flew) & Agent_of(x, I) & Theme_of(x, My_spaceship) &
To(x, MorningStar)

The main advantage of this way of representing *flew* is that it makes explicit, and thus inferable, that I was actively engaged in something, and that my spaceship was in a different way engaged in the same thing, no matter what we call it. In other words, the sentence establishes a semantic term in which

the phrases act as values to attributes. All phrases are born equal, so to speak, but it is the main verb here that comes with the thematic frame: it still is the sentence's queen. The headness of the verbal predicate can only be read from the derivation, though. In the lexicon, the verb is endowed with a thematic frame that unfolds in the transition from Stored to Applied and Flat Logical Form. For example, a lexical template for the finite verb *slaapt* 'sleeps' at SLF not only has a variable for the subject term in store, but also a genuine quantifier introducing an event; abstracting away from bookkeeping, the SLF in the lexicon looks like this.

(292) *SLF of* slaapt *'sleeps'*

```
< [store0 SubjectTerm, λY.some(U).[event(U, sleep) & Y(U)] 0erots],
λA.λI.λV.agent_of(V, A) & attime(V, I) & tense(V, pres) >
```

Basically, the event is 'quantified in' the finite or infinite structure of the verb, just like, at some moment in a derivation, the subject or any other argument is. The verb imposes its argument structure on the whole sentence, including its own semantic impact. In the ultimate representation, after derivation, grammatical headness can no longer be traced, as it is irrelevant to meaning and may obscure logical operations. In any case, this explicit way of handling events allows for a representation language that is independent of syntactic realization – an interesting feature with respect to generation (see section 2.7).

As for the typological status of the event variable – *U* and *V* in (292) – we take their type to be *e*, essentially. Assigning an elementary type to them implies that in the semantic representation of sentences we cannot exploit the algebraic structure and the partial ordering of the set of events. Event variables are sorted by the system-predicate *event*, and identifiable as such. As far as we can see, there is no technical objection to typing them differently, however. Montague's *s* for situational indices may qualify. In that case, the intuitive reading of an event structure would be that there is a particular situation to be labelled with a certain verbal concept. *John sings* is read as: there is a particular situation that we label as a *singing* event in which John acts as an agent. There is, however, a linguistic reason not to resort to any different type. All languages are able to put predicative concepts in nominal jackets. These *nominalizations* may or may not have a distribution which slightly differs from other nominal constituents, but they certainly are part of a nominal paradigm. Reckman (2009) and Reckman and Cremers (2007) present an analysis of really predicative nominalizations in the present framework, which shows that it is possible to give full credit to the predicative semantics

of nominalizations while maintaining normal nominal syntax, in particular, quantification and adnominal modification. The sheer concept of nominalizations asks for a syntactically homogeneous treatment. In that respect, there is no reason to switch types; at the end of the day, verbal and nominal predicates converge. This view is also expressed by Khalaily (1997), albeit in a completely different framework.

The second question with respect to reification is whether it applies to all predicates. It is clear that *event* is not the right predicate to classify all verbal predicates. We need *states* of some kind to refer to predications like *lie* and *be at*, following Reckman (2009: ch. 3). There it is also argued that proper nouns and adjectives, at least intersective ones, can be modelled as introducing states. The linguistic evidence is less convincing here. But in Dutch nominal and adjectival predicates can be referred to with the very same set of pronouns that is used for reference to propositions, events and verbal states, and (neutral) noun phrases.

(293) Karel lijkt nogal zenuwachtig de laatste weken. *Dat* was hij eerder niet
Karel seems rather nervous the last weeks. That *was he before not.*
'Karel seems rather nervous these weeks. He was not before'

(294) Hij is dokter. Iedereen kan *dat* hier worden.
He is doctor. Everybody can that *here become.*
'He is a doctor. Everybody can become one here.'

This anaphorical reference is to nominal and adjectival phrases in predicative positions. But there is no evidence that these predicates are different from the ones used in nominal or adnominal positions. Consequently, we take all nouns and adjectives to refer to states, following Parsons (2000). So, the first reading of *Every man seeks a unicorn* would be (295) rather than (242)a.

(295)
```
quant(B,every).[[quant(P, some).[state(P, man) &
theme_of(P,B) ] ⟹ quant(C,some).[event(C, try) &
quant(D,the).[property(D).[quant(A,some).
[quant(Q,some).[state(Q, unicorn) & theme_of(Q,A)] &
quant(E,some).[event(E, beat) & agent_of(E,B) &
theme_of(E,A) & attime(E,F)]]] & agent_of(C,B) &
theme_of(C,D) & attime(C,F) & tense(C,pres)]]]
```

These states introduced by nouns and adjectives are not specified for time. By default, their extension in time covers the temporal extension of the verbal predicate to which their argument is thematically connected. The reasoning here is that, in Dutch, (296) is a simple claim about the present prime-minis-

ter's former career, whereas (297) seems to imply that the then prime-minister had to go to school (again?).

(296) De minister-president is in Zeeland naar school gegaan
the minister-president has <pres+perf> *in Zeeland to school gone*
(297) De minister-president ging in Zeeland naar school
the minister-president went <past> *in Zeeland to school*

This balance in temporal extension can be made explicit, but because it can also be stated as a language default, we leave it out.

With respect to quantification over events and states: in our approach, all predicative objects are bound by a dedicated existential quantifier. This quantifier, although scope-sensitive as such, does not participate in scopal ambiguities, in that it does not take wide scope, for reasons explained above: their interpretation and identification is handled by Skolem functions, providing a contextually parametrized referent from the set of events (see section 2.4.2). Thus, the existential quantifier is treated as a kind of dependent definite description. So we have to explain why events and states are bound by existential operators rather than by the iota-operator or the like. The reason is this. Even if we typologically treat events and states as normal objects, there is no reason to believe that they are unique in their sort at a certain index, as would be the implication of marking them definite, in the sense of being semantically independent and fixed. All that is claimed by the identificational use of a nominal predicate (*the man, a man, no men*) is that some way-of-being of an assumed entity can be labelled with a certain concept. It does not claim that this labelling amounts to rigid naming of the way-of-being, that no other labelling of that particular way-of-being is possible or viable, or that the labelling is axiomatic. The concepts may be rigid designators, as Kripke (1972) argues, but the ways-of-being are certainly not rigidly identified. Moreover, the Skolemization of nominal states has the advantage that the state's identification is made to be dependent on the referent of the predication. In that vein, a sentence like (298) has a quantified logical form like (299) and a skolemized representation like (300). In the latter representation, neither *man* nor *walk* introduces a quantifier. The state is identified by a Skolem function associated with the nominal predicate *man* and parametrized for the valuation of *x*, and the event is identified by a Skolem function associated with the verbal predicate *walk,* which is parametrized for the values of the individual variable *x* and the temporal object *t*.

(298) The man walks
(299) ιx. ∃t. time(t, present) & ∃s. state(s, man) & ∃e. event(e, walk) & theme(s,x) & agent(e, x) & attime(e, t)
(300) ιx. state(f_{man}(x), man) & event(g_{walk}(x, t), walk) & theme(f_{man}(x), x) & agent(g_{walk}(x,t), x) & attime(g_{walk}(x,t), t)

The dependency that comes with Skolemization is comparable to the way in which even extensional adjectives denote properties that are partly determined by the noun class over which they are predicated: the *red* in *this apple is red* denotes a colouring pattern different from the *red* in *this flag is red*. Here too, the way of being red is determined by the choice of the referent, whereas the predication as such is not ambiguous.
Adopting simple existential quantification for states and events, it is certainly tempting to exploit it for marking other semantic phenomena. What immediately comes to mind is the infamous ambiguity between collective and distributive readings for plural and plural-like predications. We could decide to interpret (301) as either one of (302)-(304).

(301) All women are singing
(302) ∀x. ∃y. state(y, woman) & theme(y, x) ⇒ ∃z. event(z, sing) & agent(z, x) & time(z, pres)
(303) ∃y. state(y, woman) & ∀x. theme(y, x) ⇒ ∃z. event(z, sing) & agent(z, x) & time(z, pres)
(304) ∃y. state(y, woman) & ∃z. event(z, sing) & time(z, pres) & ∀x. theme(y, x) ⇒ agent(z, x)

A few aspects are noteworthy. No good interpretation is available for the scopal differences between the universal quantifier and the state quantifier. It seems undecidable whether the state quantifier is semantically independent or not – even when we leave a restriction for the universal quantifier. It is impossible to design a situation in which (302) is true and (303) false. And of course that is due to the intuitive durativity and the non-eventuality of states. We cannot force non-identity of states; they are mass, rather than countable. It is considerably easier to design different models for (303) and (304). The first sentence may be true for an opera agent who observes that each of her divas is actually performing some role somewhere. In that situation, (304) may not apply. It seems, then, as if the possible collectivity of an event can be represented by scopal independency of the event. This presumes, however, that the existential quantification is dynamic, in the sense that its power to bind across sentences is influenced by its referentiality. That is not the case. The anaphorical relation in (305) is viable whether we interpret the first sentence collectively or

distributively. But the anaphor in (306) is only compatible with a wide-scope reading of the existential quantifier.

(305) All students are *having a good time*. I want *it* too.
(306) All students admire *an associate professor*. I know *her* well.

This points to a general phenomenon: propositions, properties, events and states refer *sui generis*. Their behaviour with respect to number or, more generally, their algebra differs essentially, as was argued in *e.g.* Cremers (1993a, 2002). Moreover, referentiality is maintained, even in the presence of negation.

(307) You did not hit Bill. I saw *that*.

In this example, the pronoun refers to a non-event, say a state: the state of not being an event of hitting, of 'you' not being the agent in that event and of Bill not being its theme. It looks as if, as Asher (1993: 215) puts it, '...negation always transforms an event inside its scope into a state outside its scope'. As a matter of fact, the interpretation that accounts for the anaphor must be

(308) the index in space and time for which there is no event of John hitting Bill.

This is the state that no inviting event occurred or John was not its agent or Bill was not its theme. This observation, however, also confirms the position in Asher (1993: ch. 1) that events are not closed under negation: the negation of an event is not an event. In our semantics, the representation of the first clause in (307) would be like (309). After Skolemization, this renders (310)a, which in turn is equivalent to the disjunction (310)b.

(309) $\neg\exists$e. event(e, hit) & agent(e, j) & theme(e, b)
(310) a. $\neg$ (event(f_{hit}(j,b), hit) & agent(f_{hit}(j,b), j) & theme(f_{hit}(j,b), b))
b. $\neg$ event(f_{hit}(j,b), hit) $\vee \neg$agent(f_{hit}(j,b), j) $\vee \neg$ theme(f_{hit}(j,b), b)

The latter representation amounts to the following proposition: whatever the choice function f_{hit} returns as a value on arguments *j* and *b* is not an invitation event or *j* is not its agent or *b* is not its theme. The first disjunct may seem weird, but it is not. A set of events and states induced by the predicate *hit* may contain all kinds of indices, which have in common only the fact that they are selected by application of the concept *hit*. The set may contain the spatio-temporal indices at which no event of invitation occurred: the state of nobody inviting anybody. This index is the value of $f_{hit}(j,b)$ if neither *j* nor *b* was involved in inviting events. This index does not identify an event, and that

is exactly what the first disjunct says: the *hit*-induced relationship between j and b is not an event. The relation of *j* not inviting *b* itself is real and addressable, however. Thus, Skolemization offers an alternative to Asher (1993: 217), who concludes that events force us to '...construe the aspectual and transformational character of negation as affecting a level of semantic interpretation other than denotations'.

The simple fact that predicates always refer, even when absorbing negation, elicits the hypothesis that they are introduced by *Skolem* functions, rather than by scope-sensitive quantifiers. Like choice functions, Skolem functions exist by the axiom of choice. Their existence does not need to be reclaimed in any particular representation. They can be considered as lexically assigned attributes of predicates. Consequently, instead of the representations in (311) and (312) with higher order quantifiers over choice functions as in Winter (2001), DELILAH would produce the Applied Logical forms (313), where e^i and s^i stand for the result of applying an indexed Skolem functions and *i* marks the temporal argument of the function.

(311) All men sing
∀x. ∃g. state(g(x,i), man) & theme_of(g(x,i), x) ⇒ ∃f. event(f(x,i), sing) & agent_of(f(x,i), x) & attime(f(x,i), i).

(312) All students admire an associate professor
∀x. ∃g. state(g(x,i), student) & theme_of(g(x,i), x) ⇒
∃y. ∃f. ∃h. state(h(y,i), assocprof) & theme_of(h(y,i), y) & event(f(x,i), admire) & agent_of(f(x,i), x) & theme_of(f(x,i), y) & attime(f(x,i), i)

(313) a. quant(every, x).state(s^i, man) & theme_of(s^i, x) ⇒
event(e^j, sing) & agent_of(e^i, x) & attime(e^i, i)

b. quant(every, x). state(s^i_1, student) & theme_of(s^i, x) ⇒
quant(some, y). state(s^i_2, assocprof) & theme_of(s^i_2,y) &
event(e^i,admire) & agent_of(e^i, x) & theme_of(e^i, y) &
attime(e^i, i)

In Flat Logical Form, no quantification of the event and state variables will be marked either.

Finally, there is a decisive reason to adopt a neo-Davidsonian event structure. *Extended lexical units* or *constructions* may be such that the optional modification of elements that do not contribute to the meaning of the predicate directly is to be interpreted as a modification at the semantic top level. For example, in the constructions of the type *honger hebben* 'to be hungry', the mass noun *honger* may carry a determiner that conveys the mood of the state: *geen honger hebben* (lit: no hunger have) means 'not to be hungry' and *weinig*

honger hebben (lit: little hunger have) means 'to be a little hungry'. The nominal modification is by-and-large free, but always interpretable at the predicative level. In the semantic set-up of the extended lexical unit *honger hebben,* this transfer of essential semantic information can only be established if at the top-level predication enough 'hooks' are available. Quantification over events and states offers these hooks. It reduces the difference between *nouniness* and *verbiness* (cf. Wetzer 1995) at the semantic level – for syntax this reduction was already pleaded for by Khalaily (1997). Without this cross-categorial equalization, systematic or even finite treatment of semi-transparent extended lexical units does not seem possible. In this respect, neo-Davidsonian event structure leads to normalization and homogenization of logical form, as pointed out by Reckman (2009). The question will be discussed further in section 2.6.

2.5.2 Pluractionality

Modern linguistics has developed an impressive tradition of applying model theory to semantics. With respect to events, this tradition has two foci: the theory of groups and related objects, and the investigation into the nature of events and states. The first domain originates from Link (1983) and enriches the model with entities built out of other entities. The idea is that these entities exist outside language and are introduced – implicitly – into natural languages by processes like pluralization and interfere with interpretation. The other domain is strongly connected to the study of aspect and *Aktionsart* and deals with the way states and events unfold in time and space, and the reflection of these model properties in the way we refer to them in natural-language propositions. There, we have Vendler's classification and, for example, Verkuyl's work on aspect (cf. Verkuyl 1993) and work on the algebra of events, like Bach (1986) and Krifka (1990). The two foci converge in the treatment of what is by now called *pluractionality*: the study of the interaction between quantification and predication, addressing issues like distributivity and collectivity, iterativity and other aspects of simultaneous multiple predication.
Here we want to outline a framework for dealing with pluractionality in a computational way, *i.e.* relating to the construal of logical form and to inference. This framework leans heavily on our conviction that the structure of the world is not reflected in the structures of languages in any interesting way.
First, we do not go along with the idea that, apart from atomic entities, the model for the interpretation of natural language contains *super-atoms* like Linkean groups, nor with its consequence that the semantics of natural languages gives

rise to group-forming operators. As explained in Cremers (1993a) and Cremers (2002), we believe that the group concept is both too strong and too weak: it multiplies entities beyond necessity and it does not properly restrict inferences. The conjunction of two proper names in the following sentence does not refer to a group, since, if it did, the sentence would entail nonsensically that Hijzelendoorn was involved in developing quantum physics.

(314) Niels Bohr and Maarten Hijzelendoorn developed quantum physics.

The sentence is distinctly false because no group {Bohr, Hijzelendoorn} exists and the latter certainly did not contribute in any interesting way to quantum physics in its seminal years or thereafter.
As far as we can see, groups may be ontologically relevant but superfluous in the model theory of natural languages. Moreover, we believe that number is nothing but an agreement feature of natural languages, with as little semantic relevance – relevance for the model and for inference – as case or gender. The tasks that groups fulfil in formal semantics – to account for collective and cumulative readings – can easily be conveyed to decomposition of predicates – a device needed anyway. This will be illustrated below.
Secondly, we do not believe that the phrasing of natural language reflects structural properties of processes in time and space. That is, we do not believe that any constituent in natural language can be held responsible for expressing properties of processes. To put it more bluntly: the algebra of events is not involved in any interesting way in the semantics of natural language. No feature of the semantics of natural-language propositions hinges on the intrinsic nature of an event or state. Natural-language semantics is not about the question which sequence of moves counts as a minimal walking event in a non-metaphorical interpretation of an occurrence of the verb *walk* in

(315) The duck was walking on the water.

Whether this sentence is true or not depends only on whether we accept the description of whatever the duck was doing as an act of walking. If not, we may not even know why we disagree, and still enjoy fruitful communication. It makes no sense to encode a definition of walking as a process of moving and muscling in the semantics of a sentence. Thus, we simply refer to the events predicated, without providing a model-theoretical definition of the event. That definition is for the encyclopedia, not for the lexicon.
Thirdly, we try to tackle problems with pluractionality by adopting an idea by Schein (1993): *essential separation*. In this view, different thematic relations

in a sentence select different events, or every eventual predicate introduces a sequence of events and states. In a sentence like

(316) The boys carried the piano upstairs

for every boy there is an event in which he is the agent and there is a state of being upstairs with a piano as a theme, for example:

(317) For every boy x, x is the agent of some event of carrying and there is a state of being upstairs for some piano.

The analysis that some predicates – *e.g.* telic ones – introduce sequences of events and states is also developed in Arsenijevic (2006), referring to Verkuyl (1972) and Ramchand (2002), among others. Arsenijevic has the following decomposition for telic VPs with three thematic arguments:

(318) [$_{vp}$ [$_{\text{INITIATING SUBEVENT}}$ Participant1 [*lexical_predicate* Participant2]
[(concat) [$_{\text{RESULTING SUBEVENT}}$ Participant2 [*lexical_predicate* Participant 3]]]

As an example, the 'template' [$_{vp}$ *John put the bag in the closet*] is analyzed as

(319) [$_{vp}$ [$_{\text{INITIATING SUBEVENT}}$ John [*add_to/control/contact/place* the bag]
[(concat) [$_{\text{RESULTING SUBEVENT}}$ the bag [*place_in* the closet]]]

In his view, this is a syntactical structure where each sub-event comes with a fully-fledged sentence. The sentences are essentially concatenated, for example by temporal ordering and/or by a causative linking of the type *and therefore*, *and thus*, or *and by that*. The nature of this link is given by the VP. The lexical templates in the analysis are also induced by, or derived from, the VP predicate; the first one, however, can be identified as the *add_to* feature of Verkuyl (1993), and the notion of *contribution* in Cremers (1993a: ch. 2). To implement analysis (318) in a syntactically less demanding framework, we assume that every predicate comes with an internal aspectual structure, by means of decomposition, which for every argument specifies one type of event. We assume that *every* non-intransitive eventive predicate can be decomposed. Consequently, we take the simple transitive sentence (320) to have an eventive structure as in (321):

(320) The boys have painted a car
(321) For every boy x, x is the agent of some *painting*-related event and (by the sum of these actions) some car is in the state of *having been painted*.

Note that in this structure an algebraic relationship between the two events is lacking. In particular, we do not claim that the agentive event is related to a painting event in any interesting algebraic sense. It is neither required to be a sub-event of painting nor is it linked in space or time to any painting event. The *concatenation* of (319) is *cumulative causation*: the sum of the painting related events *leads to* the resultive state in a way that is not specified in the sentence, let alone in the lexicon.

The quantifier on the subject determines the nature of the accumulation. If the quantifier is non-referential, the painting-related event is equivalent to painting and any individual agentive action *leads to* the state of a car having been painted. This is distribution of the subject over the predicate. If the quantifier is referential, no such equivalence is claimed. This leads to the *collective* reading. As a matter of fact, both readings instantiate the cumulative reading of Scha (1981) for sentences like

(322) Nineteen software companies own seven mainframes

where the number of mainframes owned is the sum of the mainframes of all companies. In our approach, the subject's atoms are *always* involved in, and contribute to, what is predicated of the object, and the cumulation's arithmetic – the nature of the sum to be made – is determined by the subject quantifier. Numerals like those in (322) allow for simple addition of the subject atoms' contributions, where each atom is mapped on some rational between 0 and 1, reflecting the impact of the contribution to the Boolean value 1 for the predication over the object. The sentence is true if the sum of the atomic contributions is 1. For real quantifiers, the sum is simple: every atomic contribution is taken to be 1. Real quantifiers can not induce the simple addition that underlies (322).

This approach makes a few predictions. Most prominently, it predicts that only events can introduce ambiguity into the cumulation, since only events can decompose and concatenate in the way assumed above. Nouns, for example, can only be distributive. Furthermore, it predicts that only multi-argument predicates can introduce an opposition between distribution and collectivity. Intransitive verbs are like nouns, in allowing distribution only. The sentence *The men sing* is not ambiguous but univocally distributive. The English verb *meet* is the exception here, which explains why in almost any other language *meet* is transitive, or at least reciprocal. Our analysis dooms *meet* to be an irregular intransitive.

In summary, our format for the eventual structure of a transitive sentence is a decomposition of the verbal predicate into *essential separation* (Schein

1993), with the separated events being related cumulatively according to the quantifier of the 'external argument'.

(323) $Q^1x.\ S^1(x)\ \exists e_1\ P'(e_1)\ \&\ agent(e_1,x)\ \&^{cumulativecausation(Q1,\ P,\ P')}\ Q^2y.\ S^2(y)\ \exists e_2\ P(e_2)\ \&\ theme(e_2, y)$

In this set-up, the connection between the two separated events – represented here as *conjunction-plus* – carries the semantic load of the main verb: a dedicated form of causation, reduced to some arithmetical operation on quantified causes. The connection therefore depends both on the main predicate and on the subject quantifier: the predicate determines the space for what count as causes for – or, more generally, contributions to – its being brought about, and the quantifier determines the mode of addition.
Clearly, this separation can be introduced lexically and the cumulative causation can also be deduced compositionally. That is: a transitive predicate will lexically be associated with a lambda term introducing essential separation and cumulative causation, with a slot for the arithmetical constraint to be borrowed from the subject. To make this work, we have to assume that every predicate P comes with an encyclopaedically determined set of related activities RA^P, which certainly does not only contain predications derived from P 'algebraically'. Moreover, the nature of the relationship between the subject-oriented event and the object-oriented state can be spelled out as a specific instance of a general predicate λQλPλRλxλy.*contributes*(Q,P(x),R(y)) relating a quantifier *Q* and predicate-related events *x* and *y* in order to state that an event *x* of predicate *P* contributes to a degree determined by *Q* – the agentive quantifier – and *R* to bringing about a state *y* of predicate R. The *contributes-relation* is introduced lexically as part of a verb, and indexed for the nature of Q. In that case, the propositional frame (323) turns into (324), and a verb like *carry* comes with lexical specification (325):

(324) $Q^1x.\ S^1(x)\ \exists e_1.P'(e_1)\ \&\ agent(e_1,x)\ \&\ Q^2y.\ S^2(y)\ \exists e_2.P(e_2)\ \&$
$theme(e_2, y)\ \&\ contributes^{Q1}(P'(e1), P(e2))$

(325) $\lambda Q\lambda R.Q(\lambda x.\exists e1.\exists P' \in RA^{carry}\ P'(e1)\ \&\ agent(e1, x)\ \&\ P(\lambda y.\ \exists e2.\ carry(e2)\ \&$
$theme(e2, y)\ \&\ contributes^{Q1}(P'(e1), carry(e2))$

If Q is a real 'distributive' quantifier, P' is identified with *carry* and the related events are identified too: every agentive contribution is set to 1 with respect to bringing about the thematic state. This is equivalent to the merge of P' and P, resulting in normal distributivity. If Q is referential, the function specifies that the P'-event(s) contribute to some non-zero degree possibly lower than 1. It is worth noting that the relative scope of the two argument quantifiers is not

relevant to pluractionality under neo-Davidsonian semantics. The 'number' of carrying events is not related to the specificity of the objects if both quantifiers are scope-sensitive, since the quantification over the event is bound to have narrow scope, as explained before.
This analysis diverges from the approach in Cremers (1993a: ch. 2), in that no adding up of the contributions to 1 is required under referential quantification. In the new analysis, the sentence

(326) The boys carried the piano upstairs

states that each of the boys contributed to the carrying and that the piano ended up upstairs; it does not claim that the contributions of the boys exclude other essential contributions to bringing about the resultative state. The sentence does not exhaust the set of contributors, and so it does not entail

(327) No one else contributed to carrying the piano upstairs.

This lack of exhaustion correlates to the referential nature of the quantifier that is responsible for the *possibly-less-than-one* contribution. The sentence *John and Pete sang a song* does not exhaust the list of song singers: why would *John and Pete built a house* exhaust the list of contributors to the construction of the house, while not exhausting the list of those who built a house?

The *essential separation* approach unfolded above is not yet implemented in DELILAH, in order to not complicate the logical form beyond necessity. Instead, we represent the structure (323) 'simply' in the format

(328) $Q^1x.\ S^1(x)\ Q^2y.\ S^2(y)\ \exists e\ P(e)\ \&\ agent(e, x)\ \&\ theme(e, y)$

The translation from (328) into (324) is straightforward, once the predicate *P* is offered in a decomposed manner.

2.6 EXPLOITING LOGICAL FORM FOR PARSING

2.6.1 Logical Form, Grammar and Computation

We take logical form to be that mode of linguistic analysis in which lexical concepts, inferential semantics and information structure interact. The required analysis is formal, in the sense that it should account for the inter-subjective and – thus – systematic aspects of sentential and lexical meaning. In particular, logical form is a level of grammatical representation at which semantic consequences can be computed – a view already expressed by Higginbotham (1985). Logical form is therefore one of the ultimate targets of linguistic analysis and the representation of what makes natural language a unique module of human cognition. Typically, logical form emanates from deep processing; there are no shallow routes to semantic precision, flexibility and coverage.

The claim that logical form is formal by necessity does not imply that it is bound to comply with first-order predicate logic. Predicate logic does not entertain a privileged relation to semantic interpretation: the set of meanings of natural language is neither a subset nor a superset of the well-formed and interpretable propositions of predicate logic or their complements. It may be a helpful tool in describing certain aspects of the relations between concepts and operators in natural language; in our view, however, it is neither the target nor the anchor of natural-language semantics. A logical form is a representation for which some adequate notion of semantic consequence (entailment) can be defined.

Overlooking modern grammar, it makes sense to state that the relationship between logical form and syntactical structure defines the arena. That is not a trivial observation: though linguistics ranks among the older sciences in the world, it took millennia for the proper balance between form and interpretation to be questioned from a grammatical perspective. In recent times, Bertrand Russell (1949) has argued that logical form is not even homomorphic to syntactical structure. Generative grammar split on the question of with which aspects of meaning grammar could afford to deal (Seuren 1998: ch. 7). In the same period, Montague (1972) defined logical form by compositional interpretation, but correlated it to pseudo-syntax. Pursuing this semantic perspective, categorial grammar started taking the syntax seriously and producing interpretations by derivation, deduction or unification (Ades and Steedman 1982,

Moortgat 1988, Morrill 1994 and much other work in this spirit). Categorial grammar's strong generative capacity, however, challenges linguistic relevance (see chapter 1). At the same time, modern grammar theories appear to converge at some formal link between structure and interpretation. The compositional nature of the main semantic configurations is by now undisputed, after the generative self-reflection of Heim and Kratzer (1998) – that is, if we agree on the lexicon being the only source of semantic wisdom, and on meaning being computable. For a deeper assessment of the relationship between syntax and semantics in grammar, however, see the final chapter.

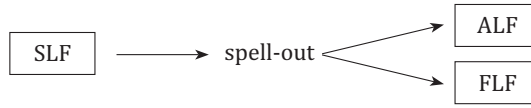
In computational linguistics, logical form is not a very common module of language processing systems. Approaches that avoid incorporating explicit grammar deliberately refrain from logical form, *a fortiori*. Many scholars working on such systems seem to share the scepsis about the computability of meaning that some theoretical linguists entertain. These 'agnostic' strategies govern the field in our days. As a matter of fact, for each language that is targeted by computational efforts, the number of systems doing semantic analysis is quite restricted. And among these, computation of logical form for inference is rare. Notorious icons here are the LINGO enterprise (Copestake and Flickinger 2000), the grammars involved in the Verbmobil project (Wahlster 2000), the PARC XLE parser (Maxwell and Kaplan 1993) and DRT-related approaches, like Bos (2001) and Bos (2004). None of these deal with Dutch.
As for Dutch: memory-based language learning as established in Daelemans and Van den Bosch (2005), does not seem hitherto to have targeted propositional interpretation or any other semantic level. The problem for these learning approaches is twofold: there is hardly any tagging from which propositional semantics can be induced – there is nothing to be stored or learned – and if it existed, the scarceness of data might be overwhelming with regard to the subtlety that propositional interpretation for inference calls for. Robust and wide-coverage parsers like Alpino (Bouma *et al.* 2001, http://www.let.rug.nl/~vannoord/alp/Alpino/) and the older Amazon (http://lands.let.ru.nl/amazon/) do not aim at full logical semantic representations. Inference – the core business of computational semantics according to Bos (2004) – is not addressed.

2.6.2 Logical form in Delilah

Delilah computes three related levels of logical form: Stored Logical Form or SLF, Applied logical form or ALF and Flat Logical Form or FLF (section 2.2).

SLF is *derivational*: it is constructed by applying rules of grammar, which induces unification of complex symbols. It is *compositional* in that it expresses and reflects important aspects of the grammatical structure, and by being built derivationally. Moreover, it is *underspecified*. Important features of the interpretation are not made explicit at SLF. SLF underlies the construal of both ALF and FLF. Specialized algorithms translate SLF into ALF and FLF post-derivationally. Consequently, ALF and FLF are no longer fully compositional in the strict sense: not every aspect of these logical forms is functionally related to the grammatical structure. ALF and FLF both specify all definable features of the interpretation. They differ in the ways of specifying semantic properties. In ALF, matters of scope and semantic dependency are encoded globally and implicitly, as in standard predicate logic. In FLF, scope and semantic dependency are compiled out and made explicit at local levels. FLF acts as a semantic index on ALF.
Therefore, in DELILAH, fully-specified logical form is derived from underspecified SLF by an algorithm that section 2.6.4 argues is exploiting grammatical knowledge.
Alshawi *et al.* (1991) explicitly address the question of why one should entertain multiple levels of logical form. They claim the Quasi Logical Form of the Core Language Engine to be a single level of semantic representation, with additional procedures of resolution providing values for free variables. In that sense, our three logical forms also provide one single representation that consists of several layers. We claim that these different layers can have distinct functions. We certainly do not claim that the logical forms arise from different semantic modes. There is just one process of logical form construal, unfolding at different stages. Here is the picture.

(329) Derivation



2.6.3 Stored Logical Form

2.6.3.1 Construal

Stored Logical Form is constructed independently of derivation. The structure of SLF is defined lexically. It is instantiated during the derivation, but particular derivational moves or procedures do not influence that instantiation. Neither syntax nor control can modify SLF. As a consequence, syntactically different sentences can have the same SLF. Different derivations of the same

sentence will have equal SLFs by definition. As an example: the following sentences will have equal SLFs in a DELILAH-approach.

(330) John was beaten by nobody.
Nobody beats John.
John, nobody beats.
It is John who nobody beats.
It is John who is beaten by nobody.

Therefore, DELILAH realizes the old idea of deep structure. There is a distinctive level of derivation at which all these sentences are equal. They are not born equal, though, but simply converge semantically by unification of lexical specifications. This unification process is steered by syntax and parsing control, but its outcome is independent of that steering. SLF, in this sense, owes its architecture to the concept of Quasi Logical Form introduced and explored by Alshawi (1992).

Each lexical phrase is provided with a template – an HPSG-style attribute-value matrix – that specifies the structure *Store+Body* as the value of the lf-feature *semantics*. *Body* is a lambda term, representing the canonical meaning of the phrase. *Store* contains slots for the logical forms (lf) of dependent phrases: a verb stores the lf of its arguments, a preposition the lf of its complement, *etc.* The storage reflects the combinatoric patterns specified in the syntax. So the store contains lambda terms for all dependent constituents, and not just for scope sensitive operators, as was proposed by Cooper (1983) for early Montague grammar. For each slot in *Store*, the variable it operates on in *Body* is specified. Here is a general scheme for the *Store+Body* structure in a lexical template; all markers are variables, except for the predicates and relations in *Store*.

(331) *Store*: { $\text{DependentLF}_1 \lceil X_1$, ..., $\text{DependentLF}_n \lceil X_n$ }
Body: $\text{operator}_1 \hat{}$...$\text{operator}_q \hat{}\, \text{relation}_1(X_i, X_j)$ & ... & $\text{relation}_p(X_k, X_m)$

The variables in *Store* that stand for the logical forms of dependent constituents – $DependentLF_i$ in (331) – are linked to these logical forms inside the template. In general, they are instantiated by *Store+Body* structures themselves in the process of unification. As the outcome of this graph unification, the template of the computed phrase provides the *Store+Body* structure that contains all relevant logical forms of the sub-phrases involved but underspecifies the interaction of operators. In general, the *relations* in (331) are constants specified in the template itself.

The store may contain elements that do not correspond to syntactic constituents. In particular, the store of a finite verb will contain meanings indexed by its stem. This choice typically reflects the level of morphological decomposition fixed in the lexicon and the nature of the morpho-syntactic interface. For a very simple sentence like *Every man works,* after unifying the relevant templates for *every, man* and *works,* the *Store+Body* structure looks like this:

(332) *Store:* { {*Store:* {*Store:* ∅, *Body:* λX. man(X)⌈Q}
Body: λP.∀y. Q(y) → P(y)}⌈S},
∃z. event(z, *work*)⌈E }
Body: agentof(E, S) &attime(E,T) & time(T, present)

The operator $\varphi\lceil Y$ indicates that the formula φ in store has to be converted as to the variable X in the body: depending on the implicit typing, either the body is a term $\lambda X.\psi$ and type-sensitive β-conversion renders $\psi[\varphi/X]$, or φ is a term $\lambda R.\sigma$, the body is $\lambda X.\psi$ and β-conversion can only yield $\sigma[\lambda X.\psi/R]$. Neglecting these bookkeeping features of the storage, (332) could be read like this:

(333) *Store:* { {*Store:* {*Store:* ∅, *Body:* λx. man(x)}
Body: λQ.λP.∀y. Q(y) → P(y)},
λR. ∃z. event(z, *work*) & R(z)}
Body: λT.λE.λT.agentof(E, S) & attime(E,T) & time(T, present)

The interpretative formula before conversion would be this:

(334) λQ.λP.∀y. Q(y) → P(y)
(λR. ∃z. event(z, *work*) & R(z)
(λT.λE.λS. agentof(E, S) & attime(E,T) & time(T, present)(now)))
(λx. man(x))

The resulting conversions are here, stepwise.

(335) (i) *[now/T]*
λQ.λP.∀y. Q(y) → P(y)
(λR. ∃z. event(z, *work*) & R(z)
(λE.λS. agentof(E, S) & attime(E, now) & time(now, present)))
(λx. man(x))
(ii) *[(λE.λS. agentof(E, S) & attime(E, now) & time(now, present)))/R]*
λQ.λP.∀y. Q(y) → P(y)
(λS. ∃z. event(z, *work*) & agentof(z, S) & attime(z, now) &
time(now, present))
(λx. man(x))

(iii) *[(λx. man(x)/Q]*
λP.∀y. man(y) → P(y)
(λS. ∃z. event(z, *work*) & agentof(z, S) & attime(z, now) & time(now, present))
(iv) *[(λS.∃z. event(z, work)&agentof(z, S)&attime(z, now)& time(now, present))/P]*
∀y. man(y) → ∃z. event(z, *work*) & agentof(z, y) & attime(z, now) & time(now, present))

The representation (333) is, therefore, a kind of abbrevation for reasons of bookkeeping of the typed conversion protocol.

The representations for the quantifier, the noun, the verb and the inflectional element occur in a structured and labelled way in the sentential SLF: they are explicitly linked to constituents in the grammatical analysis. In (336), part of the lexical specification of the verb *zag* 'saw', as in *Elke man zag Henk werken* 'every man saw Henk work', is represented. Links between SLF and the other tiers of the analysis are underlined; the link is on indices of type X+Y. All structural variables are in bold face.

(336) *lexical template for* zag *'saw'*

```
ID:A+B
HEAD:CONCEPT:see
     PHON:zag
     SLF:see
     SYNSEM:ETYPE:state  ....
            PERSON:or([2, 3])
            TENSEOP:at-past
            VTYPE:semi_aux
PHON:C
PHONDATA:lijnop(zag, A+B, [arg(left(1), 0, D),
         arg(left(11),wh,E), arg(right(1), 9, F)], C)
SLF:{{[G&(B+H)↑I, {{[J$K&(B+L)↑M], [], []},
    N@some^O^and(quant(O, the), property~[M, O],
    entails1(O, decr), N, entails(O, incr))}&(B+L)↑P,
    Q@some^R^and(quant(R, some), state~[R, see],
    entails1(R, incr),
    and(Q, entails(R, incr)))&(A+B)↑S],[], []},
    and(and(and(and(experiencer_of~[S, I],
    goal_of~[S, T]), theme_of~[S, P]), attime(S, K)),
    tense(S, past))}
SYNSEM:CAT:s_vn
       CONTROL:controls(goal_of~[A+B, T], U~[B+L, T])
       PREDTYPE:nonerg
       TENSE:tensed  ....
TYPE:s_vn\0~[np^0#B+W, np^wh#B+H]/0~[vp^9#B+L]
ARG:ID:B+H
    PHON:E
    SLF:G
    SYNSEM:CASE:nom
           CAT:np
           NUMBER:sing
           OBJ:subject_of(A+B)
           PERSON:or([2, 3])
           THETA:experiencer_of
ARG:ID:B+L
    PHON:F
    SLF:J
    SYNSEM:CAT:vp
           EXSEM:X
           EXTTH:U~[B+L, T]
           THETA:theme_of
ARG:ID:B+W
    PHON:D
    SLF:X
    SYNSEM:CASE:obliq
           CAT:np
           OBJ:dirobject_of(A+B)
           THETA:goal_of
```

This template clarifies that whatever plays an active role in SLF is anchored in the grammatical analysis by a network of variables, open to dynamic instantiation. In this sense too, SLF is compositional and derivational.

2.6.3.2 Application of SLF: disambiguation

SLF reflects the morphological-syntactical complexity of the phrase: every constituent that contributes to the meaning of the whole phrase is represented in the relevant *Store+Body* structure. For this reason, different SLFs of one sentence correlate to different meanings. This property of SLF can be used to determine to what extent *extended lexical units* or 'constructions' have contributed to an SLF and, thus, what degree of lexical aggregation it represents. Normally, an interpretation with a high degree of lexical aggregation is preferred over an interpretation that was not, or less, based on extended lexical units. For the sentence *Nobody has kicked the bucket until now* the reading *Until now nobody has hit the bucket such that it fell* should have a considerably lower priority than the reading *Until now nobody has died.* In DELILAH, comparing SLFs in this respect suffices for selecting the least naive interpretation. It has been widely acknowledged by now – and it has never been denied, for all we know – that the lexicon of a natural-language processing system not only specifies single words; it must also – and maybe even predominantly – contain phrases or phrase structures that have a specialized syntax and/or semantics. At present, scholars proclaiming Construction Grammar (cf. Croft 2001) stress the central role of non-atomic construal in natural language. In formal grammar and computational linguistics applicants of HPSG, Categorial Grammar and Tree Adjoining Grammar always have accounted for considerably more involved structures than just atomic units.

Poß and Van der Wouden (2005) show that there is a huge variety of ways in which words and structures cluster to build complexes with specialized syntax or semantics. In general, lexical units limiting lexical selection or syntactical transparency will come with a meaning to which not every proper part contributes according to its proper class. Here are just some examples from Dutch:

- *hebben* with a DP can mean *possess*, among others; together with mass nouns like *honger* and *dorst*, it expresses in all its morpho-syntactic varieties the property of being hungry (thirsty); the verb itself hardly contributes to this meaning; the construal *to possess hungriness (thirstiness)* cannot be barred from being parsed;
- prepositional phrases may introduce adjunctive modifiers, but very often verbs select certain prepositions to express specialized meanings, e.g. *werken op iemands DP*, meaning *affect somebody's DP*, where the DP

introduces a psychological concept; parsing cannot be withheld from an adjunctive construal, but the selected meaning deserves priority;
- intransitive verbs V can be part of the so-called Dutch *way*-construction, expressing the meaning of moving to a certain position by V-ing, e.g. *to laugh oneself a way/path to DP*; the characteristic DP with the *way*-type NP does not play a role in the semantics.

In all these cases, the *Store+Body* structure representing the meaning of the extended lexical unit is simpler than the SLF resulting from a parse that does not account for the lexicalized meaning. In particular, the Store will contain less distinct lambda terms. Crucially, we do not presume that the semantic structure of an extended lexical unit is simpler in any logical or arithmetical sense. All we claim is that the store of an extended lexical unit will show less internal structure than its distributed counterpart. This follows from the construal of SLF. Therefore, a simple measurement on the stores of SLFs may select the simplest and highly aggregated and most 'normal' reading.
Here is an example. Any sentence containing phrases of the type *honger hebben* will come with two analyses: one reflecting the reading 'be hungry', the other reflecting the reading 'possess hunger'. The latter SLF will contain a store where the lambda term for the noun 'hunger' is specified. The former SLF will have an empty store instead, as the lambda term for 'to be hungry' irrespective of its logical complexity will not be stored, but specified in a body field of SLF. Computing the aggregated complexity of the stores and comparing them will identify the former SLF as the simpler one. Since this SLF is part of a full analysis, we can select this analysis as the preferred interpretation. The case is illustrated for the sentence *ik heb honger* 'I possess hungriness / I am hungry', with simplified representations of the *Store+Body* structures for the sake of transparency. The preferred reading has a simpler store than the non-preferred one.

(337)
SLF 1: `{ { i:x }, be_hungry(x) }`
SLF 2: `{ { i:x, hungriness:y }, possess(x, y) }`

Delilah produces all readings permitted by the grammar. For each SLF, it defines the structural complexity, *i.e.* the degree of embedding of *Store+Body* structures. For every acceptable parse, it computes a number that expresses the ratio between the structural complexity of the SLF and the total number of terms specified in the stores. This ratio marks the semantic complexity of the SLF. Parses can be ordered according to these ratios. In addition, Delilah computes the total number of predicates or small clauses in the bodies of the

SLFs. This number is used as a secondary marker for semantic complexity, but does not necessarily discriminate between lexical units and semantically composed phrases.
The approach to reading selection on the base of SLF complexity is steered by some principles:

(338) *Compositional complexity rul*e
The degree of compositionality of a phrase corresponds to the number of lambda terms in its store as compared to the number of syntactical arguments.

This gives an immediate first approach to the compositional complexity of a lexical sign.

(339) *Compositional complexity of a lexical sign*
The compositional complexity of a lexical sign equals 0 if the store is empty; otherwise, it is the number of lambda terms in its store divided by the number of proper parts that the sign specifies.

The compositional complexity is 0 for, *e.g.*, simple one-word phrases. It is $n/n = 1$ if all n specified proper parts of a construction contribute to the meaning of the whole phrase. It is a rational number if the construction is not purely compositional.
Apart from this simple metric, one could also count the number of semantic primitives and/or the number of variables in the compositional logical form, either in store or in the body or both, but we consider those data to be secondary, though not necessarily uninformative. Of course, metrics other than the two above for measuring compositional complexity are also conceivable.
Measure (339) extends to non-lexical signs after unification by making it recursive.

(340) *Complexity of Stored Logical Form*
The (compositional) complexity of a Stored Logical Form is the sign complexity plus the sum of the Stored Logical Form complexity of all the terms in its store.

In this we have a purely semantic, yet quantitative criterion for selecting readings. If two analyses differ only in their logical forms, choose the one with the lower compositional complexity of its Stored Logical Form. That analysis will contain the most aggregated meaning. It entertains at least one more extended lexical unit (elu) than the competitor.

In NLP, the most aggregated reading will be preferred, generally, since elus are listed in the lexicon because their aggregated meaning is more prominent than the full composition of their individual meanings. The complexity measurement therefore results in the following selectional strategy:

(341) *Underspecification selection rule*
If two signs differ only in compositional complexity of Stored Logical Form, the one with the lower compositional complexity of Stored Logical Form represents the lexically preferred prominent reading.

This way, the selection is driven by the identification of lexical or constructional units and the evaluation of their interpretations. As an example, we can see that among the two Stored Logical Forms for (342), the underspecification selection rule easily identifies the *be hungry* reading of sign (343) as lexically preferred over the *possess hunger* reading of sign (344), by comparing their SLF complexity.

(342) Elke man heeft honger
Every man has hungriness

(343) *reading 1*

```
head:[phonology:/heeft/, semantics:() ...]
      argument(1):[semantics:< store:{
                               < store: [],
                                 body:λP.λQ.∀Z.P(Z)→ Q(Z) > #Y},
                               body: λY.man(Y) >,
                  cat:np ...]
      argument(2):[phonology:/honger/, cat:np, semantics:....]
      semantics: < store:{
                  < store:{
                    < store:[],
                      body: λP.λQ.∀Z.P(Z)→ Q(Z)> #Y }
                    body: λY.man(Y)> #X },
                  body: λX.be_hungry(X) >...
```

SLF complexity according to (339): 1/2 + 1/1 = **1.5**

(344) *reading 2*

```
head: [phonology:/heeft/, semantics: possess, ... ]
argument(1):[cat:np, role:theme, ...,
             semantics: < store:{
                          < store: [],
                            body: λP.λQ.∀Z. P(Z) → Q(Z) > #Y},
                          body: λY.man(Y) >]
argument(2):[phonology:/honger/, cat:np,
             semantics: < store:{
                          < store: [],
                            body: λP.λQ.∃Z. P(Z)& Q(Z)> #X},
                          body: λX.hungriness(X) >]
semantics: < store:{
             < store:{
               < store: [],
                 body: λP.λQ.∀Z. P(Z) → Q(Z) > #Y },
               body: λY.man(Y) > #W,
             < store:{
               < store: [],
                 body: λP.λQ.∃Z. P(Z)& Q(Z)> #X },
               body: λX.hungriness(X) > #V }
             body: λW.λV.possess(W,V) >
```

SLF complexity according to (339): 2/2 + 1/1 +1/1 = **3**

Delilah is primed to pick the simplest reading for a sentence by exploiting the underspecification of SLF. This method is very reliable, because it is anchored in lexical specifications. In the analysis of the sentence *Sommige kinderen hadden weinig honger* ('some children were not very hungry') the *possess*-reading will be suppressed. This is because the lexicon contains an extended lexical unit relating *honger* and *hebben*, the SLF of which is simpler than the SLF constructed from independent lemmas for *to have* and *hunger*. The SLF for the whole sentence is constructed by sheer unification, including an integrated interpretation for the quantificational parameter *weinig* 'little'. Therefore, the complexity of the lexically specified SLFs is conserved during the derivation.

This approach to resolving lexical ambiguity lives off the layered structure of SLF. The basic idea is that the semantic elements which are contributed freely – without pre-derivational lexical commitment – are stored, whereas all other semantic specification dwells in the *body*-part of SLF. Free combinatorics thus increases the complexity of the stores, by definition and exclusively. Measuring the storage is therefore measuring the degree of combinatorial freedom. Note that this variation in degrees of freedom can only be accounted for at SLF, that is, before all semantic interaction is mixed up in a complex proposi-

tion. Consequently, in DELILAH, the structured SLF is an essential feature of lexical items, whether complex or simple.

Underspecified logical form therefore offers a natural and grammatically precise way to tell 'heavy' compositional readings from 'light' constructional readings while parsing, without any additional information being needed. It only refers to the semantic nature of the construction and the way in which (proper) parts of the construction contribute or do not contribute to the overall meaning. Since the overall meaning of constructions must be specified lexically anyway, no additional data are required for the metric to be applicable. The selection procedure comes free with a lexicon that allows for deep processing and semantic inference by specifying constructional meanings.

2.6.4 Applied Logical Form

In our approach, there is no special grammatical task for ALF. At the same time, this format – or a variant of it – complies best with requirements of human readability. To illustrate that, we repeat the (labelled, edited and 'normalized') SLF, the disjunctive FLF and the ALFs of the sentence *Elke man zoekt een eenhoorn* 'Every man seeks a unicorn' (from section 2.2).

(345) Stored Logical Form

```
< [store0
<<[store1 λX.state(X, man) 1erots] λJ.λK. quant(I, every) & J(X) &
theme_of(X,I) & entails1(I, decr) & K(I) & entails(I,incr)>>,
<<[store2 <<<[store3 <<<<[store4 λY.state(Y, unicorn) 4erots],
λQ.λR.quant(P, some) & Q(Y) & theme_of(Y, P) & entails1(P,
incr) & R(P) & entails(P, incr) >>>> 3store],
λT.quant(U, some) & event(U, find) & entails1(U, incr) & T(U) &
entails(U, incr)>>> 2erots],
λW.λU.λP.λF. agent_of(U, I) & theme_of(U, P) & attime(U, A) >
1erots]
λX.λY.quant(Z, the) & property(X(L)), Z) & entails1(Z, decr) &
Y(Z), entails(Z, incr)>>,
λC. quant(D, some) & event( D, try) & entails1(D, incr) & C(D)
& entails(D, incr)
0erots],
λI.λZ.λD.λA. agent_of(D, I) & theme_of(D, Z) & attime(D, A) &
tense(D, pres) >
```

(346) Flat Logical Form

```
state( s^i, man) &
theme_of( s^i, G+↓+every+[] ) &
event( e^j, try) &
property(D+↓+the+[]) &
event(e^k+[D], find) &
agent_of(e^k+[D], G+↑+every+[]) &
  (theme_of(e^k+[D], H+↑+some+[D,G] );
   theme_of(e^k+[D], H+↑+some+[G] );
   theme_of(e^k+[D], H+↑+some+[] ) ) &
state( s^j, unicorn) &
  (theme_of(s^j, H+↑+some+[D,G] );
   theme_of(s^j, H+↑+some+[G] );
   theme_of(s^j, H+↑+some+[] ) )&
attime(e^k+[D], F) &
agent_of(e^j, G+↑+every+[]) &
theme_of(e^j, D+↑+the+[]) &
attime(e^j, F) &
tense(e^j, pres)
```

(347) Applied Logical Form

(a)

```
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(D,the).[property(D).[quant(H,some).[state(s^j,
unicorn) & theme_of(s^j, H) & event(e^k, find) & agent_of(E,G) &
theme_of(e^k,H) & attime(e^k,F)]]] & agent_of(e^j,G) &
theme_of(e^j,D) & attime(e^j,F) & tense(e^j,pres)]
```

(b)

```
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(H,some).[state(s^j, unicorn) & theme_of(s^j, H) &
quant(D,the).[property(D).[event(e^k, find) & agent_of(e^k,G) &
theme_of(e^k,H) & attime(e^k,F)]] & agent_of(e^j,G) &
theme_of(e^j,D) & attime(e^j,F) & tense(e^j,pres)]]
```

(c)

```
quant(H,some).[state(s^j, unicorn) & theme_of(s^j, H) &
quant(G,every).[[state(s^i, man) & theme_of(s^i, G)] ⟹ event(e^j,
try) & quant(D,the).[property(D).[event(e^k, find) &
agent_of(e^k,G) & theme_of(e^k,H) & attime(e^k,F)]] &
agent_of(e^j,G) & theme_of(e^j,D) & attime(e^j,F) &
tense(e^j,pres)]]
```

Just like FLF, ALF is spelled out post-derivationally from SLF. It is cast as enriched predicate logic, in the sense that it must accommodate operators and quantifiers other than strictly logical ones. Contrary to FLF, it is neither a conjunction, nor operator-free. Just like FLF, all semantic dependencies are unambiguously and fully encoded. ALF is structured by logical operators.

ALF can best be seen as the logical label of the semantic process. We assume that its role in natural-language processing is limited. Yet, the number of ALFs delimits the number of readings at FLF. The multiple disjunction of local readings in (346) suggests an explosion of readings. This is not intended. Reckman (2009) presents an index on FLF to the effect that in a certain disjunction the choice of one reading implies the choice of a local reading elsewhere – to wit, the same one. This index on readings is not represented here, but is assumed to be part of the FLF-generation procedure.

2.6.5 Flat Logical Form

2.6.5.1 Construal

Each SLF that passes the complexity test is submitted to a post-derivational algorithm, outlined in section 2.3.3, in particular in (272). This algorithm performs two tasks: it converts the SLF into a coherent formula and it compiles onto each variable the additional semantic information resulting from this conversion.
The first step involves β-reduction of implicitly typed lambda terms, while explicitly specifying scope dependencies between semantic operators like quantifiers, modalities and negation. This conversion is sensitive to scopal variation of operators, to the semantic nature of operators and to structural restrictions on scope imposed by islands or other intervention effects. In this step, a good deal of a semantic theory can be effectuated, for example with respect to islands, scope and negation. This step yields a more or less standard logical form. All quantifiers, though, are described rather than interpreted as operators (as *e.g.* in (295) and (313)), to allow for quantificational variety.

In the second step, the information spelled out by the conversion is specified at each occurrence of each variable by a compilation protocol that turns the semantic operators superfluous. After applying this protocol, each variable is locally sugared with information as to

- its entailment property,
- the quantificational regime it is bound to,
- the variables its instantiation is dependent upon.

The entailment property indicates whether the predicate the variable is an argument of allows for upward, downward or no entailment with respect to the variable. The specification of 'governing' variables indicates whether a

variable is referentially independent or has to be valuated by a choice function; the notion of variables y_i governing a variable x thus amounts to the introduction of a choice function $f(y_1,..,y_n)$ for x, claiming: given values for y_i there is a way to determine a value for x such that the proposition resulting from these valuations is true.

The information encoded on the variables is compiled from an intermediate representation where lambda terms are fully converted and scopes are specified. The index with respect to entailment – *up* and *down* – specifies the local monotony properties of the binding quantifier in the relevant domain, according to its definition as a general quantifier. For simple, lexical determiners this is straightforward, as these properties are lexically defined by the theory of generalized quantifiers (in Barwise and Cooper 1981, and Zwarts 1986, for example). For complex determiners like *at least n but not more than m* entailment properties have to be computed from the composition; in many of these cases no monotony can be detected, however. This calculus is discussed in Reckman (2009).

The index with respect to the binding quantifier can be complex, as the quantifier itself is complex. Yet, complex quantifiers too should be classified in such a way that *e.g.* existential impact of the quantifier can be derived immediately. Note that the entailment property of the variable *y* varies with the domain of its quantifier: in the restriction of the universal quantifier, the variable bound by it allows for downward entailment in the nuclear scope it gives rise to upward entailment. Referential dependency does not vary with domains.

In our system, FLF is derived from a purely semantic, fully-specified logical form ALF (cf. section 2.6.4). Neither this logical form nor its derivative FLF contains any language-specific information. Therefore, if two sentences mean the same, their logical forms must be equivalent – which is undecidable – and their FLFs will comply. Paraphrasing an example of Shieber (1993), the following five sentences share one particular canonical and normalized semantic representation, in so far as they share a meaning:

(348) Clapton led Derek and the Dominos
Derek and the Dominos was led by Clapton
The leader of Derek and the Dominos was Clapton
Clapton yazzℜr ĩ Derek ɔd the Dominos
Clapton yămôs amuzăr ɔn Derek ɔd the Dominos

According to the requirements above, their FLFs, when fed into appropriate generators for English and Tuareg respectively, should give rise to these, and maybe even more, sentences. Note that looking at logical form in this way implies that it is underspecified in comparison to natural-language sentences inducing it. This underspecification is inevitable, even when the logical form itself is fully specified from a logical point of view: sentences 1 to 5 are neither equal nor equivalent, but their logical forms are.
As a matter of fact, Flat Logical Form is an *index* on ALF. The FLF is derived by taking every single predicate term at ALF and indexing every variable in that term for the way it is bound, for polarity and for the variables on which its valuation depends. Here are two examples.

(349) Every man invites a woman that he does not know

standard first-order representation:

```
∀x. [M(x) ⇒ ∃y. [W(y) & I(x,y) & ¬K(x,y)]
```

logical form ALF:

```
∃e1 ∃e2 ∃e3 ∃e4 ∀x. [[state(e1, M) & th1( e1,x)] ⇒ ∃y.
[state(e2, W) & th2(e2, y) & state(e3, K) & ¬[th3( e3, x) &
th4(e3, y)] & event(e4, I) & th5( e4, x) & th6(e3, y)]]
```

index FLF:

```
{ state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↓+all+[]),
  state(e2+↑+some+[], W), th2(e2+↑+some+[], y+↑+some+[x]),
  state(e3+↓+some+[], K), th3(e3+↓+some+[], x+↓+no+[]),
  th4(e3+↓+some+[], y+↓+no+[x]), event(e4+↑+some+[],I),
  th5(e4+↑+some+[],x+↑+all+[]), th6(e4+↑+some+[],y+↑+some+[x])}
```

(350) Few monkeys dare to fly

standard – no first-order standard

logical form ALF:

```
∃e1 ∃e2 ∃e3 Few(x). [state(e1, M) & th1( e1, x) & state(e2, D)
& th2(e2, x) & iy. [property(y) & th3(e2, y) & th5(y, e3) &
event(e3, F) & th4(e3,x)]]
```

index FLF:

```
{state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↕+few+[]),
 state(e2+↑+some+[], D), th2(e2+↑+some+[], x+↕+few+[]),
 property(y+↕+the+[e2]), th3(e2+↑+some+[], y+↕+the+[]),
 event(e3+↑+some+[y], F), th4(e3+↑+some+[y], x+↕+few+[]),
 th5(y+↕+the+[e2], e3+↑+some+[y])}
```

Of course, the construction of ALF involves numerous decisions on the semantics of natural language, many of which are not consolidated (cf. Cooper *et al.* 1996). These decisions, like introducing intensional domains or assigning wide scope to state and event quantifiers, are not at issue here: whatever

clause occurs in ALF is indexed at FLF. The index is computed along with LF, and can be exploited for computing entailment between sentences and sentence generation.

The exploitability of the FLF index on logical form resides in its permutability. Given an FLF, one can select the subset of clauses that share a particular variable. Let us call this subset a *net.* For each net, a normal form can be constructed, *e.g*. by some ordering of the predicates involved. Here is a complete network for the FLF of (350):

(351)

e1-net: `< state(e1+↓+some+[], M), th1(e1+↓+some+[], x+↓+all+[]) >`
e2-net: `< state(e2+↑+some+[], W), th2(e2+↑+some+[], y+↑+some+[x])>`
e3-net: `< state(e3+↑+some+[], K), th3(e3+↑+some+[], x+↑+no+[]),`
`th4(e3+↑+some+[], y+↑+no+[x]) >`
e4-net: `< event(e4+↑+some+[], I), th5(e4+↑+some+[], x+↑+all+[]),`
`th6(e4+↑+some+[], y+↑+some+[x]) >`
x-net: `< th1(e1+↓+some+[], x+↓+all+[]), th3(e3+↑+some+[], x+↑+no+[]),`
`th5(e4+↑+some+[], x+↑+all+[]) >`
y-net: `< th2(e2+↑+some+[], y+↑+some+[x]),`
`th4(e3+↑+some+[], y+↑+no+[x]), th6(e4+↑+some+[], y+↑+some+[x])>`

The effort to construct the network of the FLF index is proportional to the number of variables in the LF. The network is the anchor for computation of entailment and for generation. It seems a characteristic property of natural-language Logical Form, reflecting the semantic coherence of sentences, that in its FLF every variable defining a net occurs in at least one other net. It is worthwhile to point to this form of connectivity by way of conjecture.

(352) *Connectivity conjecture for FLF*
Given the LF of a well-formed and meaningful sentence S, in the network that represents its FLF, every indexed variable that defines a net occurs as an indexed variable in at least one other net.

2.6.5.2 Application: inference

The notion of entailment for unrestricted natural language is far from being logically validated. From a processing point of view, entailment is problematic on both the logical and the semantic sides. As for the semantics, many – if not most – constructions of natural language lack any consolidated interpretation. As for the logic, even first order representations of natural language are too rich to be left to theorem-provers. In addition, Shieber has pointed out that equivalency for first-order logic, being undecidable, controlled generation of (unrestricted) natural language from pure first-order logic con-

straints is not feasible (Shieber 1993). This means that reasoning in natural language by entailment, using first-order logic, cannot induce provably correct verbalization of the outcome.
It is not just a trend that these days textual entailment is generally computed by shallow means – not by full grammatical representations. The RTE challenges show that relatively shallow analysis, combined with intelligent search and/or learning strategies, may cover a fair amount of 'common sense' entailment, as was pointed out by *e.g.* Chambers *et al.* (2007, Tatu and Boldovan (2007) and Bobrow *et al.* (2007). Moreover, deepening the grammatical analysis does not necessarily improve the result – the argument here is by Bos and Merkert (2006). Yet, the constructional nature of unrestricted natural language calls for deep processing: the lexicon for processing is bound to be phrasal, to contain complex lambda terms and to be submitted to subtle syntactic combinatorics in order to maintain phrasal semantic integrity. The phrasal lexicon is the source of all semantic wisdom, but in order to exploit it in processing a lot of grammar is required. Cooper *et al.* (1996) argue convincingly that several tasks for natural-language processing require structural semantic processing – deep structural semantic processing.

FLF is an operator-free conjunction of disjunctions. Each conjunct indexes a part of the ALF. All logical information is specified in every small clause as indices on variable occurrences. This representation intends to facilitate inference. To decide whether a certain inference is possible, it suffices to linearly inspect an FLF and, for each conjunct, to locally decide whether or not it gives rise to (part of) the hypothesis, *i.e.* the sentence to be inferred.

Here, we propose a means of exploiting deeply processed and fully-specified logical form for the sake of computing entailment for unrestricted natural language and for generation. The representations used can be computed, and are taken to enhance semantic inference between sentences and to feed into controlled generation (see section 2.7).

Our strategy is to approach the modal notion *S entails T* for Dutch sentences S and T by inspecting their respective FLF networks. We define a notion of *l-coverage* for FLF networks and suggest that l-coverage of FLFs is a sufficient condition for entailment.

(353) For Dutch sentences S and T, S *entails* T if FLF(S) *l-covers* FLF(T).

In order to compare the FLFs, we must assume that their respective networks can be compared *salve* alphabetic variance. This is far from trivial, but we will not pursue this here, since this problem is more general; we simply assume that an educated guess as to the line-up of variables in the two FLFs can be made.
Under this assumption, we first define the notion of *i-coverage* for individual FLF-clauses φ and ψ. Below is a tentative inventory of its instances; each instance is illustrated by a natural-language entailment relation in which that instance of *i-coverage* is intended to be the decisive licensing factor – the phrases involved are in small caps. In this definition we take <φ,ψ> to be a *subnet* – the clauses share a variable by definition. σ stands for two-place relations between an indexed variable and a conceptual constant, like *state* and *event*; τ abbreviates all relations between indexed variables, like thematic roles. Furthermore, we assume that between concepts and predicative constants a subsumption order may be defined, by meaning postulates and/or by ontologies such that X↓ ⊑ X↑ means that X↓ is at least as specific as X↑. Moreover, lists of governing variables are supposed to be in alphabetical variance when checked; this is indicated by the notation F/F'. Finally, an *intensional net* is a net which is headed by an intensional predicate like *proposition/1* or *property/1* and the defining variable of which is dependent on some other variable.

(354) I-coverage

(a) *identity:*

φ *i-covers* ψ

if φ and ψ are alphabetic variants

(b) *strengthening*:

φ[F] *i-covers* ψ[F']

if φ and ψ are alphabetic variants except for the list of parameters F and F', and F-F' does not contain variables that define an intensional net in the FLF of the covering clause and F'-F does not contain variables that define a net in the FLF of the covered clause

(These men kissed A WOMAN *entails* SOME WOMAN is kissed;
He said that A MAN died *not-entails* A MAN died;
SOME WOMAN kissed him *not-entails* These men were kissed by A WOMAN)

(c) *upward coverage:*

<σ(A,C),τ(A, x+↑+Q+F)> *i-covers* <σ(A,C↑), τ(A, y+↑+Q+F')>

if C↑ is defined by subsumption

(Every candidate is GERMAN *entails* Every candidate is EUROPEAN)

(d) *downward coverage:*

<σ(A,C), τ(A, x+↓+Q+F)> *i-covers* <σ(A,C↓), τ(A, y+↓+Q+F')>

if C↓ is defined by subsumption

(At most three candidates are EUROPEAN *entails* At most three candidates are GERMAN)

(e) *downward extension:*
<σ(A,C), τ(A, x+↓+Q+F)> *i-covers*
<σ(A,C), τ(A, y+↓+Q+F'), τ'(A, y+↑+Q'+[ ]) >
if τ' does not occur in the covering net
(No woman gave flowers *entails* No woman gave flowers to me)

(f) *upward reduction:*
<σ(A,C), τ(A, x+↑+Q+F)> *i-covers* σ(A,C)
(John ate an apple *entails* John ate)

(g) *existential impact:*
τ(A, x+M+Q+F) *i-covers* τ(A, y+M+some+F')
unless Q = *no*
(That student read a book *entails* Some student read a book
Few students read a book *entails* Some student read a book)

(h) *upward concept:*
σ(e1+↑+Q+F, C) *i-covers* σ(e2+↑+Q+F', C↑)
if C↑ is defined by subsumption
(She is a girl *entails* she is a woman)

(i) *downward concept:*
σ(e1+↓+Q+F, C) *i-covers* σ(e2+↓+Q+F', C↓)
if C↓ is defined by subsumption
(At most three students were ill *entails* At most three students had the flu)

(j) *upward relation:*
τ(A, x+↑+Q+F) *i-covers* τ↑(A, y+↑+Q+F')
if τ↑ is defined by subsumption
(This man hit the woman entails the man did something to the woman)

(k) *downward relation:*
τ(A, x+↓+Q+F) *i-covers* τ↓(A, y+↑+Q+F')
if τ↓ is defined by subsumption
(No men did anything to this woman entails No men hit this woman)

(l) *distribution:*
<φ, ψ> *i-covers* <φ', ψ'> iff φ *i-covers* φ' and ψ *i-covers* ψ'

This list of elementary coverings is rather tentative, but the ambition will be clear: the relationship between FLFs can be constructed from more elementary relations at the level of nets and subnets. In linguistic terms, it means that, by using the FLF index, we are trying to reduce the relation between (meanings of) sentences to a relation between (meanings of) phrases, notwithstanding the complex logical and syntactical interaction between these phrases. On this basis, we can define a notion *n-coverage* for the relationship between nets.

(355) A net N is *n-covered* by a net M if every clause in N is *i-covered* by M.

The *i-coverage* almost has to be functional in this definition: in the list of pairs of i-covering and i-covered subnets, no subnet can occur in an i-covering posi-

tion more than once, with the possible exception of applications of *downward extension* (354) (e).
This leads to the following notion of *l-coverage* between FLFs.

(356) *L-COVERAGE*
An FLF $\Phi = < \varphi_1, \ldots \varphi_k >$ *l-covers* an FLF $\Psi = < \psi_1, \ldots \psi_m >$ iff
for every network $N_\psi = <\psi^1 \ldots \psi^n>$ in Ψ there is exactly one (possibly empty) network $N_\varphi = <\varphi^1 \cdots \varphi^m>$ that *n-covers* it.

In order to back up conjecture (353), which introduces l-coverage as a sufficient condition for entailment, we argue that we essentially treat the FLF index as a conjunction of independent clauses. FLF abstracts away from any non-commutative connective at LF, in particular, from the material implication. Although FLF-indexing itself is non-reversible, one observation is relevant: conjunction entails implication if we do not allow for *ex falso* interpretation of implication in the combinatory semantics of natural languages. Seuren (2006) argues that the cognitive-pragmatic construal of natural language does not leave space for zero-valued antecedents. Following his lead, we assume that phrasal semantics does not induce *ex falso* interpretation – the universal quantifier has existential impact. Under this assumption, the 'conjunctive' logic of FLF as presented above is more restrictive than the logic of ALF, which goes proxy for the relation between sentences. In that case, conjecture (353) seems a fair and cautious approximation of entailment in natural language.

2.7 GENERATING FROM LOGIC

2.7.1 Generation as translation

In the preceding section, DELILAH was explained as computing – apart from a semi-standard full representation ALF – two related but procedural and formally very distinct levels of semantic representation: underspecified, compositional SLF and fully-specified, para-compositional FLF. SLF is produced as part of the graph unification, which is the kernel of the parsing and generation procedures. Therefore, its construction does not complicate the derivation. Yet, on SLF, measures can be defined that express essential semantic properties of the structure and can be exploited for selection of readings and

reduction of ambiguity. Developing the best and most telling measurements in this respect is an empirical matter, but it is clear that hard-boiled criteria can be found and applied in reducing ambiguity exactly where it resides: in the semantic structure of a sentence. Of course, not all problems of selecting readings can be handled at SLF. Pure global polysemy cannot be decided upon by inspecting SLF. But the delicate balance between levels of lexical aggregation can most certainly be tracked at this level.

SLF seems a good level for generation to take off because of its sensitivity to constituent structure. We aim at a procedure to generate sentences the SLF of which gives rise to an FLF that entertains a strict logical relationship to another FLF, the generation constraint. That is, the procedure starts with analyzing an FLF, proceeds with guessing a syntactical structure on the basis of that information, then spells out its SLF and compares the result with the original FLF. In this sense, the labour division between SLF and FLF may contribute to purely meaning-driven translation, as argued in Alshawi (1991) and Copestake *et al.* (2005). Rosetta (1994) made clear that machine translation on a semantic base – this is *not* a pleonasm nowadays – flourishes by strict and controllable compositionality. FLF approaches the representation of meaning in the spirit of Minimal Recursion Semantics (MRS) as implemented in the English Resource Grammar (Flickinger *et al*. 2000, Copestake *et al.* 2005). The main point of convergence is that MRS and FLF present full semantics while avoiding syntactical complexity of the logical form. There are some differences, however. Since FLF is derived from SLF, all constraints on the interaction of semantic operators – like weak island conditions – are integrated in this derivation. The construal of FLF thus embodies an explicit theory on the syntax-semantics interface. Furthermore, FLF encodes all scopal and inferential information directly onto the logical form, rendering inference strictly local, as was pointed out in section 2.6.5.2.

In the following sections, we present a system for generating natural language that acts as a semantic translation automaton. Both the source and the target are FLFs; the precise semantic relationship between these two FLFs can be computed, and yields the criterion for the generation's success. In between, Delilah's generator explores the syntactic-semantic interface, deriving well-formed sentences with SLFs to be converted into FLF. Schematically:

(357) *Generation procedure*
(a) input: FLF InputFLF
(b) transform InputFLF into a set of lexical constraints Constraints
(c) generate a sentence Sentence that obeys Constraints
(d) transform Sentence's SLF into OutputFLF
(e) check whether OutputFLF entails InputFLF

(f) if (e) is negative, repeat (c)-(e); else: done;
(g) output: Sentence, *translating* InputFLF

This procedure is essentially non-deterministic. In stage (b), as much information as possible is gained from the full semantic representation InputFLF, but – quintessentially – this logical form contains too little information to determine the structure of a sentence representing it. Because the semantic constants in FLF are taken from lexical SLF – the spell-out does not add or delete any conceptual content – the acquisition of information from FLF amounts to a line-up of lexical units that can or must contribute to the sentence, with some pre-established relations between them. Starting from this line-up, at stage (c) a sentence is constructed according to the generation procedure of chapter 1. The generation procedure tries to accommodate at most and at least the concepts of the line-up. If the procedure comes up with a sentence that meets these requirements, its FLF is computed. We assume that OutputFLF is at least as specific as InputFLF because InputFLF inevitably underspecifies the features of its translation. Therefore, at stage (e), OutputFLF is checked to entail InputFLF. The outcome of this check may be negative, for reasons beyond predictability but having to do with the fundamental incongruence between form and meaning. If so, the generation procedure is so little deterministic that it may come up with another sentence meeting the lexical and conceptual requirements. We assume that in principle it is impossible to add reliable information to this backtracking manoeuvre as to the source of the proven incompatibility. If that were possible, the information could have been added before, making the generation essentially deterministic. But in that case, InputFLF would contain full information with respect to sentence construal. This runs counter to every semanticist's experience: the track from form to meaning is one-way. Going from form to meaning and back is taking a roundabout, with obligatory change of vehicle. Yet, we will argue that the grammars for parsing and generation are essentially and characteristically identical, to wit, the joint of the syntax of chapter 1, the semantics of this chapter and the lexicon of chapter 3.

2.7.2 From logical form to lexical line up

Entailment is a relation between natural-language sentences. That relationship can be judged by language users: natural-language semantics is an empirical art when founded on entailment (cf. Chierchia and McConnell-Ginet 2000). At the end of the day, it is not the relation between formal representations, but the informal relation between sentences, that can and must be tested. To lead

entailment and reasoning back to natural-language processing, we must be able to generate from those logic representations that are exploited for reasoning and inference. We now present a non-deterministic procedure to generate Dutch sentences with a predefined, fully-specified formal meaning; of course, the procedure is grafted on the DELILAH parser and generator. The input to the procedure is an FLF index, as presented above. This index formula contains semantic information only. The output is a well-formed Dutch sentence with a full grammatical representation, again providing a formula in LF and its index FLF. The main characteristics of the procedure are:

- the input constraint is not biased towards the syntax or the lexicon;
- the generation procedure is non-deterministic, but finite;
- the result can be validated logically: input and output semantics are formulas in the same language.

Here, we describe a method to relate FLF to the lexicon and the (categorial) grammar by exploiting the semantic nets induced by it, exemplified in (351). These networks are shown to be able to steer lexical selection and grammatical unification. *L-coverage* (356) is applied to validate the result.

The generator is hypothesis-driven: it tries to construct a well-formed and meaningful phrase of a given category, with a complete parse in the form of a unified graph representing a complex symbol. The generation procedure is strictly meaning-driven: it operates without any structural preconditions, unlike the logical form-driven generators described in Carroll *et al.* (1999) and White and Baldridge (2003). In these 'realizers' – as in the famous translation system of Rosetta (1994) – the input to the generation procedure contains essential pieces of structural information relevant to the output. Here, we propose a purely semantic way of constraining the realization, as our input constraint completely abstracts from syntax. Generation proceeds by selecting appropriate phrases from the lexicon after inspecting an agenda and by testing their unification. The generation is successful if the hypothesis can be checked, no item is left at the agenda and some non-empty structure has been created.

Basically, the algorithm tries to find templates and tries to unify them according to an agenda which is set by an initial hypothesis and updated by applying combinatory categorial rules. The agenda consists of two parts: *given*, corresponding with complex symbols already adopted, and *to_find*, corresponding to structures still to be checked. A successful unification of complex symbols according to the agenda is the genuine result of the procedure.

An FLF offered to the generator is 'chunked' into nets. As an example, the set of clauses in the FLF of a Dutch sentence *Elke vrouw probeerde te slapen* 'Each woman tried to sleep' is given here.

(358) { state(e1+↓+some+[], *woman*), th1(e1+↓+some+[], x+↓+every+[]),
tense(e2+↑+some+[], *past*), event(e2+↑+some+[], *try*),
th2(e2+↑+some+[], x+↑+every+[]), property(y+↕+the+[e2]),
th3(e2+↑+some+[], y+↕+the+[e2]), event(e3+↑+some+[y], *sleep*),
th4(e3+↑+some+[y], x+↑+every+[]),
th5(y+↕+the+[e2], e3+↑+some+[y]) }

For each variable all the clauses in which it occurs as an argument are selected. Here is the resulting list of variable-induced nets over (358).

(359)

e1-net: <state(e1+↓+some+[], *woman*), th1(e1+↓+some+[], x+↓+every+[])>

e2-net: <event(e2+↑+some+[], *try*), th2(e2+↑+some+[], x+↑+every+[]),
th3(e2+↑+some+[], y+↕+the+[e2]), tense(e2+↑+some+[], *past*)>

e3-net: <event(e3+↑+some+[y], *sleep*), th4(e3+↑+some+[y], x+↑+every+[]),
th5(y+↕+the+[e2], e3+↑+some+[y])>

x-net: <th1(e1+↓+some+[], x+↓+every+[]), th4(e3+↑+some+[y],
x+↑+every+[]), th2(e2+↑+some+[], x+↑+every+[])>

y-net: <property(y+↕+the+[e2]), th3(e2+↑+some+[], y+↕+the+[e2]),
th5(y+↕+the+[e2], e3+↑+some+[y])>

Each of the nets P[X] in (359) is taken to be a sequence of simultaneous conditions on the semantics of some lexical phrase. Together, the nets determine the lexical space for a generation process. Per net, all the candidates in their lexical quality of complex symbols are selected. That being done, the generator tries to produce a grammatical construct in which each net is represented exactly once. The agenda for this process is derived from the contingent syntactic properties of the candidates.

The conceptual agenda raised by the FLF-network strictly limits the freedom of the generator. Even when the available lexical space is extended by allowing purely functional additions, infinite looping is excluded under the cancellation agenda. On backtracking, the generator will produce all and only the sentences that live within the lexical space constructed by the input network and are reachable by the grammar.

Yet, this generation procedure from FLF is essentially non-deterministic in at least two senses:

- the (structure of the) FLF does not fixate the structure of the sentence, by definition of FLF;
- the output-FLF may not match the input-FLF, according to a certain semantic standard.

Like LF, FLF underdetermines not just the syntax, *e.g.* the word order, of sentences generated in this way. Because of this ‘inverse underspecification’ – LF

and FLF are not the outcome of a derivation but the spell-out of an underspecified unification result – the generation procedure cannot fixate all the characteristics of the produced semantics in the logical space in advance or *on the fly*. There are two sources for this flaw:

- FLF may itself contain less specifications (aspect, mood, tense, …) than any verbalization would introduce;
- inspection of the input-FLF cannot predict which logical dependencies between variables are blocked or improved by following a certain construction mode for the sentence.

The first aspect of generative underspecification is evident: one cannot be sure that an FLF contains all the information that a full sentence will produce by default. Natural language is meant to be more expressive than any logical representation. Complex symbols may introduce additional meanings to those mentioned in the conceptual agenda, *e.g.* by default specifications like tense on finite verbs. The concepts in the input are a subset of those in the output. Moreover, the input FLF, when constructed by reasoning, may not specify semantic dependencies that are inherent in sentential construal, like intensional embedding.

The second incongruence between input-FLF and output-FLFs is due to the form-driven nature of sentence meaning. Whether or not a certain operator can scope over another depends partly, if not mainly, on its embedding. The generation process may isolate an operator on a strong or weak island, by choosing certain phrases or certain syntactical patterns for the respective nets. For example, a quantifier embedded in a nominal construction (*to make the promise that ….*) has fewer scope options than a quantifier embedded in a non-nominal, but conceptually equivalent construction (*to promise that …*). In the same vein, intensional domains are not completely predictable from inspecting an FLF. Generally, weak and strong islands of any sort are induced by syntax, and the syntax is underspecified, by definition and inevitably. Consequently, the generation procedure cannot be enriched with an additional agenda controlling possible scopal dependencies. Scope can be checked or compared only *post hoc*.

Since FLF – and in fact, every purely semantic logical form – contains too little information to fully determine the generation procedure, generating from logic is a non-deterministic trial, by necessity. The outcome of the process can or must be checked against the input constraint. It is important to realize, however, that the input constraint and the output FLF can differ in a limited number of ways only. For example, the output may contain concepts that are not present in the input. But this situation occurs only if these concepts are

introduced by default, when applying certain complex symbols and if they passed the restrictions on unification imposed by the semantic networks. The output is far from being in free variation with the input.
As was argued above, it is not wise to check for strict identity or equivalence of input- and output-FLFs. Taking into account the considerations given above with respect to the 'inverse underspecification', we propose that the normal check would be as in

(360) Accept S with OutputFLF as a generated translation of InputFLF into Dutch iff OutputFLF *l-covers* InputFLF.

Informally, this means that the generated sentence is at least as specific as the input, or that a model for OutputFLF is also a model for InputFLF, but not necessarily the other way round. Again, it should be noted that the conceptual difference between InputFLF and OutputFLF will be very limited, restricting lexical resources to those induced by the semantic nets of InputFLF.

2.7.3 From lexical line-up to sentence: intertwining agendas

Given the lexical constraint of the previous section – basically: a restriction on the lexicon – the generator can start producing sentences within this restriction. The engine for this procedure is the categorial grammar and the *constructional agenda* that can be derived from its types (see chapter 1). For the sake of efficiency, we may add to the procedure a *semantic agenda*: a list of concepts that must occur exactly once in the sentence to be produced. Again for reasons of efficiency, it makes sense to combine the constructional and the conceptual agendas. The semantic agenda does not provide constructional information, though. Therefore, the generation procedure is essentially nondeterministic, if not of the *trial-and-error* breed. Yet, there is no objection to exploiting the network information as much as possible. It is, for example, advisable to retain the network itself as part of the semantic agenda, and put it to use in the following way. Suppose the constructional agenda requires a phrase with an *np* as a head to be instantiated and suppose that this part of the agenda is dictated by a phrase representing network *e2-net.* Before selecting a phrase to fulfil this task, check which variable in the actual network is related to that *np* and determine for that variable which semantic constraints it induces. Use these constraints to select a candidate phrase from the restricted lexicon. In the example it is clear that if the desired *np* is supposed to deliver the agent for the *try*-instantiation, the *x-net* immediately tells

you that this agent entertains universal quantification: a candidate for the *np*-phrase had better comply with this semantics.
The architecture for this intertwining of agendas is an empirical matter. Instead of trying to capture idle wisdom in words, let us look stepwise at the construction of a sentence according to DELILAH's present state.

Suppose we have an FLF like (361) with networks (362):

(361)
```
{ agens(e^i, x+↑+every+[])
  event(e^i, sing)
  atplace(e^i, y+↑+some+[])
  state( s^j, car)
  theme_of( s^j, y+↑+some+[])
  state( s^k, man )
  theme_of( s^k, x+↓+every+[] }
```

(362)
```
x-net:   <agens(e^i, x+↑+every+[]),
          theme_of(s^k, x+↓+every+[])>
y-net:   <atplace(e^i, y+↑+some+[]),
          theme_of(s^j, y+↑+some+[])>
e^i-net: <agens(e^i, x+↑+every+[]),
          event(e^i, sing),
          atplace(e^i, y+↑+some+[])>
s^j-net: <state(s^j, car),
          theme_of(s^j, y+↑+some+[])>
s^k-net: <state(s^k, man) &
          theme_of(s^k, x+↓+every+[])>
```

The *x-net* does not contain a non-functional concept. The set of lexical items induced by that net, LEX(x-net), therefore consists only of quantificational elements representing the functional concept *every*, among which of course are determiners like *elke* 'every' and *alle* 'all'. They are stored in the *ad hoc* lexicon, referring to variable x. The same holds *mutatis mutandis* for LEX(y-net). LEX(e^i-net) is centered around the concept *sleep* and contains at least those lexical instances of this concept that introduce both events and agents; it does not contain the nominal versions of *sleep* as they will not meet these requirements. LEX(s^j-net) is centered around the concept *car* and selects those lexical entries of this concept that introduce at least a state. Similarly, LEX(s^k-net) contains at least the nominal instances of the concept *man*. All retrieved lexical items are indexed for the variable of their network.
Each LEX-net is constructed according to a particular selection algorithm scrutinizing the particulars of the network. These algorithms – the *selectors* – are at the heart of the procedure: they must be liberal enough to allow every lexical item that might be needed for the production, and restrictive enough

to keep the generation within the semantic borders of the input FLF. The construction of these algorithms is feasible: every network is finite and all small clauses in the networks are headed by standardized predicates from a finite set. Yet, the selectors are language specific and in need of experimental validation. They represent grammatical intelligence.
So, the dedicated lexicon for the generation procedure is the union of these five sets. This lexicon is enriched with two more sets:

- the set of all lexical items that are specifically requested by one or more of the entries in the union, in particular lexically-selected 'words' in multi-word expressions or constructions;
- some selection of functional elements that may be necessary to bridge the underspecification gap between logical form and full sentence, in particular auxiliaries and functional markers without operational content, like Dutch *er*.

These sets can be constructed pre-derivationally, or addressed during generation, 'on demand'. It is important to note that in both cases the resulting lexicon for the generation procedure is a severe restriction on the full lexicon. After the dedicated lexicon has been established, generation takes off in the very same way as described in section 1.9. On the basis of a categorial hypothesis – *let there be an S* – agendas are created, executed and checked until a grammatical string complying with the hypothesis arises. The lexical space for the full generation procedure is reduced to the union of the LEX(X-net) sets and the two sets mentioned above.

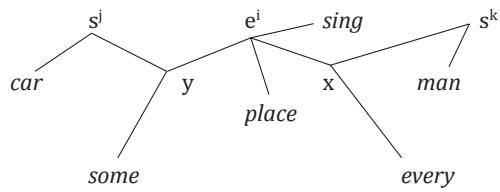
Of course, it makes sense to load the agenda in advance with the restriction that from each LEX(X-net) exactly one item is chosen for the ultimate production. Moreover, the relationship between the networks can be used as an agenda for the construction of the sentence. The relevant observation here is that for an FLF to be represented by a sentence its networks have to be connected in the following sense.

(363) FLF is *connected* iff every net in it contains at least one clause that also occurs in another net of the FLF.

Connectedness means, among other things, that the networks do not constitute a partition of FLF. If an FLF is not connected, the formula lacks semantic coherence. Sentences can be constructed only for coherent parts of the formula.
Now suppose that an FLF is connected, as (362) proves FLF (361) to be. The coherence of the FLF can be expressed in a connected graph, linking variables

and constants if they co-occur in at least one clause. Here is such a graph for (361)/ (362):

(364)



Every non-constant is a vertex supporting at least two edges. This graph can be used to steer the generation agenda*, e.g.* by starting at the densest vertex and following a depth-first strategy. In this way, the intrinsic combinatorial agenda according to which the generator operates and the semantic *desiderata* can be combined into one goal-directed procedure.
Restrictions like these on the generation procedure improve efficiency but do not disturb its essential non-determinism. In fact, finding the proper balance between lexical restrictions and the application of the grammatical agenda is an experimental rather than a theoretical matter. The generation procedure has its own syntactical backbone strong enough to accommodate additional lexical or semantic constraints. It seems beyond our grammatical reach, however, to predict the semantic fine structure of a sentence, in particular with respect to matters of scope and referential dependencies. Only a fully-generated sentence can therefore be checked for compatibility with the input.

2.7.4 Testing logical form by entailment

A sentence generated according to the protocols is bound to be grammatical and to comply with the lexical restrictions. There is no guarantee, however, that its Flat Logical Form entertains the intended semantic relationship with the input condition. In order to check whether it does, the derivationally produced SLF has to spell out as a family of FLFs. According to section 2.6.5, this family amounts to a conjunction of disjunctions. Since we assume that the input condition generally under-specifies the particulars of the sentence-to-be-produced, the produced Flat Logical Form is at least as specific as the input condition. Therefore, the intended semantic relation must be: *the produced logical form entails the input*: PrLF ⊰ IP. On the other hand, we do not want the produced FLF to contain content or information not covered by the input. The protocols of the previous sections are designed to prevent that. As a matter of fact: we take a sentence that is less specific than its semantic condition to be

a better instantiation of that condition than a sentence that is more specific. In both the following triples, the condition is more adequately and more correctly represented by the entailed than by the entailing 'translation'.

(365)	*condition*:	John hits Bill
	entailed/acceptable translation:	Bill was hit.
	entailing/over-specified:	Bill was hit by John with a stick
(366)	*condition:*	John did not hit Bill
	entailed/acceptable translation:	Bill was not hit by John with a stick
	entailing/over-specified:	Bill was not hit

Therefore, one could also require the produced form to be at most as specific as the input: *the input entails the produced logical form* or: IP ≺ PrLF. Consequently, we may want the two logical forms to be equivalent: IP ⫤⊨ PrLF. But the requirement of simple equivalence ignores the asymmetry of the directional argumentation for entailment: the input cannot be required to specify everything that a full sentence in a particular language brings along, and the sentence cannot be kept from coming with intrinsic essentials like tense and aspect which may be absent in IP, in particular when the input is not language-borne.
The analytic nature of FLF and the simplicity of its inference engine offer a solution to the problem. To see how, let us first be specific about the requirements of the produced logical form:

- no clause in PrLF introduces non-functional concepts absent in IP;
- every clause in IP is uniquely reflected in the PrLF – a bijection should exist between IP and a subset of PrLF;
- no clause in the PrLF is more specific than its reflection in IP: at clause level, the produced FLF is entailed *per clause*.

To meet these requirements simultaneously, PrLF minus the image of IP has to be submitted to severe restrictions, both with respect to the variables in the remnant clauses as well as with respect to the concepts. These restrictions are language-dependent, influenced for example by finite morphology. In Dutch, for example, PrLF cannot contain states or events outside the image of IP but it may contain tense specifications in that segment.
This yields the following definition of a successful generation procedure.

(367) Given semantic condition IP, PrLF is a correct representation of IP *iff*
(a) IP' is a conjunctive subset of PrLF such that for every clause φ in IP' there is exactly one clause ψ in IP and vice versa for which ψ ≺ φ
(b) PrLF - IP' contains only clauses that obey predefined semantic restrictions.

By the construal of Flat Logical Form, this constraint can be checked linearly. If the check is positive, the produced sentence is one of the kind we were after. If not, simple backtracking will produce another sentence.

By combining the connected graphs of type (364) and the type-driven agenda of the generator, a generation will almost inevitably produce sentences which comply with the semantic infrastructure expressed by the graph. In particular, one may expect that the input graph is an alphabetical variant of a subgraph of the one constructed on the output- FLF. This is a presupposition of condition (367). Scopal dependencies, however, cannot be arranged by manipulating the agenda. Scopal dependencies follow from the structure as a whole; they are determined under a top-down regime, whereas the constructive agendas operate piecewise and bottom-up. Scope is reconstructed post-derivationally, in the transition from SLF to FLF and ALF, rather than being constructed on-line. Its validity with respect to the input can only be tested and not predicted. Generally, clauses with independent variables entail clauses with referentially charged variables: if there is a woman that every man loves, it must be the case that every man loves one or other woman. Consequently, following (367), if a clause in the input-FLF specifies variable dependency, the output LF must specify an equivalent dependency. Again, scheme (354) can do the job.

3. LEXICON: the language's encyclopaedia and database

3.1 STORING KNOWLEDGE OF LANGUAGE

3.1.1 Lexicalism: atoms and molecules

The computational model of language presented here is knowledge-driven. It lives on the explication of grammatical combinatorics. The knowledge needed is both analytical as well as synthetic, both detailed and global, concrete and abstract, heading for the rules as well as for the exceptions. This knowledge has to be explicated and stored somewhere, and be readily accessible for parsing and generation. The explication and the storage had therefore better be efficient, with respect to operation as well as maintenance and manipulations like change and extension – learning, basically.

Accessibility of grammatical knowledge presupposes that the knowledge is retrievable on demand. And this requires a transparent storage policy. A language's immediate building blocks, the atoms of form and meaning, also known as *words*, *morphemes* or *phrases*, are the easiest addresses to handle, to retrieve and to order information. Let us call them word complexes, for the time being. There is little doubt that they are the basics of human awareness when it comes to language, as convincingly argued in Levelt (1989) and not really challenged since. These word complexes are also the best anchors for storing grammatical knowledge in a computational system. The reservoir of words presents itself as the immediate database for knowledge of language.

This point of view is quite common, but is no good unless we have an idea about what the building blocks of a language really are. This amounts to

the question at what level of language form and meaning meet. Clearly, pure forms are below that level, and texts are above it. This leaves us with morphemes, words, phrases and sentences as candidates. Sentences fail for the purpose of storing information, as they form an infinite set – information stored in an infinite database is hard to retrieve. As for morphemes, categorial grammarians in particular have argued, with some force, that the meaningful combinatorics of word formation and sentence formation are quintessentially equal (Moortgat 1988, Hoeksema 1984, and – to some extent – Wheeler 1981). This, however, is misguided, Aronoff (2008) claims: '... what happens inside lexemes is qualitatively different from what happens outside them'. He sticks with Chomsky's hypothesis on the need to lexicalize nominalizations: the semantics of morphological operations are idiosyncratic, and morphological combinatorics do not match syntactical operations at phrase level (Chomsky 1970, Rozwadowska 2006) – it must be admitted, though, that generative syntax is more abundant than the categorial algebras of Moortgat and Hoeksema. This *lexicalist hypothesis* accounts for the difference between word grammar and sentence grammar. Sentence grammar – the one that has to provide propositional interpretation – cannot steer word grammar in the following sense: if some properties of words or phrases are asked for, the sentence grammar lacks the expressive power to tell the word grammar how to provide or check these properties. We want the sentence grammar to be restrictive somehow, with less expressive power than unrestricted rewriting systems, for reasons of computational complexity. The word grammar, however, can easily be shown to operate in an unrestricted mode: it erases, it adds, it reshuffles and overrules, by necessity, in a way that we would never allow sentential combinatorics to operate. Of course we want to derive both a passive participle and its adjectival instances from some stem form of a transitive verb, but the information coming with the stem and feeding into these derivations has to be manipulated: the combinatorial properties of the participle and the derived adjectives differ essentially from the properties of other verb forms. To put it more bluntly, word grammar cannot be based on unification – it is based on destruction and reconstruction. This delicate process is what this chapter is about. Its results are complex symbols stored in a lexical database, available to the parser and the generator and disclosed by fast and precise search algorithms. These complex symbols are molecules to the syntactical and semantic chemistry of parsing and generation: they are self-contained, well-defined and unique clusters of properties and attributes relevant to sentence grammar. They are *signs* (Morrill 1994) in the sense that they combine syntactical and semantic information that is

conservatively exploited by the sentence grammar: it feeds into unification, but is not altered by serving as such.

The lexicon explicitly specifies all the combinatorial and semantic peculiarities of a language. If a certain verb form comes with a specialized meaning when combined with a certain preposition, there will be a complex symbol defining exactly this state of affairs. If that combination has several syntactic instantiations, there will be a complex symbol for each of these instances. If a certain preposition in combination with a certain type of noun gives rise to a specialized meaning on which the noun's determiner operates, there will be a complex symbol or a family of complex symbols holding that particular combination. If a certain combination has combinatorial restrictions not common to its kind, there will be a complex symbol that expresses these restrictions. The lexicon is constructive and exhaustive: it defines and delivers all phrasal molecules. At the same time, the lexicon itself is constructed, generated by a bunch of involved algorithms that account for inheritance and data economy. These algorithms, the word or phrasal grammar, distribute, add and change all kinds of grammatical specifications. They create graphs the unification of which is the final target of sentence grammar. Here is an example of such a graph: the first person singular main clause template of the separable particle-preposition-verb combination *laat+toe+tot* 'admit (someone) to (something)'; it is just one of the numerous templates representing the verb form *laat* 'make, do' in the lexicon. All values that are dependent on other values in the template and thus occur more than once are in bold face. Some mainly administrative features are left out.

(368) *lexical template for* laat ... toe *'admit'*

```
ID:A+B
HEAD:CONCEPT:admit
     PHON:laat
     SLF:admit
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at-pres
            VTYPE:bi_trans
PHON:C
PHONDATA:lijnop(laat,A+B,[arg(right(-1),0,D),arg(left(11),
         wh,E),arg(right(-2),6,F), arg(right(0),8,G)],C)
SLF:{{[H&(B+I)#J, K&(B+L)#M, N&(B+O)#P,
    Q@some^R^and(and(quant(R,some), admit~[R], event~[R],
    entails1(R,incr)), Q, entails(R,incr))&(A+B)#S],[],[]},
    and(and(and(theme_of~[S,J],agent_of~[S,P],
    goal_of~[S,M]), attime(S,T)), tense(S,pres))}
SYNSEM:CAT:s
       EXTTH:agent_of~[A+B,P]
       PREDTYPE:nonerg
       SUBQMODE:U
       TENSE:tensed
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]

ARG:   ID:B+I
       HEAD:PHON:tot
       PHON:F
       SLF:H
       SYNSEM:CAT:pp
              OBJ:indirobject_of(A+B)
              THETA:theme_of

ARG:   ID:B+L
       PHON:D
       SLF:K
       SYNSEM:CASE:obliq
              CAT:np
              OBJ:dirobject_of(A+B)
              THETA:goal_of

ARG:   ID:B+O
       PHON:E
       SLF:N
       SYNSEM:CASE:nom
              CAT:np
              FOCUS:focus
              NUMBER:sing
              OBJ:subject_of(A+B)
              PERSON:1
              QMODE:U
              SUBCAT:pron
              THETA:agent_of

ARG:   ID:B+V
       HEAD:PHON:toe
       PHON:G
       SYNSEM:CAT:part
```

The graph's representation is just for convenience: it might as well be represented by an unordered list of fully-specified paths, all headed by a node TOP (see section 3.3.1), or by a figure with labelled edges or vertices.

This graph is generated from two input data: a basic graph for intransitive particle verbs and some lemma-like specifications for the verbal complex *toe+laten+tot* 'admit', among which, the concept. The combinatorial category requires the subject to be left-dislocated and the particle to occur at the right-hand side of the verb. The head of the constituent is the verb form – 2nd person singular. The tense is specified as part of the overall Stored Logical Form. The subject argument is obligatory: its type is part of the category and its yet uninstantiated phonology ARG:PHON:E is specified in the value of PHONDATA, the label heading the linearization data. The concept it represents HEAD:CONCEPT:admit stems from the input and is integrated in the more extensive cluster of lambda terms under SLF. This ordering of lambda terms holds the full semantic architecture for the (sentential) constituent emanating from this graph. The particle does not contribute to the semantics: no semantic value for the particle *toe* would make it to the constituent level, as it has not been integrated in the value for SLF at top level. The same holds for the obligatory and selected preposition *tot* 'to'. Both the particle and the prepositional object as well as the subject are specified in the graph, as values of an ARG feature, and these specifications are open to unification with other graphs. All ARG labels have a unique ID value, for identification, linking them to the ID at top level, to be instantiated by unification, again. Under these labels, all constraints imposed on unification candidates can be specified. Above, the subject is required to be first person and a focusable pronoun. In the same vein, the subject could have been required to be animate, for example. This is a way to state that the subject of *toe+laten+tot* is animated by its occurring as such, not necessary by lexical specification: unification is not blocked by one-sided values.

It may be clear from this example that the differences between our lexical format and the practices in Head-Driven Phrase Structure Grammar (HPSG) are rather limited. They will be discussed in the next section. The base line is that in lexical entries – derived lemmas, templates, lexical graphs, complex symbols – like (368), all grammatical knowledge converges. It is the storage and the source of grammatical wisdom. With this ambition, the proper balance in the lexicon can be defined: that the lexical graphs be self-contained grammatical objects that do not need any other source of information or data in order to contribute to the formation and interpretation of sentences. This view implies that all inheritance relations are compiled into the singular lexical graphs: two graphs are lexically related if and only if they share some values, and there is no other relationship between the graphs of *sings* and

sing than there is between the graphs of *sings* and *walks*: they share certain paths. Moreover, the graphs contain whatever lexical-semantic features play a role in determining semantic structure, like the *qualia* and the patterns of type coercion discussed in Pustejovski (1993). Each lexical graph is an object in its own right. That mutual independence is an important precondition for the kind of fast retrieval for the sake of parsing and generation that we will introduce in section 3.5.

3.1.2 Delilah and HPSG

Head-Driven Phrase Structure Grammar, introduced in Pollard and Sag (1987), has become the most influential grammatical framework for computational linguistics; there were times when the framework seemed to be the shortcut for grammar-based computational linguistics. Sag (2003) has it spring off from an alternative view on generative grammar. This alternative to derivational generative grammar is constraint-based, not transformational, and strongly lexicalist. This last feature in particular requires the Delilah system to be compared to the HPSG approach, which underlies high-standard parsing systems like LINGO, Alpino and Verbmobil, all available on the internet. The Delilah system has been developed in the light of HPSG.
At the theory's home site http://hpsg.stanford.edu we find a neat summary of HPSG's main ideas. Here is the list, with each idea being shortly described (from the site) and compared to Delilah.

HPSG is a constraint-based, lexicalist approach to grammatical theory that seeks to model human languages as systems of constraints. Typed feature structures play a central role in this modelling. Some of the leading ideas of current work in HPSG are the following:

Strict Lexicalism
Word structure and phrase structure are governed by partly independent principles. Words and phrases are two kinds (subtypes) of sign.

In Delilah, words and phrases are 'created', organized, manipulated and combined by principles which are independent of the (rules of) syntax. Words and phrases are indistinguishable in the Delilah lexicon, however. All constraints are imposed and effected in the same manner, whether the constraint is of a phonological, syntactic or semantic nature.

Concrete, surface-oriented structures

'Abstract' structures (e.g. empty categories and functional projections) are avoided wherever possible, in favour of 'minimal' grammatical structures.

DELILAH's lexicon consists exclusively of phonologically non-null material. It may, however, introduce meaningful structures not explicitly licensed by syntactic units. These structures are not introduced apart from the 'real' lexicon, though.

Geometric prediction

The hierarchical organization of linguistic information plays a significant role in predicting the impossibility of certain kinds of linguistic phenomena.

DELILAH's lexicon is not hierarchically organized in the sense that some objects depend, or live, on others. DELILAH's signs are directed graphs, and the only hierarchy in the lexicon originates from this directionality. Grammatical processes are not governed by the hierarchy between lexical objects. The graph's geometry is not exploited for combinatorial or theoretical issues.

Locality of selection

According the theory of valence articulated in Pollard and Sag (1994), lexical heads select only for the synsem *objects (a kind of syntactico-semantic complex) of their complements, subjects, or specifiers. It follows that category selection, role assignment, case assignment, head agreement and semantic selection all obey a particular kind of locality determined by valence selection features. This is a kind of geometric prediction.*

In DELILAH, selection is local in the sense that all parameters have to be part of one single lexical data structure – a template – when selection is activated. Since information can only be added by a conservative form of unification, locality of relevant information is maintained through derivation. Selection is not restricted to particular objects, though.

A distinction among types of agreement

Agreement phenomena have been classified by Pollard and Sag (1994) as syntactic concord, anaphoric agreement, or pragmatic agreement. Their theory of indices predicts, inter alia, the absence of case agreement in anaphoric agreement. (...)

This distinction is not exploited in DELILAH. Concord is handled by local unification. Anaphors are resolved by additional post-unification mechanisms. Pragmatic agreement is not within DELILAH's scope.

Local encoding of unbounded dependencies

Filler-gap phenomena and other long-distance dependencies are treated not via grammatical transformations, but rather in terms of certain feature specifications that are present throughout the 'path' from filler to gap. This feature-based theory in essence predicts the existence of grammatical phenomena sensitive to such specifications, i.e. phenomena that occur only within the domain of unbounded dependency constructions.

In DELILAH too, unbounded dependencies are based on lexical specifications, namely in the lexical categories. Restrictions on the path are accounted for in the multi-modal combinatory categorial grammar that steers the unification.

Lexical cross-classification

Within HPSG, words are rich in information. Lexical information is not simply listed, however; rather it is organized in terms of multiple inheritance hierarchies and lexical rules that allow complex properties of words to be derived from the logic of the lexicon. Current research is developing extensions of hierarchical lexicons that allow lexical rules to be eliminated and linking patterns to be derived in a general fashion from semantic properties.

DELILAH's lexicon is created by a complex set of rules implementing weak forms of inheritance and generalization over lexical classes. The final lexicon, however, is flat, in that every entry is completely independently represented and unfolded. The hierarchical organization of the data structures themselves, and their graph-like format in particular, is exploited for dynamic classification in parsing and generation. The HPSG approach is certainly anchored in artificial intelligence (Sag 2003: 303 ff). The DELILAH lexicon does not embody any claim on organization of information beyond its retrieval mechanisms.

Hierarchical cross-classification of grammatical constructions

There are new proposals within HPSG to model constructions, as well as signs, in terms of feature structures. This allows constructions to be analyzed via multiple inheritance hierarchies. This in turn provides a way of modeling the fact that constructions cluster into groups with a 'family resemblance' that corresponds to a constraint on a common super-type. This strain of HPSG has thus coalesced with one conception of 'Construction Grammar'.

The DELILAH lexicon contains all instantiations of all constructions, salve coverage. All lexical templates are built in the same way, by a complex set of procedures. These rules respect common ground for the sake of efficiency, but there is no built-in hierarchy in the resulting lexicon. The family resemblance between templates is exclusively established by dynamic classification, but any resemblance can be retrieved that way. The morphological paradigm of a

verb, for example, is no different from the class of second person finite forms of transitive verbs with a right-hand subject. The way the lexicon is unfolded does not bear on its internal structure, which is plainly flat.

Obliqueness-based binding theory
Generalizations about constraints on the binding of referentially dependent elements are stated in terms of relative obliqueness (o-command), rather than configurational superiority (c-command).

The DELILAH binding algorithms use both obliqueness and configurational information. These operate post-derivationally on fully specified grammatical graphs.

Linearization theory
Current work in HPSG is exploring modes of serialization which are not based on the model of traditional phrase structure grammar (where sentences are word strings defined derivatively in terms of phrase structure). This has implications for the treatment of discontinuous constituency, allowing even the introduction of levels of linear syntactic organization that are to some extent dissociated from the combinatorial relationships among the items serialized.

DELILAH's multimodal categorial grammar is designed for dealing with discontinuity. In principle, it is capable of handling any form of discontinuous linearization that can be expressed by reference to syntactic categories. As explained in chapter 1, it is, nevertheless, quite restrictive and fairly rigid.

In general, HPSG focuses on the organization of the lexicon and the signs. It seeks restrictiveness and linguistic relevance in this domain. DELILAH has a more liberal attitude towards the structure of signs. It seeks restrictiveness in the rigid, multimodal combinatorial categorial grammar – categorial list grammar – introduced in chapter 1. The less constrained view on the lexicon is mainly motivated by the huge and yet unexplored variation in multiword expressions or extended lexical units (Sag *et al.* 2002, Poß and Van der Wouden 2005, Poß 2010). They appear in several of the following sections.

3.1.3 Words and worlds: a lexicon is not a dictionary

For computational linguistics with semantic ambitions, the most intriguing relation to be explored is the distinction between the lexicon and the encyclopaedia. For those who assume that knowledge of language must be absorbed from modeling and inspecting huge corpora, the distinction is hard. Language

is definitely not about itself. Texts feeding into automated lexical storages are about something other than language. Information derived from those texts is information about how the worlds are caught in phrases by people using a certain language. One may assume that this *is* semantics. In our view, it is not. Semantics is about the preconditions under which language can be used to capture the worlds. *What* is actually claimed, and how often, and by whom is beyond the language's architecture. The language system is neither defined nor affected by someone claiming something.

To derive a lexicon on the basis of texts acting as encyclopaedias, then, is doomed to yield an excerpt of those encyclopaedias. Though intriguing as a window on the use of language in a certain community, these excerpts cannot define the language, for the simple reason that what is going to be said or could be said but is yet unrevealed is as much language as what has actually been said. Language is the capacity of being meaningful, and the lexicon is the place where this potency is maintained and vitalized, not buried or mummified. In exactly this sense, the Semantic Web enterprise (Berners-Lee, Hendler and Lasilla 2001, Shadbolt, Hall and Berners-Lee 2006) differs essentially from natural-language semantics. Existing knowledge of all kinds is specified there as a defining characteristic of objects of all kinds. In natural-language lexicons, however, only the opportunity to refer can be sketched, not its 'extension' on any index.

The lexicon, as we envisage it, can and must contain all constraints and frames that help to guarantee a sentence's interpretability in whatever model. The lexicon does not necessarily contain factual or contingent properties of objects covered by a concept. Of course they can be added *ad libitum*, but every addition will constrain the general applicability of the lexicon, and improve its applicability within a certain model.

World knowledge – the encyclopaedia – has been widely invoked in computational linguistics to assure common-sense inference. In particular, it has been argued that natural-language processing systems should have access to information about named entities and normality in order to avoid recognizing or producing anomalies of the type

(369) The boy that a MIG painted green had been pregnant with almost every iron song before any donkey.

It is almost standard to notice, though, that this kind of sentence is a perfectly well- formed and interpretable sentence. Because of that well-formedness and interpretability, one may judge that my present world is not a model for that sentence, or accept it as a way of describing the present state of affairs.

Semantically, however, either of them is a fine result – one of the possible outcomes of an evaluation – and neither evaluation points to a problem of language. One may observe that her present interpretation of being pregnant is hard to comply with painted boys having that property or songs being iron. It is not the lexicon that has to be specific about these *qualia*: it cannot be so, unless it is restricted in advance to my little world. Note, however, that these *de re* specifications are essentially different from assuming that a predicate, when applied positively to an argument, imposes some attributes to the argument. The qualia imposed by combinatory syntax, should be either disjoined from all other lexical specifications or should not be checked under unification, *e.g.* by labeling them differently. Those selectional features alone can never be a reason to reject a sentence. Assuming graph unification, however, it is straightforward to have a predicate impose certain qualia on its arguments, no matter whether these qualia are realistic or not. But most certainly, a free language generator producing (369) is doing fine.

The lexicon is not the place where normality is defined. The lexicon does not tell us what the actual world looks like – it is not a genuine encyclopaedia. The lexicon is the place where virtual semantic space is created – the virtual semantic space we live by, to use a simple metaphor. The lexicon does not tell us how it is outside language. At best, it tells us something about language use and the human imagination – not about truth outside the gates of Eden. The lexicon does not tell us that pregnancy is reserved to female mammals.

For all kinds of purposes and applications, restrictions on the semantic space can be imposed on the lexicon, by brute force or by statistical modeling of normality. There are two basic problems for anomaly regimes, however, deserving attention. First, we have the finiteness problem. If sentence (370) is anomalous, each of the sentences in (371) is too. Yet, it seems impossible to sum up all equivalents to the relevant predicates.

(370) The male regretted being pregnant.

(371) The individual whose chromosome structure was definitely not equivalent to those of most individuals that may become pregnant regretted being so.
The regular procreator of her dearly beloved oldest daughter regretted being pregnant.
The male regretted being in a state in which till then only women had turned out to be.

Secondly, polarity can be used to declare or even define anomaly:

(372) Until now, almost no male has been known to be pregnant.

This sentence is a mere this-worldly fact, and it is certainly not more anomalous than *hot snow does not exist*. Consequently, judgments on anomaly are post-interpretational and, therefore, not part of any interpretative device proper.
If normality is not a genuine feature of the lexicon, knowledge of the world is not a feature of the lexicon either.

3.1.4 Lexical chemistry: cooking the graphs

In the preceding sections, the lexicon was depicted as a database, containing all the linguistically relevant information. It was claimed that the process of constructing this database is knowledge-driven: whatever there is to be known about individual lexical items has to be made explicit, stated and stored in a general, retrievable way. In combination, the requirements of explicitness and retrievability lead us to a distributed lexical format, in which no piece of information is hierarchically ordered above another. Phonological, morphological, syntactical and semantic information must be available simultaneously and must be randomly accessible. Though the data are not unordered (see section 3.5) they do not dwell in an a priori system. The lexicon consists of *complex symbols* and each complex symbol belongs to all classes of symbols defined by a feature value it carries at the end of its paths; the graph in (368) is as much related to other graphs with a path `SYNSEM:CAT:s` as it is related to other graphs with a path `HEAD:PHON:laat` or with a path `HEAD:SLF:admit`. Consequently, the lexicon consists of some enumeration of complex symbols, each of which is a completely independent object. The system for enumeration is random in principle, and can be chosen on purely pragmatic grounds (see sections 3.4 and 3.5).
As a matter of fact, the same holds for the way in which the lexicon is generated. The DELILAH lexicon is produced by an extremely complicated set of rules that combine 'classical' lemma data with canonical graphs and rules defining the lemma's offspring. To put it bluntly: all the graphs for all the morphological forms of the verb *laten* 'let' are created by applying the rule machinery to a particular lemma data structure and one or more 'pre-defined' canonical graphs, *e.g.* for intransitive verbs. Whether two intransitive verbal paradigms live on the same canonical graph is only a matter of efficiency, not of principle. At the end of the day, the lexicon is as flat as a pancake.
This strategy brings about the option that *inheritance* of features and values is no longer grammatically relevant. Inheritance is a major lexical concept in HPSG (*cf.* Sag 2003; ch. 15). It regulates the spread of features and values throughout the lexicon, and is part of the economy and architecture of the

grammar. If the lexicon is completely spelled out, however, in each and every painful detail, the lexicon's history or the way it came into being does not carry additional information. Even the classical notion of a lemma becomes obsolete at the level of the lexical database. Whether or not this history is subject to computational, lexicological or grammatical principles is for the record, but not relevant for parsing or generation. It certainly makes sense to scrutinize the process of lexicon formation in order to find regularities there. However, this creates an expert system for lexicography rather than a module of formal grammar. The semantics of constructions, in our view, renders any hard-boiled principalism redundant. The enormous variety of ill-known collocational effects overshadows, for the time being, the regimenting effect of pre-established inheritance hierarchies. As a matter of fact, in many constructions hardly anything inherits according to a grammatical pattern. For example, in the Dutch *way*-construction an intransitive verb – any unaccusative intransitive verb – combines with a reflexive pronoun, a *way*-type noun phrase and directional pp to create a causative *move* event where the meaning of the intransitive verbal head only occurs as the cause, and neither the *way* NP – lexically fixed – nor the reflexive – a dependent anaphor – adds to the meaning (Poß 2010). Similar observations can be made about a variant of the construction without *way*. The members of the construction are underlined.

(373) Geen enkele bankier had zich een weg naar de Raad van Bestuur kunnen golfen
No banker had himself a way to the Board of Directors been-able play-golf
'No banker could have succeeded in moving into the Board of directors by playing golf'

(374) Geen enkele bankier heeft zich de Raad van Bestuur weten binnen te golfen
No banker has himself the Board of Directors been-able inside to play-golf
'No banker was able to move into the Board of Directors by playing golf'

The constructions themselves are rare in any corpus, but the classes of constructions are manifold (Grégoire 2010). Their variety and lexical construal are discussed in section 3.4.4. The point here is that sound opportunism in cooking graphs appears to be called for, rather than esoteric principles of design. The main principle seems to be that for each construction, only one lexical element needs an additional entry in the lexicon. This element can be called the *head* of the construction: it pays the price and becomes ambiguous. The other elements of the construction are simply selected, in whatever form they occur elsewhere.

3.2 MODES OF LEXICAL KNOWLEDGE

The lexicon stores knowledge of language – all knowledge needed for processing. As such, the lexicon stores knowledge in all classical domains of grammar: phonology morphology, syntax, semantics. Moreover, it may contain relevant information on phonetics and information structure, and even pragmatics – though we consider pragmatic properties of sentences to be beyond computability. Furthermore, lexicons can be used as stores for contexts – remarkable or frequent phrases other than constructions – as is done in the better-written lexicons, or information on means of use. There is a natural boundary to the storage: relevance for processing. This keeps the encyclopaedia out, for example. It allows for ontological hierarchies to be adopted by the lexicon to the extent that they play a role in an application, or are addressed in inference.

3.2.1 Phonological form: the one-dimensional grammar

In its present state, the DELILAH lexicon contains little phonological information. Of course, a fully-fledged lexicon must contain information as to the phonological structure of words and phrases. This information is widely available, but sound is hard to process on a general platform. Phonology is the lexicon's white spot, but its absence is not dramatic for DELILAH's modeling purposes. As a consequence, the elementary encoding in our system is orthographical.

This being the case, however, phonological form is built by templates linearizing lexical units. It is the surface realization of the syntactic structure. Every template contains information on the linear ordering of its lexical components. The linearization is not completely arranged by unification: rather, unification instantiates the variables in a program that is executed post-derivationally. Below are the phonologically relevant lines from (368).

(375) *phonological network in template of* laat ... toe '*admit*'

```
ID:A+B
HEAD:PHON:laat
PHON:C
PHONDATA:lijnop(laat,A+B,[arg(right(-1),0,D),
          arg(left(11),wh,E),arg(right(-2),6,F),
          arg(right(0),8,G)],C)
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]
...
ARG: ID:B+I
     HEAD:PHON:tot
     PHON:F
          ...
ARG: ID:B+L
     PHON:D
          ...
ARG: ID:B+O
     PHON:E
          ...
ARG: ID:B+V
     HEAD:PHON:toe
     PHON:G
          ...
```

The feature PHONDATA is valuated by a formula `lijnop/4`. The predicate operates on the orthographical representation of the phrase's head ('laat'), the template's index and a list of specifications on the four arguments, and returns its value ('C') to the attribute PHON. The specifications of the arguments provide the relevant data: the mode of composition for each of the arguments, its own 'phonology' – yet uninstantiated – and an indication of at which side of the head and at which relative distance to the head the argument occurs. All this information on the linearization of arguments is derived from the TYPE value when the template is constructed.

Even though the phonological value of an argument's head can be specified as part of the selective construction – as is the case twice in (375), the argument's phonological form itself can be complex. When instantiated, an argument comes with its own PHONDATA, to be executed in order to establish the argument's phonological form. Because of this recursive aspect and because in the lexicon, the patterns of discontinuity are largely unpredictable, lin-

earization cannot catch up with unification in an agenda-driven generation procedure. Therefore, `lijnop/4` can only be executed when all relevant linearization data are available: after the derivation has been completed. The procedure itself is described in chapter 1 as the operation `makestring/3`.

As a side effect, the independence of this linearization procedure offers an opportunity to check the coherence between the recognizing and the generating grammars. In an ideal world, the linearization result should be equal to the input string. If not, the linearization procedure must be assumed to have bugs.

3.2.2 Morphology: the combinatoric guide

The lexicon is word-based, rather than morpheme-based. This is a choice of system. DELILAH does not (yet) entertain a morphology, a word grammar. It specifies, though, systematic morphological variation in order to produce correct labels for complex symbols. These forms are produced by a set of rules operating on the orthography of a 'main' word and producing all the forms that can be derived. For verbs, for example, all finite and infinite forms, including adjectivally and adverbially used ones, are generated. Each of them is represented orthographically and marked for its role in the verbal paradigm: its templates will be generated on the basis of that role.

For the verb *verbranden*, 'to burn' the following list of forms will be produced automatically – not every form, though, is acceptable to every taxpayer.

(376) verbrand: *1person+singular+pres / imperative+singular / 2person+singular+pres+postverbalsubject / perfective participle / passive participle / adjective*
verbrandt: *3p+sg+pres / 2p+sg+pres+preverbalsubject / imp+pl*
verbranden: *pl+pres / infinitive / noun*
verbrandde: *sg+past*
verbrandden: *pl+past*
verbrandend: *pres part*
verbrandende: *adjective / noun*
verbrandenden: *noun*
verbrande: *adjective / noun*

For each of its labels, each form can be associated with multiple templates, differing from each other in combinatoric properties. They happen to share the concept *burn*, however. Consequently, a mono-concept lemma like *verbranden* may end up with sixty or eighty different templates in its transitive

branch. In its unergative reading, the verb will produce a similar number of templates. In principle, all these templates are different objects. Accidental similarities caused by situational convergence of different rules are filtered. As a consequence, every template is a unique attribute-value matrix (avm). Lexical redundancy is accounted for by the fact that across the lexicon all verb forms with a certain label are built into complex symbols by a single rule cluster. Instead of defining, *e.g.*, the affix *-t* in isolation as a present tense morpheme, the lexicon will treat all verb forms showing this affix with the same '2-and-3p-sg-pres' module, just as a different module accounts for all participles.

Other categories with morphological variation are instantiated in a comparable way. The rules producing the forms are complex but not deep. They had better be fed by phonological rather than by orthographical input – that is an option. The rules producing the templates of the distinct forms, on the other hand, carry and reflect much of the syntactic subtleties of Dutch. In particular, all variation in order – including systematic inversion – is encoded by the lexical rules in all kinds of attributes, like TYPE, PHONDATA, and ARG:SYNSEM, and as such is distributed over a template. The lexical rules embody the generalization, up to and including exceptions and constructional particularities. In a finite way, they must express all that can be known about the concatenation of forms in a particular language. Moreover, these rules express that knowledge in a very explicit way. Their manipulation of templates – they are real transformations (see section 3.4) – is complex but transparent and reconstructable, and every piece of knowledge is returned as a particular value to an attribute.

In this sense, the morphological paradigms represent the backbone of the combinatoric engine; they index the syntax.

3.2.3 Syntax: the unification agenda

3.2.3.1 Categories as data structures

Strictly speaking, DELILAH's categories encode three types of information:

(377) (a) for each phrase or constituent, its proper parts
(b) for each part, its position relative to the head
(c) for each part, the conditions under which its category can be composed

The way in which this information is derived and used is the subject of chapter 1. Here we consider only how categorial information appears in the lexical template.
The rules of lexical construal associate every complex symbol with exactly one category. The category of a template is completely determined by other attributes. This information is retrieved on construal and assembled in the data structure that becomes the value of TYPE:

(378) (a) Head\ LFlag~[TopLeft^Mode$_i$,..] / RFlag~[TopRight^Mode$_j$, ..]
(b) Head\ 0~[TopLeft^Mode$_i$, ..] / 0~[TopRight^Mode$_j$, ..]

Each of the two lists is finite, with a low cardinality, and may be empty. The values for Head, TopLeft and TopRight come from a restrictive set of categorial literals, among which are *s*, *np* and *vp*. The values for Mode are indices, again, from a restricted set. The sets of categorial literals and of modes are disjoint. LFlag and RFlag are dynamic indices, defined by the rules of grammar, but in the lexicon their value is always 0; (378)b reflects the lexical state of a category.

As a matter of fact, the composition modes Mode$_i$ referred to in (377)c have little impact on the template's structure. The modes occur as part of the TYPE value where they are introduced as indices on argument types. They are addressed by the rules of syntax – both in parsing and generation – but are neutral with respect to unification; they are imposed by the constituent but not specified in the argument's sign. Consequently, they are not used as constraints on unification. Their role is merely to regulate the sequence of unifications that make up a sentence's analysis.

The constituent structure itself is reflected in the structure of the template. For each proper part, the template specifies a sub-graph as a value of an ARG(x) attribute where x is a unique and identifying index – this index is represented only when convenient. The sub-graph identifies the constraints imposed on the argument phrases by the mother construction, among which is a specification of the argument's type occurring as a literal in the matrix category. The relevant part of template (368) is represented below, with the complex indices on the ARGs, the mode attributes ('flags') and the direction/distance attribute indicated; the latter were left out of the representation in (368).

(379) *syntactical network in template* laat ... toe '*admit*'

```
ID:A+B
...
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]
...
ARG: ( ID:B+I
       SYNSEM:[... CAT:pp ... DIR:right(1)...FLAG:6] ... )

ARG: ( ID:B+L
       SYNSEM:[.. CAT:np .. CASE:obliq .. DIR:right(5) .. FLAG:0] )

ARG: ( ID:B+O
       SYNSEM:[... CAT:np ... DIR:left(11) ...FLAG:wh] ... )

ARG: ( ID:B+V
       SYNSEM:[... CAT:part ...DIR:right(10)...FLAG:8] ... )
```

These argument sub-graphs are the proper target of unification, to the extent that the unification of two templates is defined as an essentially antisymmetric operation.

(380) Two templates T_1 and T_2 unify *iff* there is a sub-template ARG(x):T_3 of T_1, and T_2 and T_3 are compatible.

Whether or not two templates are selected for unification is decided by rules of grammar which are also antisymmetric by nature. Thus, unification is the main process of natural language grammar, according to Kayne (1994).

3.2.3.2 Features and values

If we assume that unification of templates is basically antisymmetric, an argument's *attribute-value matrix* in a lexical template generally specifies two regimes: it states which feature-value pairs the mother construction imposes on the argument and it states which values the argument, when instantiated, is supposed to deliver to the construction values. All and only specifications to these effects are needed in the sub-avm. The nature of the value indicates its function: if it is a variable, the feature is delivered, and if it is a constant, the feature is imposed. Delivered values will also occur outside the sub-avm. As an example, take the lexical template for the simple intransitive *werkt* 'works' with just one (subject) argument specified. In that sub-avm, every feature is

indicated for being imposed (↓) or delivering (↑); a few redundant bookkeeping features have been left out.

(381) *lexical template of* werkt *'works' with mood of features specified*

```
ID:A+B
HEAD:CONCEPT:work
     PHON:werkt
     SLF:work
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:3
            TENSEOP:at-pres
            VTYPE:nonacc
PHON:C
PHONDATA:lijnop(werkt,A+B,[arg(right(-10),0,D)],C)
SLF:{{[E&(B+F)#G,H@some^I^and(and(quant(I,some),work~[I],
    event~[I], entails1(I,incr)), H,
    entails(I,incr))&(A+B)#J],[], []},
    and(and(agent_of~[J,G], attime(J,K)), tense(J,pres))}
SYNSEM:CAT:s
       EXTTH:agent_of~[A+B,G]
       PREDTYPE:nonerg
       SUBQMODE:L
       TENSE:tensed
TYPE:s\0~[]/0~[np^0#B+F]
ARG:ID:B+F
    PHON:D ↑
    SLF:E ↑
    SYNSEM:CASE:nom ↓
           CAT:np ↓
           NUMBER:sing ↓
           OBJ:subject_of(A+B) ↓
           PERSON:3 ↓
           QMODE:L ↑
           THETA:agent_of ↓
```

The interaction of imposed and delivered features indicates the minimum and maximum of the class of features that have to be specified in the lexicon. A template does not need more specification than what can be imposed on it, and a template must not be less specific than what it is supposed to deliver. As an example: for an NP-template to be unified with (381) it does not need to be specified for the imposed features, but it must come with values for the features to be delivered. As a consequence, an NP-template needs to have only phonological content (PHON) and underspecified semantics (SLF) to qualify as an argument.

Knowledge of syntax amounts to specifying agreement in a proper balance of imposed and delivering features. Moreover, it assumes an antisymmetrical relationship between the construction itself – imposing values – and the constitutive parts – delivering values. The lexicon is phrasal, and organized by constructions.

3.2.4 Semantics: ultimate knowledge of language

3.2.4.1 Use of Concepts

Talking about the meaning of words, no *dictum* is more relevant than Frege's famous arithmetical statement:

(382) Nur im Zusammenhange eines Satzes bedeuten die Wörter etwas
Only in the frame of a sentence words mean something
(G. Frege. *Grundlagen der Arithmetik* (§ 62))

The proper translation may lead us into debates on in- en extensionality beyond the scope of this work. The quintessence, however, is that the denotation of words and phrases lives on the propositions in which they are framed, and not the other way round. All there is to the meaning of words in isolation, *e.g.* in a lexicon, is abstract and non-derivable. In a Carnapian approach to intensionality, words in isolation at best represent a function that delivers a referent when the word is *used* (in the framework of a sentence) to refer in a particular world. Unfortunately, human beings are excellent in recognizing reference and in converging on reference, but extremely poor in checking, not to speak of computing the Carnapian function underlying that referentiality. That is, we do not have the slightest problem in understanding your sentence

(383) That man is drinking a beer

even without checking what you mean by *man, drink* or *beer* and without having to agree with you on these intensionals. In fact, we understand your sentence, even if we turn out to disagree strongly on any concept used. People hardly ever check each other's concepts. And if they do, there will be a debate or a miscommunication.

Intensional functions are hard to define, for two reasons: we need other undefined intensional functions, and we do not know whether our function is correct – if we had a criterion here – and what it covers. Lexicographers tend to

compensate for that by shifting meaning from the function definition to the labels of relations between undefined concepts: *synonymy, antinomy*, *hyperonomy* and so on. The problem is, however, that these relations are extensional. A *chair* may be a piece of *furniture* in many situations but it is not so by definition – there is no intrinsic linguistic relationship between the two concepts, just as there is no intrinsic linguistic relationship between the concepts *whale* and *mammal, drink* and *eat*, *love* and *hate*, *city* and *capital*; the only relationship between the pairs is that in your particular world some phenomena may be both or neither or just one of them. Or, to put it in other words, the junctions in (384) are neither ungrammatical nor un-interpretable; but we must assume that the brain behind the sentences does not take roses to be flowers.

(384) John brought her roses and flowers.
John will bring her roses or flowers.

Neither is the next sentence false by definition:

(385) This rose is not a flower.

Yet, most lexicons will state that a rose is a flower – and there is a certain tendency among the living ones to see it that way. Generally, what is a rose will be referred to as a flower, but that is a statement of extension. It belongs in the encyclopaedia of the normal world conceived by a headmaster, but it does not express knowledge of language beyond its truthful use in the accidental world we live in.

Even though we may assume that anyone using *rose* or *love* applies some intensional function, there is no good reason to assume that we can *define* that function intensionally, as shared knowledge of language users: there are quite a few nouns we use felicitously without knowing their encyclopaedic status. Categorizing them conceptually would not express knowledge of language, not even knowledge of language use, but knowledge of this-wordly normality.

The intensional gap we can observe in language use may even be considered to be the engine of reference. When we say something in a certain situation, we refer to a state of affairs that has never been referred to before. We do so by using words from a finite set. If their intension were to be fixed, we might miss the novelty of the concept. This, of course, is the message of Lakoff and Johnson (1980): meaning shifts along cognitive patterns like *love is war*. The shift is not random, but its peculiarities are unpredictable.

Frege's statement (382) raises the question how words can get meaning amidst a complex network of interdependent phrases. The answer here is that structure conveys the message, and that structure is induced by functional elements. The structural elements in (383) are *that, is* and *a*. They are not even concepts: their meaning is fixed, but far from trivial and not easy to reconstruct. Yet, for functional terms, a lexical meaning can be established, as Montague (1972) showed.

For all other terms, however, all that can be said is that they represent some concept. This concept is beyond rigid formalization. This is not meant to disqualify heroic approaches to explain and capture lexical variation like Mel'čuk's *explanatory combinatorial lexicon* (*e.g.* Mel'čuk and Zholkovsky 1984), Sowa's conceptual *graphs* (Sowa 1984), Pustejowski's *type coercion* (Pustejowski 1993), Gärdenfors' *conceptual spaces* (Gärdenfors 2000) and many other efforts by semanticists, lexicologists and lexicographers through the ages. The point is that neither the properties of concepts *an sich* nor the relations between concepts are computable in any interesting sense. The source of this intractability is that the semantic properties of words or phrases are undecidable. For every non-trivial property P that is claimed to be semantically relevant to a word W, each of the following statements is true:

- we cannot tell whether P is relevant to W in all (relevant) contexts in which W is used meaningfully;
- we cannot tell whether there are other properties Q that might be semantically relevant to W in some context in which W is meaningfully used;
- we cannot tell whether such properties P and Q are independent;
- we cannot tell whether P is a characteristic property of W.

And '*we cannot tell*' means that there is no empirical practice to validate P, in exactly the sense in which Chierchia and McConnell-Ginet (2000: ch. 1) refer to the *empirical domain of semantics*: converging judgements of language users. We may gather converging judgements on lexical meaning in specified contexts. But an old observation is that any noun N that is claimed to be characterized by a property P may be used meaningfully in a sentence which amounts to *Some N is not P*. Therefore, even a converging judgement on P for N in a context neither limits nor characterizes N. What we can do, though, is measure its use and qualify its 'normality' or default-value based on this. Even for this purpose, however, we may run into circular traffic: at a certain moment some semantic grounding must be supplied, of the same nature that we are after. In this respect, it is wise to realize that vagueness, meaning transition and metaphori-

cal meaning is not exceptional; rather, it is the rule. As for vagueness, it must be stressed that not only predicates for which some metric exists are vague, but all predicates are: verbs, nouns, adverbs, and adjectives. The paradoxes discussed and analyzed in Van Deemter (2009) are perfectly general.

As a matter of fact, Chaffin and Herrmann (1988) argue correctly that the sets of relations with which conceptual networks can be built themselves require independent empirical justification – that is different from empirical validation, which was denied to lexical semantics mentioned earlier. We do not believe that it is the computational linguists' task to try and define these networks, although one may exploit computational tools to describe the way they are used. For all sorts of applications in specific domains or for specific purposes, the elementary concept may be substituted by more elaborated objects, like ontologies or lemmas from an encyclopaedia. Alternatively, concepts themselves can be seen as a molecular structure of semantic atoms, in the spirit of Wierzbicka's *Natural Semantic Metalanguage* (*e.g.* Goddard 2002). Such an enterprise is also beyond computational modelling of natural language. It amounts to implementing a theory of lexical meaning. In the same vein, Ebeling's (1978) complex theory of signs and semantic features may appeal to structuralist linguists, but his following dictum may prevent computational lexicographers from trying to implement it with finite resources: *even the most subtle semantic aspect can be described as a feature of something* (Ebeling 1978: 109).

The *concept*-field in DELILAH's lexicon is, as a consequence of this reflection, valuated with a link, a label or a hook, rather than by content. The lexicon is semantically and conceptually open.
Note, however, that synonymy is a decision not on concepts, but on language. Two phrases may mean the same, even when that meaning is not trivially given. This *intensional synonymy* is rather the exception, as it seems. At least, it is not wise to mix ontologies and meaning.

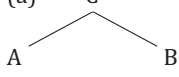
3.2.4.2 Types and lambdas

The *Empirical Domain of Semantics*, which Chierchia and McConnell-Ginet (2000) refer to, comprises those semantic properties that relate to propositions and, more generally, the way in which words and phrases combine into verificational statements. It is only natural that the semantic categories in this domain are closely connected to syntax. We have to assume that shared interpretations – another way to introduce empiricism – are induced by overt and covert syntax, to the extent that the judgements converge: if interpretations

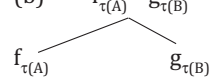
are stable and shared, they cannot be accidents. Compositionality is the name of the game: the meaning of a phrase is a function of its construction and the meaning of its parts. In order to model compositionality properly, however, the construction of a phrase, the semantic contributions of its parts and the way meanings combine must be made explicit. The ingredients for the compositional bakery are stored and labelled in the lexicon: categories, types and functions or, better, types relating categories to functions. If they have not been lexically declared, they will not appear anywhere.

Type theory, when it is applied to natural-language analysis, provides exactly the coherence of form and meaning that underlies the empirical dimension of natural- language semantics. Types index functions, and functions we need in order to construct the semantic network prompted by a sentence. The basic pattern is given below. If two syntactical units of categories A and B combine and produce a unit of category C, then C *means* the composition of two functions $f_{\tau(A)}$ and $g_{\tau(B)}$. Here t(X) is the type corresponding to X and composition passes into application if the type of the secondary function in the composition is the domain of the primary function: $f_{\tau(A)} \circ g_{\tau(B)} := f_{\tau(A)} (g_{\tau(B)})$ iff $\tau(A) = <\tau(B), \sigma>$. Moreover, given the types of A and B, only one of the two compositions can be valid: $f_{\tau(A)} \circ g_{\tau(B)}$ is defined iff $\tau(B) = <\alpha,\beta>$ and $\tau(A) = <\beta,\sigma>$ and $\alpha \neq \sigma$. Thus, the types maintain – if not embody – the fundamental antisymmetry of syntax (Kayne 1994); see also chapter 1.

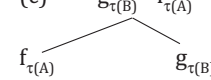
(386) (a)



(b)



(c)



Every template in the lexicon containing a combinatorial category comes with a semantic type, but the relation between categories and types is not functional: categories are not exclusive for types, nor types for categories. Therefore, the type of a phrase is defined and decided in the lexicon. A phrase is of a certain type only if the phrases' template gives rise to that type.

Yet, semantic typing in the DELILAH lexicon is implicit rather than explicit. Two reasons justify this obscurantism:

- not every syntactic argument is reflected in the meaning of its phrase;
- the syntax is rigid (see chapter 1), but the frozen syntactical combinatorics do not necessarily reflect the associated composition.

The first argument for implicit typing relates to the abundance of phrases the syntax of which does not reflect their semantic structure. To put it clearly: the

kick in *to kick the bucket* heads a transitive construction, but the corresponding function is not applied to the meaning of *the bucket*. Explicit typing would have to be overruled explicitly, which makes no sense.

The second argument is of a systematic nature. Functions are modelled by sets, as a function from A-objects to B-objects is a homomorphism from set A into set B. So, functions themselves form sets, since they can be arguments to functions typed otherwise. From algebra we know that each member of a set can be identified by its unique ultra-filter: the set of subsets containing that particular individual. That ultra-filter is a function again, of a predictable type: if the individual is of type α, its ultra-filter on the domain of objects of type α is of type <<αt>t>, for *t* typing the boolean values. In the Lambek-calculus of type-logical grammar (Moortgat 1988), this is an instance of a general theorem called *type raising* or *type lifting*.

For semantic purposes, but not for syntactic combinatorics, it can be useful to exploit the ultra-filter, rather than the original function. Montague (1972) exploited higher-order functions with flexible combinatorics to deal with scope variation. In the DELILAH grammar, the higher order functions are exploited to prepare constituent meanings for the scoping algorithm (see chapter 2). Nominal objects in particular are syntactically 'eaten' by their non-nominal licensers, like verbs, but semantically the objects scope over the licensor's meaning in order to allow for scope variation between its arguments. More bluntly, a subject is a syntactic argument to its verb, but semantically the division of labour is reversed: the subject is the primary type in a composition with the verb. This incongruency is deep and essential: an eventive verb may assign a theta-role to its subject, syntactically, but the subject's algebra determines whether (or not) some individual bears the relation expressed by the theta-role to that event: *no woman* holds the agent position in (387) but no woman was the agent of song singing if the sentence is true.

(387) No woman sang a song

In order to solve the paradox, generative syntactic theory assumes that the subject phrase does not carry the theta role, but heads a *chain* of positions of which the theta role's position is the tail. Assuming the lack of correspondence between syntactical and semantic structure to be real, nevertheless, the normal categorization and typing of a transitive verb and its arguments runs as follows.

(388) *syntactic categories*

finite transitive verb:	s\np/np	or	t\e/e
subject:	np	or	e
object:	np	or	e

semantic types

transitive verb:	<e<et>>	λ-term: $\lambda x_e.\lambda y_e.R_{eet}(x_e,y_e)$
subject:	<<et>t>	λ-term: $\lambda P_{et}.\wp_{ett}(P_{et})$
object:	<<et>t>	λ-term: $\lambda P_{et}.\mathscr{R}_{ett}(P_{et})$

Clearly, the arguments' types are *lifted* in the sense explained above, in order to allow for them to be composed with the verb. Lifting here is not deductive and procedural, as in standard categorial grammar (*e.g.* Moortgat 1988), but lexical and declarative and, thus, not recursive.

In our system, the higher-order functions are in the store of the lower function, and the arguments are in the store of the head, as explained in chapter 2. Consequently, the lexical template(s) of a construction's head specify

- its syntactic arguments, in a path with label `ARG(X+Y+Z):...`
- their syntactic categories, in the path `ARG(X+Y+Z):SYNSEM:CAT:...`
- their λ-term (generally as a variable), in the path `ARG(X+Y+Z):SLF:...`
- for every argument's λ-term:
    * its position in the stores at Stored Logical Form, *i.e.* a position in the value in the path `SLF:{{[...] ...}, ...}`
    * the variable in the body of SLF to which the λ-term is converted.

Below are the relevant lines from template (368).

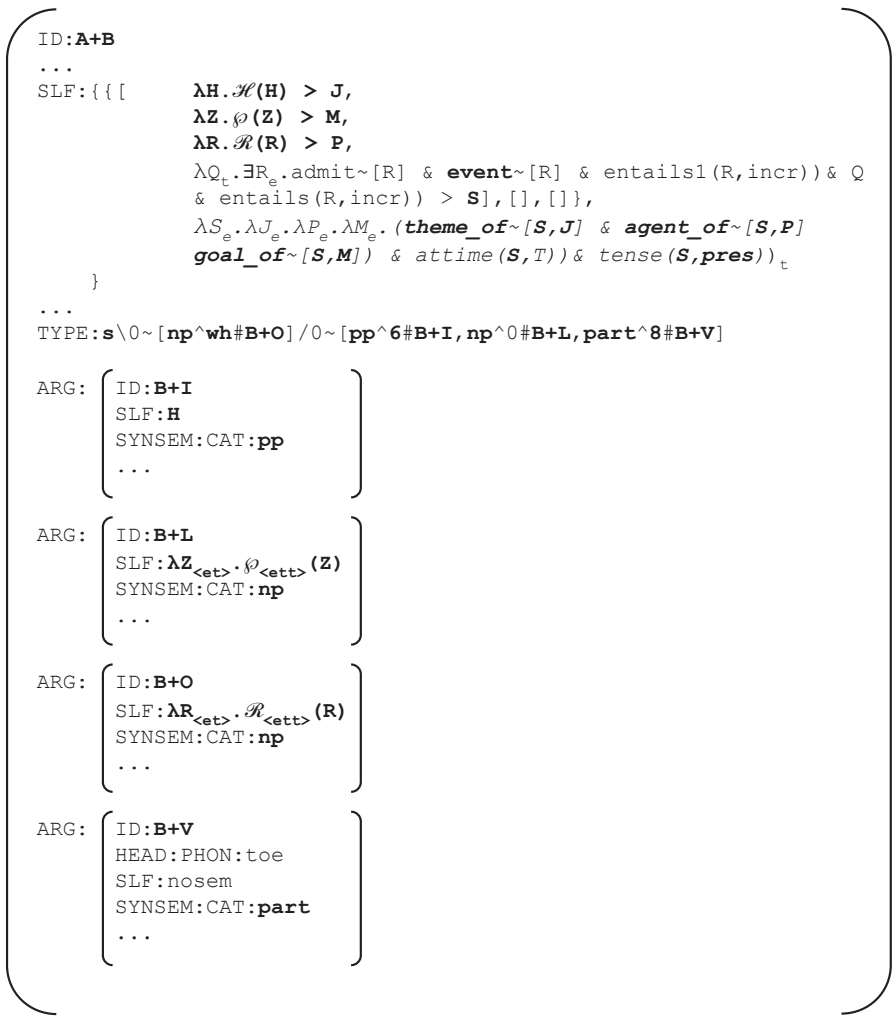
(389) *syntactic-semantic network of template* laat ... toe *'admit'*

```
ID:A+B
...
SLF:{{[H&(B+I)#J, K&(B+L)#M, N&(B+O)#P,
    Q@some^R^and(and(quant(R,some),admit~[R],event~[R],
    entails1(R,incr)),Q,entails(R,incr))&(A+B)#S],[],[]},
    and(and(and(theme_of~[S,J],agent_of~[S,P],goal_of~[S,M]),
    attime(S,T)),tense(S,pres))}
...
TYPE:s\0~[np^wh#B+O]/0~[pp^6#B+I,np^0#B+L,part^8#B+V]

ARG: [ID:B+I
      SLF:H
      SYNSEM:CAT:pp
      ...]

ARG: [ID:B+L
      SLF:K
      SYNSEM:CAT:np
      ...]

ARG: [ID:B+O
      SLF:N
      SYNSEM:CAT:np
      ...]

ARG: [ID:B+V
      HEAD:PHON:toe
      SLF:nosem
      SYNSEM:CAT:part]
```

The relevant chunks of knowledge are in bold. The top SLF contains the following information. The λ-term that interprets the agentive argument internally converts to – or: is bound to bind – the variable P in the body. The λ-term that interprets the thematic *np*-argument internally converts to the variable J in the body. The semantic contribution introduced by the *pp*-phrase and likely to be equal to the meaning of the complement of the (semantically neglectable) preposition, internally converts to the variable M. No semantic contribution of the particle is envisaged at top level. Moreover, the event quantifier itself is made explicit and stored, binding the variable S. This quantifier is introduced by the phrase itself, as an argument to its finite head, the body, which specifies the semantic space inherent in the verb's finiteness; from a morphological point of view, the verb is taken to be an argument of the inflection.

The implicit typing of the λ-terms in (389) is expressed in the following way: the type of the meanings K and N above is that of an ultra-filter on the individuals to be assigned to the variables M and P, respectively. Implicitly, M and P are of type *e*, marking the semantic arguments of the event *admit*. As a consequence, both K and N are of type *<<et>t>*. In the same vein, the body of the top SLF – italicized in (389) – represents a quadruple abstraction over the variables M, P, J and S, the last one being internally 'bound': the body is of type *<e<e<e<e,t>>>>*. Lambda abstraction is not made explicit in the representations, since lambda conversion cannot be executed properly under Prolog (Pereira and Shieber 1987). In standard notation, however, template (389) would look like this, with essential types specified.

(390) *syntactic-semantic network of template* laat ... toe *'admit'* (normalized)



The relevant composition protocol supporting the necessary β-conversions, which in the DELILAH system is applied post-derivationally (see chapter 2), is as follows, for appropriate terms of type <<αβ>β> and <α^nβ>, respectively:

(391) $\lambda P.\wp(P)_{<\alpha\beta\beta>} \quad \circ \quad \lambda x_1. \ldots \lambda x_n.\ R(x_1, \ldots x_n)_{<\alpha^n\beta>} \quad \Rightarrow$
$\lambda x_1. \ldots \lambda x_{n-1}.\ \wp_{<\alpha\beta\beta>}\ (\lambda x_n.\ R(x_1, \ldots x_n)_{<\alpha\beta>})_{<\alpha^{n-1}\beta>}.$

Thus, the DELILAH lexicon provides well-typed but underspecified semantic structures, to be instantiated by unification. These structures carry the final propositional content. In this sense, knowledge of the meaning of a sentence is established lexically, by means of combinatory semantic categories.

3.2.5 Information Structure: to the limits of decidability

According to Frege's *dictum* in the preceding section, the sentence is the main theatre of meaning: effective meanings in natural language are propositions; whenever an expression means something, it is interpreted as a proposition. Propositions are analyzed, from the very dawn of semantics, in terms of truth and validity. This point of view has proven extremely fruitful. It brought us to syllogistic logic, intensionality and generalized quantifiers, to mention just a few landmarks. Yet, sentences in natural language – the prototypical propositions – connect to other sentences in a much more complicated way than by mere juxtaposition. First of all, the context of a sentence determines and delimits all kinds of model indices like time, reference, modality and focus. Second, it may also provide the proper semantic contribution of phrases, as in the case of tense and anaphora. Still, it is wise to realize that the question whether or not a sentence fits into a certain context may be undecidable, for the simple reason that a context cannot exclude any continuation. Not every continuation may be felicitous, but a continuation of a context is only *uninterpretable* if the continuation is syntactically elliptical, and the context does not provide a proper *filler*.

(392) *context*
John is walking the dog
continuation
And me a book, sometimes. (*)

In all other cases, the propositional interpretation of a continuation may not be a proper update to the context from a cognitive point of view, but we can tell so only after (partial) interpretation. Contextual anomaly presupposes inter-

pretation. Whether or not a particular sentence updates the context properly, is beyond computation, though. Interpretation would require the set of continuations to be enumerable. The consequence is that it makes no sense to enrich the lexicon with information for selecting or testing proper continuation. As far as we can see, informational 'ungrammaticality' at text level – inappropriateness – cannot be defined in any interesting, *i.e.* finite manner.
Information structure that can be encoded, however, is the grammatical substrate to focus. It comes in two forms: certain (relative) positions have focus by default, and certain phonological units are too weak to occur in those positions. For example, in Dutch the position at the left periphery of the finite constituent – SPEC-CP in generative terms – is in focus when not occupied (or blocked) by a subject (Zwart 1993). Non-subject constituents occurring in that peripheral position can be marked as being focalized. In the DELILAH categorial grammar, this position is detected. Combinatory categories are specified in the lexical templates. As a consequence, focus is predicted lexically for non-subject constituents in left-peripheral position. At the same time, phonologically weak pronouns must be prevented from occurring in that position. They are lexically marked as non-focal. So, they do not unify in that position, in parsing or in generation.

Yet, the amount of information structure to be captured explicitly in the lexicon is small. It is not nearly enough to cover the needs of discourse analysis. For example, at the level of the lexicon, a general decision whether a certain anaphor is bounded within the sentence is out of reach. By now, there are many strategies for detecting antecedents and ranking the candidates (for an overview, see Mitkov *et al.* 2001), but none of these relies on lexical information.

3.3 UNIFICATION: POWERING GRAMMAR CONSERVATIVELY

3.3.1 Procedures and specifications

A template – the lexical data structure we exploit – is basically a list of properties of phrases that are interpreted as constraints on their combinatorial use. When we state that a phrase is of category *np*, this statement restricts its occurrence to those positions which require or allow for an np. When a

construction imposes a category to one of its proper parts, this statement restricts the choice of candidates for that proper part. In our system, these constraints are put to work by a process called *unification*. Sag (2003: 56) has an interesting footnote on the difference between constraints and unification:

> *Theories of the sort we describe in this book are sometimes called 'unification based' but this term is misleading. Unification is a method (i.e. a procedure) for solving sets of identity constraints. But it is the constraints themselves that constitute the theory, not a procedure we might use with them. Hence, we will refer to the theory of grammar we develop, and the class of related theories, as 'constraint-based', rather than 'unification-based'*.

The almost classic division between representations and processes seems to be at stake here. The author chooses to have the representations prevail in characterizing the approach. From our point of view, however, the process too is an important part of the message. As a matter of fact, we choose and present the constraints in such a way that they can be resolved by unification, by *graph unification*, to be precise. We consider unification to be the name of the game played in combinatorial grammar. It embodies an important logical or algebraic device: negation or complementation. The logic of feature structures leaves no room for 'normal' negation or complementation. In the logic for feature systems as presented in Kasper and Rounds (1990), classical negation does not occur. The main reason for this is that a feature structure is – at best – a partial description, by definition. To negate, in a classical way, either the occurrence of a feature or a value to it amounts to expressing complete knowledge of the object that the feature structure describes. Complete knowledge, though, is not something any linguist should pretend to have. When, for example, we specify that *thing* is valued 'inanimate' for the feature *animacy*, we claim that there are occurrences of the word where it has or expresses that value. It is an existential statement. To use negation here would be universal: under no circumstances is the word 'animate' as for *animacy*. We can hardly claim to have access to this kind of certainty, as the word can occur in infinitely many contexts. Universal statements about properties of words or phrases in all possible contexts are beyond computation or experience anyway. Those statements would be essentialist or intensional.
Rounds (1997) argues that Carpenter (1992) deals with negative valuation by typing feature structures. Carpenter, however, must assume that the feature structures themselves are organized and can be typed – an assumption apt for HPSG as in Sag (2003), but not for our purposes. In our view, the constraints subject to unification do not impose a theory of language; in a certain

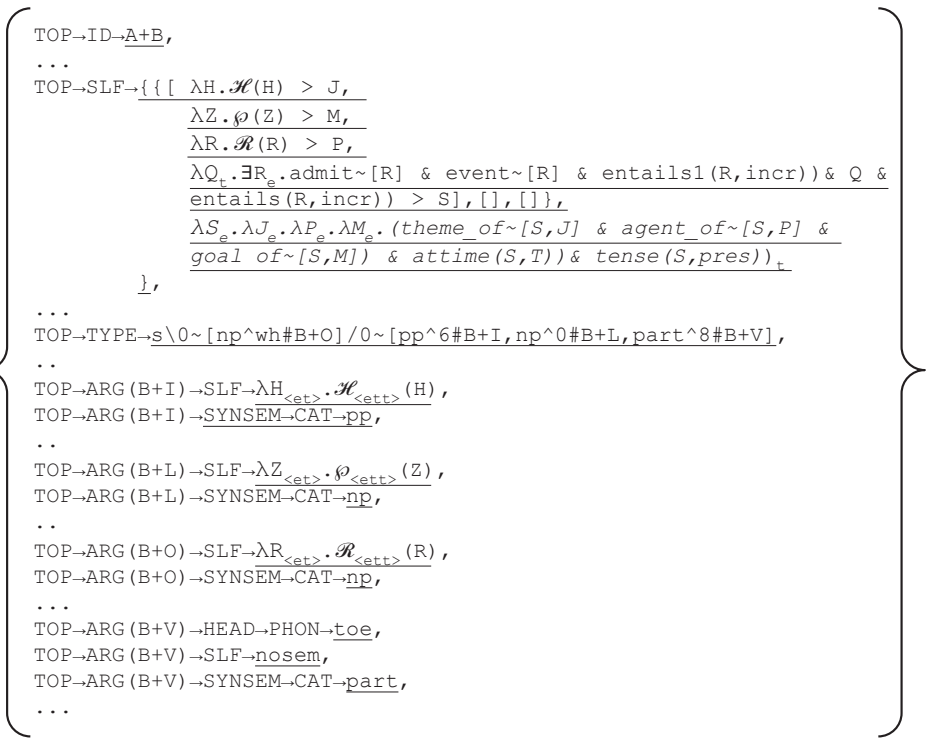
sense, our constraints are tools, rather than principles. They are just indices: handles to get things ordered properly.
When no complementation is available in the matter, the complementation must be in the procedure, in order to acquire a decent logic for our grammar. Unification offers this Boolean procedure. If two 'positive' objects meet under unification, their lack of being complemented is lifted by the openness of the question of subsumption: can the one be subsumed under the other? That is, we cannot tell whether or not a certain value or a certain feature must or cannot be assigned to a certain structure, but we cán tell whether or not a certain structure is compatible with another as it is. To put the matter in yet another perspective: unification does to feature structure what – in Frege's famous words – the proposition does to words: it creates a meaningful environment. Outside the gates of unification, a feature structure does not mean a thing.

This said, the operation mode of unification – basically a very fundamental mathematical procedure – must be defined. Firstly, we define a weakly connected directed acyclic graph by assuming a set of labels L for the vertices connected by directed edges such that:

(393) *Connected graph*
- (a) every vertex is labelled by exactly one label
- (b) there is exactly one vertex without incoming edge (*top: rootedness)*
- (c) there is a class of vertices without outgoing edges *(bottom)*
- (d) two vertices are connected by one edge at most
- (e) every bottom is labelled by a term

Clearly, a weakly connected acyclic graph can be represented as a finite set of paths from the top vertex to a bottom, where a path is a tree with a labelled top and a labelled bottom and in between nodes with exactly one in- and exactly one out-going edge. The weakly connected acyclic graph is equivalent to a forest of path-like labelled trees. This justifies our representation as a template, with every bottom spelled out even when occurring in more than one path. In this vein, template (390) has an idempotent representation as a set of paths, partially given below; bottoms are underlined.

(394) *representation graph of* laat ... toe *'admit' as a set of paths*



In our grammar, unification is an antisymmetric process: in every case of unification, a graph is projected on a well-defined weakly connected directed acyclic sub-graph labelled *arg(ument)* of another graph, and this process is not reversible. In (394), the paths starting with `TOP:ARG(B+I)` determine such a subgraph: the subgraph *governed* by that path. For the unification test itself, however, there is no hierarchy between the graphs involved. The definition of unification of feature value graphs, in terms of path unification, follows below. A path is a finite, directed sequence of labels, starting at the top node and ending with a value term; it is indicated as P→α, where α is the value term at a bottom vertex. The symbol '=' means 'equal to', while '⊔' stands for 'unifies with'.

(395) *Unification G1∪G2 of connected graphs G1 and G2*

(a) the unification G1∪G2 of G1 and G2 consists of all paths P→α⊔β for P→α in G1 and P→β in G2, where α⊔β exists, and of all paths R→σ in G1 or G2 for which no path R→τ exists in the other set.

(b) the unification G1∪G2 of G1 and G2 fails when there exist paths P→α in G1 and P→β in G2, but α⊔β does not exist.

(c) $\alpha \sqcup \beta$ exists iff either α is a variable or $\alpha = \beta$, or $\alpha = f(a)$ and $\beta = f(b)$, and $a \sqcup b$ exists
(d) if α is a variable, $\alpha \sqcup \beta = \beta$.
(e) if $\alpha = \beta$, $\alpha \sqcup \beta = \alpha$.

Even though our unification is steered in an antisymmetrical manner by the grammar, the resulting graph is subsumed by both of the *operanda*: the resulting feature structure $G1 \uplus G2$ is at least as informative as each of the original ones (Rounds 1997): it subsumes the originals. Since every combinatory process comes with unification of complex symbols, failure of unification amounts to local ungrammaticality, or denial of the hypothesis that two phrases combine.

Though unification is computationally expensive (but see Kešelj and Cercone (2002) for efficient techniques), the handling of (lexical) graphs themselves is liberal and relaxed. For example, one can add an edge to a lexical graph without any overall consequences. It is not necessary to add that edge to other graphs in the same class. In the same vein, edges can be removed without any overall consequences. Changing lexical graphs will have impact on only particular unifications. Since 'our' unification is conservative in that it neither destroys nor adds information on the fly and is governed by categories, the impact of extending or delimiting a lexical graph or a class of graphs can be predicted by inspection of the lexicon. As will be explained in section 3.5, this is easily done. Moreover, redundancy in the lexical specifications can be checked: a path $P \rightarrow X$ in a graph G of category C is redundant if no lexical graph has an argument subgraph of category C in which path P is specified.
Thus, control over the sets of lexical graphs is assured. Unification is effective, and therefore it is the backbone of every ambitious grammar automation. In this respect, it is important to observe that in DELILAH unification for parsing and generation is applied in a conservative and non-destructive way. Paths in a graph are not destroyed by unification for parsing or generation purposes. Unification leaves grammatical control to the lexicon, where the graphs are built and stored. Every single grammatical feature is guided from the lexicon; nothing happens outside its gates. The conservativity of unification is the anchor of grammatical lexicalism: for all information to be stored in the lexicon in a sensible way, we must make sure that no information is added during grammar application and no information is destroyed. We will see, though, that in *constructing* the lexicon unification – almost by necessity – must be applied destructively. In this sense, the mode of unification (conservative *vs.* destructive) is characteristic for the mode of the system: dynamic on-line

operations come with conservative unification; off-line the lexical database is built destructively.

3.3.2 Problems with re-entrance

Consider the two graphs in (396), assuming that the graph in (a) must unify its subgraph G with the graph in (b); the arrows indicate lexically imposed constraints.

(396) (a) G1: [.... [$_G$ slf: Y ↓, cat:xp ↓, ...[...slf:Y ↓ ...]] ...]
(b) G2: [.....slf: f(X), ...cat:xp, ...[...slf:X ...]]

Clearly, both X and f(X) will unify with Y, leading to an irresolvable instantiation loop: Y = X and Y = f(X), so X = f(X) = f(f(f..f(X)..)), *ad infinitum*. Though the problem is general for our form of *re-entrance* – variable linking within the same graph (see *e.g.* Bouma 1993) – we concentrate on the feature for the value for the Stored Logical Form. This feature typically has logically complex values (see section 3.2.4.2 and chapter 2).
The situation in (396) is not marginal: recurrence of a variable meaning at a higher level as in (b) is standard, and identification of lower and higher meanings as in (a) is typical for templates where the syntactical head does not contribute to meaning, *e.g.* lexically selected PPs or infinitival constructions with a purely functional complementizer. Moreover, we allow embedded constraints without restriction, as in the (a) template, to account for extended lexical units with specialized meanings. Still, we can argue that the additional requirement of co-categorization in (396) makes the clash unlikely; recall that every unifiable graph is categorized by definition, and therefore co-categorization is a precondition to unification. This being the case, the two templates in (396) show essentially different semantic dependencies within the same syntactical category. In normal combinatorial business, this is unlikely, since it runs counter to compositionality. But we must allow for zero-semantic phrases of any category, in which case nothing can be excluded. So, for the sake of argument, let us assume that (396)(a) requires a normal instantiation by a graph of category XP, and a non-compositional graph of that category is offered for unification. Now, the instantiation problem seems inevitable upon checking unifiability.

There are a few ways out. First, we may interpret the occurrence of the loop as failed unification. This is certainly in accordance with the remark on compositionality made above. Second, we can build in an *occurrence check* in these

cases, which is costly in terms of computational complexity. Third, we could impose the restriction on lexical templates that embedding of semantic conditions is not allowed – a kind of *head feature principle* as in HPSG (Sag 2003); it renders at least one of the two structures in (396) lexically illegal. This line contradicts our liberal attitude with respect to feature structures, to block as few collocational effects as possible. Fourth, we could not allow for 'eta-conversion' in unification – the unification of two variables – but instead delay unification until at least one variable is instantiated (*blockin*g; see Bouma and Van Noord 1994). But then we submit the syntax to unification, whereas we want the syntax to steer unification.
Reviewing the possibilities, we prefer the first option. The kind of clash in (396) is interpreted as failed unification, since the (semantic) networks in the two graphs differ essentially. However, we do not have to choose between options at all, because experiments do not demonstrate the problem. As a consequence, unification can be defined as elegantly as (395).

3.4 THE MAKING OF THE LEXICON

3.4.1 Organizing lexical knowledge: no lemmas – economy vs flexibility

The main characteristic of our lexicon is its flatness. It is a huge pancake, without external hierarchy. In particular, our lexicon is not organized by lemmas. Every word and every phrase occurs in all its individual glory without being subsumed under other levels of organization. For example, every finite form of a verb occurs in the lexicon with as many different graphs as it has combinatorial or semantic variants. This family of graphs is unordered, and so is the even larger class of graphs linked to other forms of that verb. The price to pay for this simple final structure is a complex set of rules for cooking the pancake. In this chapter, we describe the problems and the solutions of this *off-line* module.

Much of our knowledge of language is invested and exploited in the way the lexicon is generated. Although in the operational lexicon for parsing and generation lemma structure is not preserved, knowledge about lexical dependen-

cies, familiarity, generalizations and inheritance is compiled into the underlying generative system. The particular organization of this system, however, does not express principled knowledge of language. As a relatively simple example, consider the class of noun flections in Dutch. In general, a Dutch noun has a standard and a diminutive form, in both a singular and a plural edition. Also in general, we do not semantically distinguish between plural and singular forms – for the argument see Cremers (2002) – but we do semantically distinguish standard from diminutive. Plurals and singulars differ categorially: plurals in Dutch may be full noun phrases, singulars can be so only if they are non-countable. Not every diminutive derives its meaning from the standard form. Many plural forms are semantically indistinguishable from the singular. On the other hand, bare plurals – being NPs – must be differentiated from plural nouns – complements to quantifiers or determiners – and moreover, may have both a generic and an indefinite interpretation. Consequently, in our lexicon, the plural form *mannen* 'men' will come with at least three distinct templates, as listed below: a template as a noun with predicative semantics, a template as an NP with generic semantics and a template as an NP with indefinite semantics. In the latter two, a phonologically empty argument carries the noun meaning. This argument sub-graph is not specified for type, and therefore not open to syntax-driven unification. The crucial differential values are italicized.

(397) *lexical template of noun* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[],[],[]},D@man~[D]}
SYNSEM:AGGR:count
       CAT:n
       GENDER:nneut
       NUMBER:plur
       REFMODE:nontime
       SEX:male
TYPE:n\0~[]/0~[]
```

(398) *lexical template of generic plural np* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[{{[],[],[]},D@man~[D]}$E&(B+99)#F],[],[]},
    G@some^E^and(quant(E,gen), and(F,entails1(E,decr)),
    and(G,entails(E,incr)))}
SYNSEM:AGGR:count
       CAT:np
       FUNCR:incr
       GENDER:nneut
       NUMBER:plur
       QMODE:def
       REFMODE:nontime
       SEX:male
       SUBCAT:noun
TYPE:np\0~[]/0~[]
ARG:ID:B+99
    SLF:{{[],[],[]},D@man~[D]}
```

(399) *lexical template of indefinite plural np* mannen *'men'*

```
ID:A+B
HEAD:CONCEPT:man
     PHON:mannen
     SLF:man
PHON:C
PHONDATA:lijnop(mannen,A+B,[],C)
SLF:{{[{{[],[],[]},D@man~[D]}$E&(B+F)#G],[],[]},
    H@some^E^and(quant(E,some),and(G,entails1(E,incr)),
    and(H,entails(E,incr)))}
SYNSEM:AGGR:count
       CAT:np
       FREEARGS:no
       FUNCR:incr
       GENDER:nneut
       NUMBER:plur
       QMODE:indef
       REFMODE:nontime
       SEX:male
TYPE:np\0~[]/0~[]
ARG:ID:B+F
    SLF:{{[],[],[]},D@man~[D]}
```

Of course, these lemmas are produced by some mechanism, with some generality and some specificity. Below are a few scenarios for generating the

nominal graphs. In the *top-down* procedure, inheritance can be maximalized: independently of any particular noun, there is a general template for nouns, which is modified by a generation function in order to produce a dedicated graph for a particular instance of a particular noun. In the *bottom-up* procedure, no inheritance frame is pre-defined: we have a function that produces dedicated templates. This function, however, must embody the general properties of nouns in a dynamic manner. Between bottom and top – dubbed 'left corner' – there are several pre-defined templates, but they do not necessarily have any interesting common kernel that characterizes the category.

(400) *Top-down*
Define one single noun graph G_N. Determine for each noun N_m all its instances I_n. Determine per I_j the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function σ from graphs, nouns, instances and feature sets to graphs: $G_{N, i, j, l} = \sigma(G_{N,} N_i, I_j, F_l)$.

(401) *'Left-corner'*
Define a set of noun graphs $\{G_p | G_p$ is a graph of category *n*$\}$. Determine for each noun N_m all its instances I_n. Per I_j, determine the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function σ from graphs, nouns, instances and feature sets to graphs: $G_{p, i, j, l} = \sigma(G_{p,} N_i, I_j, F_l)$.

(402) *Bottom-up*
Determine for each noun N_m all its instances I_n. Per I_j, determine the set of particular features F_k. Apply for each N_i, for each I_j and for each F_l a function τ from nouns, instances and feature sets to graphs: $G_{i, j, l} = \tau(N_i, I_j, F_l)$.

The three strategies vary with respect to the integrity of lexical data structures. Under strategy (402) all data structures – lexical graphs – are created from scratch. Once created, they remain unaffected; even under unification, no information is destroyed. The functions that create the dedicated graphs contain functionally complete knowledge of the grammar of nouns.
Under the *top-down* strategy (400) the general template may have default values for certain features that can be overruled by the dedicated constructor functions. One can even conjecture that for a 'mother template' to be an interesting focus of nominal aspects, it must contain specifications that not every noun in each of its instances will maintain – default values are nothing more than that (cf. Bouma 1993). To state that the typical noun has a certain feature value is to state that some particular nouns may differ. Or, in different terms, the set of feature values that all nouns share is uninterestingly small. Therefore, the *top-down* strategy is either trivial, starting from an almost empty template, or destructive, as the template will contain information that is not

valid for all nouns. For the *top-down* strategy to be effective, it must allow the destroying of pre-specified paths in the top template.
The intermediate strategy seeks a balance between information destruction and sensible inheritance. It is really a matter of economy because both maintainability and linguistic transparency are at stake. If all information can be discarded in the course of the lexicon's construction, we may lose track of the level of grammatical knowledge invested in the pre-defined templates. If little or no information is shared between graphs of the same class, maintenance (adaptation, debugging, extension, ...) of the lexicon will get tough. Moreover, the construction functions for the dedicated graphs are among the most complex ones in the whole processing system (see below). Because they control vital parts of the attribute-value matrices and operate at the crossroads of generality and specialty, they too must be kept maintainable and adjustable. In the intermediate strategy of (401), the number of graphs underlying the generation of the lexicon is a matter of pragmatics, not of principle.
In the case of nouns, there is little grammar to help us to decide between either a basic template for singular count nouns and another for singular abstracts, or just one for both. In other cases, economy is more helpful. Every intransitive verb V in Dutch participates in a structure *zich een ongeluk V*, 'oneself an accident V' meaning 'to V very intensively'. Even without your calculator it is evident that creating one basic frame in which the intransitive verb is inserted and in which the crucial semantic dependencies are pre-established is much more effective than building every dedicated graph with this frame from scratch. Yet, both strategies will do the job.

The basic components of our lexicon can be summarized as follows:

(403) (a) a finite set of basic templates or graphs
(b) for each word a list of particular feature values with respect to one or more basic templates
(c) a set of rules, applying (b) to (a).

Actually, the particular specifications of a word can also be constructed as graphs. Ultimately, then, the sets (403)(a) and (b) may coincide – the lexicon is practically handmade, in that case. But descriptive economy may come in. A finite verb form, for example, lives on both properties of the individual form and on properties of the finiteness construction. It would be redundant to mix these two constructions over and over again in new unique graphs. Therefore, we had better take either a verb template to be modified with finiteness or a basic finite template to be enriched with verbal specifications – the nature of this choice will be discussed below. Set

(403)(c) contains the rules for performing these modifications. These apply a destructive form of unification called *adaptation*:

(404) *Adaptation*
The adaptation of a (basic) graph B to a specific graph C is the connected graph B[C] that unifies the largest subgraph B' of B that unifies with C, and C. That is: B[C] = B'⊎C.

Typically, B does not subsume the resulting graph B[C]: if B does not equal B', B contains information that is 'overwritten' in B[C]. Moreover, the specific graph C is typically dynamic, in that it is not stored and applied by general adaptation but it is imposed on the basic graph in a rule-like fashion: adaptation of one graph to another is a very precarious process, with all kinds of procedural implicatures too subtle to leave to a one-size-fits-all process. The use of rules to overrule or overwrite default specifications establishes the operational antisymmetry of the adaptation procedure.

Adaptation is the *off-line* constructor of the pancake lexicon – the flat compiled-out lexicon where every item is an object equal to all others. Moreover, adaptation offers a metric for lexical redundancy: the number of paths that are actually overwritten in constructing the lexicon – the overall sum of the differences between B and B' in definition (404) – can be compared to the overall number of specifications in the specific graphs C, *i.e.* the union of set (403)(b). The metric may operationalize the notion of *default value*.

3.4.2 General and special: everything as a graph

From the point of view of complex symbols or signs, it is not easy to decide whether a finite verb primarily represents finiteness or its verbal semantics. As a matter of fact, in the grammar of the Germanic languages, for example, the double nature of the finite verb is a major source of syntactical complexity: the structural position for finite inflection and the position of the verb are not necessarily adjacent. Morphologically, in these languages the verb provides the free morpheme, and finiteness is suffixed. But the morphological difference is combinatorially quite uninteresting: the bound morpheme is the functional element and it determines the main syntactical position of the word.
In a flat lexicon like the one we pursue, where every single word occurs in its own right, there is little compelling reason to derive the finite verb from its verbal root rather than deriving it from the finite basic construction, for example. There is a practical reason for choosing the de-verbal option, how-

ever: the verbal root is not predictable, whereas the bound morpheme comes from a closed class, and the irregularities in the verbal paradigms are easier to connect to the verbal roots than to the finite morphemes. Moreover, the finite form may inherit its basic valence and its thematic structure from the verbal root, so that not all syntactical features are determined by its finiteness.
Therefore, we derive finite forms by adapting a verbal basic template to a set of constraints inducing finiteness. The procedure is adaptation and not just unification, because at least the combinatorial category must be updated in the transition from infinite to finite signs: at least the category value is destructively adapted.
In a certain sense, this manoeuvre is upside down, as we apply general rules – imposing the templates of finiteness – to the more or less special attributes of a particular verbal template. The unprincipled trade-off that characterizes the in-between approach to lexical derivation shows up in (401): the special information contained in the verbal root's graph is overruled by more general, finite 'defaults'.
The same kind of decision we made with respect to de-verbal adjectival and nominal forms applies to the participles. First, a participle is encoded or 'signed' as a verbal derivative, and then its adjectival and nominal instances are produced by complex instances of adaptation: the process is inevitably destructive.

Below you will find the prerequisites of production for a standard verbal paradigm.

(405) (a) there is a *lemma* specifying
- a non-empty graph of characteristic attribute-values for that verb
- the address of one or more basic templates to be adapted by these characteristic values
- a (possibly empty) list of deviant morphonological instances of the verb

(b) there is *a set of retrievable basic templates*, elements of which the lemma can refer to
(c) there is *a set of rules* producing the morphonological paradigm of the verb
(d) there is a *set of classes of rules* each of which
- specifies a characteristic attribute-value matrix for every particular instance of the verbal paradigm
- adapts each of the templates referred to in the lemma, after adaptation to the lemma characteristics, to the attribute-value matrices for the particular instance

(e) there is a *set of classes of rules* each of which
- specifies a characteristic attribute-value matrix for a particular de-verbal instance
- adapts a particular matrix of a particular instance of the verb to this characteristic attribute-value matrices

This scheme of prerequisites does not fully reflect the actual organization of the lexical module, but it identifies the main agents in the production of a verbal paradigm in the present DELILAH architecture; similar schemes can be drawn up for nouns, adjectives and other categorial paradigms. The agents in the process could even be designed differently: the 'functional' matrices applied by the rules could just as well be independent initial templates like the ones mentioned in (b). This would lead to the following architecture.

(406) (a) for each lemma L introducing verbal paradigm V, specifying a paradigm-specific template TP
(b) for each template TV to which L refers
(c) for each morphonological instance M of V
(d) for each template TM specifying a particular function of M
(e) there is a function σ(TP, TV, TM, T) adapting TP to TV and TM to a final template T

Thus, we have a fully constructional approach, in which every grammatical specification is a retrievable template, and in which only one rule is called for, namely adaptation. There are templates not only for transitive verbs, but also for second person singular present finite forms with inversed subject, passive participles occurring as predicative adjuncts, nominalized generic plurals of present participles, and so on. This may be a desirable architecture of the lexical component: every grammatical object is a template, subject to adaptation. Unfortunately, however, the functions σ of (406)(e) may have to vary largely with these functional templates TV. When applying destructive modes of unification, it is unlikely – to say the least – that you will be able to use a one-size-fits-all adaptation strategy. Rather, every adaptation will require its own provisos, in particular with respect to the combinatory categories. Thus, under set-up (406) we still stick with a – possibly large – number of construction-specific adaptation rules: a set that may hardly be distinguishable from the present complexes which are referred to in (405)(d) and (e), to be discussed below.

3.4.3 The rules that make the language

Adaptation of graphs, templates or matrices is a powerful *off-line* process. It is almost too strong to apply *on-line*, as its destructive nature might be a threat to compositionality. For example, it does not impose a principled antisymmetrical relationship between the graphs involved, as is the case with unification in the way it is applied in DELILAH. As a constructional tool in building the lexicon, however, it is very useful. It provides compactness and generality,

and its finite product can be submitted to all kinds of well-formedness checks. Yet, adaptation itself is too coarse to be applied without additional steering. Just as syntax guides unification, adaptation is controlled by rules that specify constraints on the merge of graphs per category and subcategory. Since these rules compile knowledge of the structure of language, we will sketch their operation in this section by scrutinizing the making of the full paradigm of a simple transitive verb, *slaan* 'to beat'.

The root of the paradigm is manually specified as an instance of relation `lemma/5`.

(407)
```
lemma(
        slaan,          /* naming some root form of the paradigm */
        verb,           /* identifying the rule set to which the lemma is submitted */
        [transitive_verb, transitive_verb_with_small_clause],
                        /* listing the basic templates for the paradigm */
        [head:phon:slaan, head:concept:beat, head:sem:beat,
        head:synsem:etype:event, arg(10):synsem:theta:agent_of,
        arg(1):synsem:theta:theme_of],
                        /* specifying the particular paths for this paradigm with
                        respect to the basic templates listed in the third argument */
        [pressing12:sla, pressing23:slaat, pastsing:sloeg,
        pastplur:sloegen, participle:geslagen] ).
                        /* specifying (orthographically) 'irregular' (strong) forms of
                        the paradigm */
```

The lemma exclusively contains otherwise non-predictable properties of the *slaan* paradigm: it may occur in a sentence with a resultative small clause; it is an event rather than a state; its subject is an agent and its object a theme; moreover it entertains a few particular finite and infinite forms. Even at this elementary level, other decisions could have been made: the small-clause extension could be generalized to a default property of all transitive verbs; events may have agents as subjects by default; the forms might be predictable from tracing *slaan* to Indo-European verb classes. Clearly, every alternative choice would be reflected in the basic templates. Moreover, `lemma/5`'s third argument consists of a template in the form of a number of paths, to which the verbal templates named `transitive_verb` and `transitive_verb_with_small_clause` are adapted.

Because *slaan* is addressed as a verb, the construction of its paradigm is submitted to the class of verb-specific rules. This class is controlled by a single predicate `lemmalist/2`, which simply takes the lemma and produces a set of final templates, the full paradigm. The paradigm exists only in the

output of this predicate. As soon as the output set is adapted in the lexicon, the paradigm is lost.
`Lemmalist/2` performs the following operations:

(408) (a) it computes all different word forms in the paradigm *(sla, slaat, sloeg, sloegen, slaand, slaande, geslagen)*;
(b) for each template T in the lemma, it creates a basic instantiation BT by adapting it to the particular values of *slaan*;
(c) for every single finite and infinite form, it calls a class of rules adapting the basic instantiations BT into final templates;
(d) it calls a class of rules adapting some non-finite final templates into all de-verbal adjectival and nominal final templates.

The morphological component is not essential to our system, but economical. Although inflection is regular in general, the component deals with some mean properties of Dutch orthography. Its main function is reducing the number of forms that have to be listed in the (hand-made) lemmas.
The rules creating the finite forms are among the most complex of the whole system. In our combinatorial grammar, finite forms introduce the sentential category (see chapter 1). Therefore, all order variation at the top level of a sentence, including processes like left dislocation and inversion, has to be accounted for in the finite categories – it must be specified somewhere anyway. Moreover, Dutch is a verb-second language, which implies that main sentences and embedded sentences have different word orders, different combinatorial categories and, therefore, different finite 'heads'. Furthermore, main sentences come in three different semantic types. In addition, indefinite subjects may come with 'spooky' *er*. Although this number is partly an artefact of the grammar formalism we chose, a simple finite instance of a transitive verb like first person singular present *sla* is the phonological content of at least twenty-two final templates. All these final templates are adaptations of the basic templates, imposed and controlled by the rules for finiteness. They determine the sentential type, the combinatorial category, the order of arguments, their modes and their side effects. To follow the different stages, consider first the basic template for simple transitive verbs, with many variables to indicate the internal network.

(409) *basic template transitive verbs*

```
ID:Top+ID
HEAD:PHON:_X
     SLF:Main
     SYNSEM:ETYPE:Etype
            FLEX:infin
            VTYPE:transacc
SLF:{{[SemS&(ID+ID1)#A,
       SemO&(ID+ID2)#B,
       EStructure@some^E^and(quant(E, some), Main~[E],
       Etype~[E], entails1( E, incr),
       and(EStructure, entails(E,incr))) &(Top+ID)#EV], [],[]},
       Time@and(and(Stheta~[EV,A], Otheta~[EV,B]),
       attime(EV, Time)) }
SYNSEM:CAT:vp
       EVENTVAR:EV
       EXTTH:Stheta~[Top+ID, A]
       PREDTYPE:nonerg
       TENSE:untensed
ARG:  ID:ID+ID1
      PHON:_Subj
      SYNSEM:OBJ:subject_of(Top+ID)
             THETA:Stheta
      SLF:SemS
      ..

ARG:  ID:ID+ID2
      PHON:_Obj
      SYNSEM:CASE:obliq
             CAT:np
             DIR:left(1)
             FLAG:0
             OBJ:dirobject_of(Top+ID)
             THETA:Otheta
      SLF:SemO
      ..
```

All terms starting with a capital are variables. The values for the cases of subject and object are fixed by default. The target type of the basic template is set to *VP* and its tense to *untensed*, but that is fairly arbitrary.

One of the many final templates generated by the finite rules is the following, representing the first person singular present of a simple transitive verb in main (yes/no) questions. Its format is slightly different from the basic template's format – the former is generated, the latter is handmade. Paths and/or values changed (not instantiated) or added are italicized.

(410) *lexical template* sla *'beat' 1st person sg pres, main question*

```
ID:A+B
HEAD:CONCEPT:beat
     PHON:sla
     SLF:beat
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at-pres
            VTYPE:transacc
PHON:C
PHONDATA:lijnop(sla,A+B,[arg(right(-10),0,D),
         arg(right(-1),0,E)],C)
SLF:{{[F&(B+G)#H, I&(B+J)#K,
      L@some^M^and(quant(M,some),beat~[M],event~[M],
      entails1(M,incr),and(L,entails(M,incr)))
      &(A+B)#N],[],[]},
      quest$$and(and(and(agent_of~[N,H],
      theme_of~[N,K]),attime(N,O)),tense(N,pres))}
SYNSEM:CAT:q
       EVENTVAR:N
       EXTTH:agent_of~[A+B,H]
       PREDTYPE:nonerg
       SUBQMODE:P
       TENSE:tensed
TYPE:q\0~[]/0~[np^0#B+G,np^0#B+J]

ARG:  ID:B+G
      PHON:D
      SLF:F
      SYNSEM:CASE:nom
             CAT:np
             NUMBER:sing
             OBJ:subject_of(A+B)
             PERSON:1
             QMODE:P
             SUBCAT:pron
             THETA:agent_of

ARG:  ID:B+J
      PHON:E
      SLF:I
      SYNSEM:CASE:obliq
             CAT:np
             DIR:left(1)
             FLAG:0
             OBJ:dirobject_of(A+B)
             THETA:theme_of
```

One can see, for example, that the top-level semantics SLF is instantiated, whereas agreement features of the subject are added. Most combinatorial specifications are added or introduced by adaptation. The combinatorial TYPE and the PHONDATA (the linearization information) are absent in the basic template (409): they can only be added as the last step towards finalization. It is important to note that (409) does not subsume (410) but that (410) is an adaptation of (409) to (a) the lemma-specifications of (407) and (b) the following template for the first-person-singular-present-transitive-main question:

(411) *basic template 1st person sg pres main question*

```
ID:A+B
HEAD:SLF:MAIN
     SYNSEM:FLEX:fin
            NUMBER:sing
            PERSON:1
            TENSEOP:at_pres
PHON:C
PHONDATA:lijnop(Main,A+B,[arg(right(-10),0,D)],C)
SLF:{{[S&(B+F)#G,H@some^I^and(and(quant(I,some),
    event~[I, Main], entails1(I,incr)), H, entails(I,incr))
    &(A+B)#J], [], []}, quest$$and(and(Extth~[J,G],
    attime(J,K)), tense(J,at_pres))}
SYNSEM:CAT:q
       EVENTVAR:N
       EXTTH:Extth~[A+B, G]
       SUBQMODE:P
       TENSE:tensed
ARG:   ID:B+F
       PHON:D
       SLF:S
       SYNSEM:CASE:nom
              CAT:np
              OBJ:subject_of(A+B)
              PERSON:1
              QMODE:P
              SUBCAT:pron
              THETA:Extth
```

That is, the rule of finiteness creating the final template (410) may also be seen as one of the many instances of the adaptation relation σ/4 referred to in (406)(e).

Lemmalist/2 also produces all relevant de-verbal templates, according to (408)(d), apart from the many finite and infinite templates in the *slaan* paradigm. Among others, it feeds the templates establishing the passive participle

geslagen 'beaten' to a class of rules that create final templates for each of the following combinatorial varieties of the participle:

(412) de *geslagen* munt 'the stroken coin' *attributive adjective*
de *achthoekig geslagen* munt 'the octagonal stroken coin'
attributive adjective with resultative small clause
de *door de staat geslagen* munten 'the by the state stroken coins'
attributive adjective with agentive pp
de *door de staat achthoekig geslagen* munten
'the by the state octagonal stroken coins'
attributive adjective with agentive pp and resultative small clause
een *geslagene* 'a beaten (one)' *singular noun*
alle *geslagene* 'all beaten (ones)' *plural noun*
alle *geslagenen* 'all beaten (ones)' *plural noun for human beings*
de munt bleek eenmaal geslagen pijnlijk te ontwaarden
'the coin appeared once stroken to devaluate painfully'
...

In a lexicalist approach, all these forms must be made available and retrievable. In the DELILAH set-up, they result from a complex, partly destructive, adaptation of verbal and combinatorial or functional templates. This choice is, again, pragmatic. For constructions like nominalizations – the classical case for lexicalism in generative grammar – we have chosen to introduce them as such, rather than to derive them by adaptation from verbal templates; the choice is reasoned in Reckman (2009). Adaptation may become too complex or too intrinsic to be viable.

The two sets of rule classes in (405)(d) and (e) differ from each other in that the (d) classes take pre-final graphs and deliver final graphs, where the (e) classes take a final graph and adapt it into another final graph. The (d) classes, for example, produce the final finite templates out of raw material. The (e) classes produce final adjectival and nominal templates out of fully-specified participle templates, which remain part of the final pancake lexicon. The distinction, again, is pragmatic rather than theoretical. In fact, no aspect of the set-up (405) is mandatory: the single defining feature of our lexicon is its pancake outcome. All inheritance hierarchy and all differences between general and particular specifications are exclusively exploited in the off-line generation of the lexicon, not in its surface structure. At the end of the day, all graphs in the lexicon are equal and independent of each other.

3.4.4 The constructive lexicon

The lexicon – of any natural language – abounds in *extended lexical units* (elus): phrases with specialized meanings or combinatorics. And if everything is a graph, so is an extended lexical unit. Poß (2010) extensively discusses the variation in syntactical, morphological and of course semantic respect. Here we discuss the DELILAH implementation of the so-called *way*-construction in Dutch, as an illustration of the handling of complex elus. An instance mentioned before is repeated here, now with mandatory parts of the construction italicized:

(413) Geen enkele bankier had *zich een weg naar* de Raad van Bestuur kunnen *golfen*
No banker had himself a way to the Board of Directors been-able play-golf
'No banker could have succeeded in moving into the Board of Directors by playing golf'

We assume that Dutch has a verbal pattern of the following kind:

(414) subject + intransitive verb + reflexive pronoun + NP with 'way'-like nominal head and an attributive directional PP

Its meaning can be summarized as:

(415) *Subject* is *verb*-ing in order to arrive at *the complement of the preposition in the directional PP*

The main linguistic characteristics of the construction are these:

(416)
(a) every intransitive eventive verb is eligible;
(b) the construction is causative: the verb itself is not providing the main event of the construction; rather, the verb is instrumental;
(c) although the verb is intransitive, a reflexive anaphor occurs in an object-type position; its semantic contribution may be restricted to stressing the causative nature of the construction;
(d) the head of the *way*-NP is lexically fixed but semantically irrelevant; it may just contribute to the causative interpretation; its proper semantics do not play a role;
(e) the complement of the preposition is lexically open, just like the subject; it provides an essential parameter of the interpretation;
(f) the determiner of the *way*-NP coincides with the top-level event marking; it is a singular indefinite, and it may carry the possible negation: it is either *a* or *no*.

These constructions have to be accounted for in a semantic lexicon of Dutch. In general, elus require choosing one of the words involved as a hook. As our lexicon will contain only fully-specified and independent objects, in the DELILAH system this choice concerns only the adaptation of templates, not their final ordering or embedding in the lexicon. In the matter of the *way*-construction, the construal is fairly easy: because the syntactic head is the verb – its inflection determines the combinatorial potential of the whole – the best choice is to introduce the constructions as lemmas of every candidate verb. To this end, our lexicon contains the template (417), comparable to the general verbal template for transitivity (409); its main semantic specifications are underlined.

All intransitive verbs, *i.e.* their lemma-specifications, are adapted to this template, in addition to the other basic templates to which they are linked. The verbal rules discussed in the preceding section also apply to the outcome of this adaptation, yielding the full spectrum of finite and infinite forms. As a consequence, the pancake lexicon will also contain, among many other instances of the verbal paradigms for *lachen* 'to laugh', the form (418) – with the same network in bold as in (417), and where DIR and FLAG values of the arguments were used to compose the syntactical TYPE.

Template (418) represents the *way*-construction with *lachen* as it occurs in embedded sentences, in finite past singular form, as in:

(419) Wij wisten niet dat de bisschop *zich een weg naar het Vaticaan lachte*
'We did not know that the bishop laughed himself a way into the Vatican'
We did not know that the bishop made it into the Vatican by laughing

The template occurs in the lexicon on a par with all other templates derived from *lachen*. Its presence there, as an independent and fully-equipped combinatorial object, is an immediate consequence of the lexicon's pancake 'architecture'. Moreover, many more constructions with verbal heads have to be compiled out – a clear overview of collocations per verb or per head is not available. In its final edition, then, the lexicon will contain many millions of such objects. Such a lexicon is viable only if it can be disclosed efficiently for parsing and generation tasks. The access to this type of lexicon is the theme of the next section.

(417) *basic template for Dutch* way*-construction*

```
ID:Top+ID
HEAD:PHON:_X
     SYNSEM:CONCEPT:Main
            ETYPE:Etype
            FLEX:infin
            SLF:Main
SLF: { { [ SemS&(ID+ID1)#A, SemPPNP&(ID3+ID4)#C,
           EStructure@some^E^and(quant(E, Quant), Main~[E],
           Etype~[E], entails1( E, FuncL),
           and( EStructure, entails(E, FuncR)))&
           (Top+ID)#EV],[],[]},
       Time@and( and( Stheta~[EV,A], some^EE^and(quant(EE,some),
       and(and(event(EE), move(EE)), theme_of(EE, A)),
       goal_of~[EE, C], entails1(EE, incr), cause(E, EE)),
       entails(EE, incr)), attime(EV,Time)) }
SYNSEM:CAT:vp
       EVENTVAR:EV
       EXTTH:Stheta~[Top+ID, A]
       PREDTYPE:nonerg
       TENSE:untensed
ARG:   ID:ID+ID1+10
       PHON:_Subj
       SLF:SemS
       SYNSEM:NUMBER:Number
              OBJ:subject_of(Top+ID)
              PERSON:Person
              THETA:Stheta
ARG:   ID:ID+ID2+1
       PHON:_Obj
       SLF:{_Stores0, _@Quant^_^_}
       SYNSEM:CASE:obliq
              CAT:np
              DIR:left(2)
              FLAG:0
              FUNCL:FuncL
              FUNCR:FuncR
              NUMBER:sing
              PERSON:3
              QMODE:indef
              THETA:theme_of
       ARG:   ID:ID2+_ID4+1
              HEAD:CONCEPT:road
              SYNSEM:CAT:n
                     NUMBER:sing
ARG:   ID:ID+_ID5+3
       PHON:_Reflex
       SLF:_RefSem
       SYNSEM:CASE:obliq
              CAT:np
              DIR:left(3)
              FLAG:0
              FOCUS:nonfocus
              NUMBER:Number
              PERSON:Person
              PRON:refl
              SUBCAT:pron
ARG:   ID:ID+ID3+2
       HEAD:CONCEPT:towards
       PHON:_PP
       SLF:_SemPP
       SYNSEM:CASE:obliq
              CAT:pp
              DIR:left(1)
              FLAG:0
              THETA:goal_of
       ARG:   ID:ID3+ID4+1
              SLF:SemPPNP
              SYNSEM:CAT:np
```

(418) *lexical template for* lachte *'laughed' as head of the* way*-construction*

```
ID:A+B
HEAD:CONCEPT:laugh
     PHON:lachte
     SLF:laugh
     SYNSEM:ETYPE:event
            FLEX:fin
            NUMBER:sing
            PERSON:C
            TENSEOP:at-past
            VTYPE:transacc
PHON:D
PHONDATA:lijnop(lachte,A+B,[arg(left(1),0,E),arg(left(3),0,F),
         arg(left(10),0,G),arg(left(11),wh,H)],D)
SLF:{{[I&(B+J)#K, L&(M+N)#O, P@some^Q^and(quant(Q,R),laugh~[Q],
       event~[Q], entails1(Q,S), and(P,entails(Q,T)))&(A+B)#U],[],[]},
       and(and(and(experiencer_of~[U,K],some^V^and(quant(V,some),
       and(and(event(V),move(V)),theme_of(V,K)),goal_of~[V,O],
       entails1(V,incr),cause(Q,V)),entails(V,incr)),attime(U,W)),
       tense(U,past))}
SYNSEM:CAT:s_vn
       EVENTVAR:U
       EXTTH:experiencer_of~[A+B,K]
       PREDTYPE:nonerg
       SUBQMODE:X
       TENSE:tensed
TYPE:s_vn\0~[pp^0#B+M, np^0#B+Z, np^0#B+Y, np^0#B+J]/0~[]

ARG:  ID:B+J
      PHON:G
      SLF:I
      SYNSEM:CASE:nom
             CAT:np
             NUMBER:sing
             OBJ:subject_of(A+B)
             PERSON:C
             QMODE:X
             THETA:experiencer_of

ARG:  ID:B+Y
      PHON:F
      SLF:B1
      SYNSEM:CASE:obliq
             CAT:np
             FOCUS:nonfocus
             NUMBER:sing
             PERSON:C
             PRON:refl
             SUBCAT:pron

ARG:  ID:B+Z
      PHON:H
      SLF:{C1,D1@R^E1^F1}
      SYNSEM:CASE:obliq
             CAT:np
             FOCUS:nonfocus
             FUNCL:S
             FUNCR:T
             NUMBER:sing
             PERSON:3
             QMODE:indef
             THETA:theme_of
      ARG:  ID:Z+G1
            HEAD:CONCEPT:road
            SYNSEM:CAT:n
                   NUMBER:sing

ARG:  ID:B+M
      HEAD:CONCEPT:towards
      PHON:E
      SLF:A1
      SYNSEM:CASE:obliq
             CAT:pp
             THETA:goal_of
      ARG:  ID:M+N
            SLF:L
            SYNSEM:CAT:np
```

3.4.5 The lexicon is local

In the development of grammatical theory, the idea that grammatical relations must be restrictive is fundamental. Not every observed dependency as such qualifies a grammatical construct. Generative grammar almost lives by restrictivity, although it started out by widening the concept of rule of grammar by introducing transformations. Yet, the quest for conditions on transformations is still a fine way to describe the grammar programme of the generative 'enterprise'. The most impressive results have been in the field of locality: no rule of grammar relates items over a random structure, or: if two elements are interdependent, it is always possible to finitely characterize the path between them. In early generative grammar, this resulted in the condition on transformations that essential variables not be involved.

Sag (2003: ch. 16.6) repeats the message for HPSG in a very strict way: no construction has access to its daughter's daughters, and that is not just handy but fundamental. It is, for example, 'a striking fact about human languages' that they do not exhibit verbs that select for case in their verbal complement. So concepts of locality like this keep grammatical descriptions within the borders of theoretical relevance, restricting the notion *possible (grammar of a) language*.

Still, under a fundamental lexicalist approach to grammar, a less restrictive but more intrinsic notion of locality than Sag's access principle presents itself. If all grammatical relations were made explicit in the lexicon and the lexicon consisted of fully-specified combinatoric objects, all grammatical relations would be localized *because* they were lexical. Of course, we are free to make lexical graphs as complex as necessary to accommodate all dependencies, but complexity does not interfere with specificity. The only variables occurring in a lexical graph are terms at the end of a path. Every path is rooted, the graph is connected and directed, paths cannot be made recursive, and every graph is finitely traversable. In our system, the unit of locality is the lexical graph, in all its variety. That is, every interesting grammatical relationship can be specified within the borders of a single lexical graph. That, too, might be seen as a striking fact about human languages. This is what makes languages computable.

3.5 DISCLOSING THE LEXICON: OBJECT-ORIENTATION AND SPEED FOR SEMANTIC GENERATION

3.5.1 The enterprise

As was said before, DELILAH adheres to the 'gnostic' approach to computational linguistics: the view that explicit knowledge of language can be assembled, formalized and exploited. As the system focuses on semantics and the full specification of logical form, it must operate on a high level of grammatical and lexical fine-grainedness. As a consequence, DELILAH's lexicon is both detailed and large. It is generated on the basis of a restricted number of 'predefined', underspecified, generic templates, *e.g.* for transitive verbs or abstract nouns, representing minimal default graphs for all kinds of lexical types (see section 3.1.4). These templates can also be considered as constructions in the sense of Croft (2001). The templates formulate important generalizations about lexical relations, meanings and syntactical behaviour of phrases. This class of templates is flexible, and defined by practical and empirical considerations. A particular template is produced by specifying differences with respect to a more general template. In this sense, every template itself defines a lemma. A set of rules produces the lemma from the generic template(s), by inheritance, and the specified difference list. From this lemma, another set of rules produces graphs for all the marked instances – morphological or otherwise – of the lemma. The lexicon, a set of 'lexical entries', thus hinges on three components: a set of generic templates, a set of lemma difference specifications and a complex set of rules (see (403)). Data management, then, is done on a higher level than in a traditional database.

The lemmas or lexical entries are completely defined by HPSG-style feature-value specifications (Sag et al. 2003 and section 3.1.2). Lemmas are complex symbols, and can be represented by Attribute Value Matrices (avms), or Direct Acyclic Graphs (dags). Typically, they have a different number of features. A lemma may or may not specify a certain value for a certain feature. Besides atoms and numbers, values can be complex structures themselves, defining sub-graphs. A lemma will contain sub-graphs for semantically and/or syntactically related phrases. Unification, as defined in (395), will apply to these sub-graphs, that is, two graphs A and B unify whenever B unifies with a designated sub-graph of A, in which case A is called primary and B secondary. By definition, the primary graph constrains the secondary graph in every relevant aspect: morphologically, syntactically and semantically. This way,

the DELILAH lemma is a natural way of expressing collocational effects, from weak combinatory effects to rigid combinations. In fact, every lemma defines the domain for collocational effects. The lemma essentially separates constraints on sub-phrases of a structure from properties of the overall phrase. Inheritance, information sharing and co-indexing are specified by using the same variable as a value at different places in the graph. These places can be treated as pointers to the same memory location, rather than as identically valued contents. By definition, a graph is underspecified in the sense that not necessarily every possible or even actual value is lexically instantiated.

Section 3.4.2 describes the lexicon as a collection of explicitly defined, spelled-out, unrelated, and autonomous linguistic entities, containing all the necessary linguistic knowledge that is needed to properly use the word (form) in a language. By 'unrelated' we mean that the entities are stored independently of each other: there is no external hierarchy, and there are no imposed paradigms. By 'autonomous' we mean that entities, once retrieved, are operated independently of each other. A different approach is followed in Cornetto, which defines a combinatorial and relational, i.e. implicit, network on word level for Dutch (Vossen *et al.* 2007). In DELILAH such an information network, including, for example, collocations, has been 'compiled away', yielding real linguistic entities to start with, *e.g.* for generation purposes.

The following examples will give an idea of the lexicon's size and growth, and demonstrate the storage and access problem of a large computational lexicon. Adding the lemma for *hij* 'he' means adding 1 lexical entry, while adding the lemma for *gelopen* 'walked' (past. part.) means adding at least 19 entries; adding the lemma for *heeft* 'has' (3rd.pers. pres. sing. aux) means adding at least 133 entries; and adding the whole class of forms for the simple intransitive verb *verven* 'to paint' means adding 226 entries. Clearly, the fully written-out specification of lexical entries introduces an exponential storage factor. These figures are to be interpreted relatively and do not mean anything by themselves. They reflect the current draft state of the lexicon. Furthermore, for DELILAH's grammar-driven generation component, efficient access to the lexicon is crucial, because a word form should be produced only when its lexical specification matches certain constraints specified by the grammar and by the generation algorithm. It has been observed that searching and finding lexical entries is the main business of the generator. Therefore, efficient access methods are required for retrieval, with the ability to search and match complex lexical graphs and lexical constraints, which is hard. Finally, we developed our system in Prolog (Clocksin and Mellish 1984) for historical reasons. We implemented as many standards as possible, including ISO Prolog (Deransart et al. 1996; Clocksin and Mellish 2003), and refrained from using closed-

source libraries or third-party packages. This approach has yielded (almost) portable software. As a consequence, we are faced with the problem of implementing a fast and large-scale lexicon from a Prolog environment.

3.5.2 Two models

Lexicons may be modelled in different ways; here, we are focussing on computational models. The question whether linguistic information is stored centrally in the brain or rather in a distributed way is not addressed here. Neither is the question whether lexical entries are available in pre-compiled format, or get 'unfolded' on demand. A computational lexicon can be stored either in the internal, working memory of the computer, or in external, secondary memory, i.e. on a hard disk. Storing the lexicon in internal memory implies loading all lexical entries. Access to data structures in internal memory is usually very fast. Working memory, however, cannot be extended beyond a few gigabytes, which is not large enough for our purposes, while extension by virtual memory gives bad performance. The enormous proportions of the lexicon, let alone its foreseen expansion, and the limitations of current hardware rule out the option of storing it in internal memory. Storage of the lexicon externally does not pose a problem spacewise, but is slower and harder to access. As the lexicon is of a static nature, once it has been generated a full-featured database management system, including update facilities, is unnecessary. We can restrict ourselves to implementing a lexicon as a saved set of 'read-only' lexical entries, and provide efficient access methods for at least the most important features used by the generator, being the syntactical type, the semantic concept, and the word form.

The lexicon, built as a collection of lexical entries, can be regarded as a set of records in a database. There are a number of database models for database management. The Relational Model (Codd 1970) is based on two parts of mathematics: first-order predicate logic and the theory of relations. It is data-based and well-known from relational database management systems (RDBMS). As the programming language Prolog is based on the Relational Model, it seems straightforward to consider this model in more detail. Alternatively, the Object-Oriented Model (Meyer 1997) is investigated, because it was noted that a lexical entry in our system resembles the notion of 'object' in the OO Model. It can be called a 'linguistic object' in this model. A linguistic object stores data (lexical specification), and methods (procedures, *e.g.* for linearization). The OO Model is knowledge-based. Furthermore, the OO Model is often recommended when there is a need for high performance pro-

cessing of complex data, *e.g.* binary multimedia objects. The OO Model is well-known from graphical user interfaces ('point-and-click devices'), while object database management systems (ODBMS) are emerging.
Prolog – our programming vehicle – is itself relationally oriented rather than facilitating object orientation, but this feature did not influence the choice made. We assume that we can use the Prolog language as a powerful query to a relational Prolog database and apply Prolog's declarative semantics to the OO paradigm as well.

3.5.2.1 The Relational Model

The Relational Model was the first formal database model, solidly founded on well-understood mathematical principles and explained by Date (2003). It was invented in those days when computer memory was scarce and expensive. A relational database consists of a number of relations (often called tables), in which all data is stored. Each relation is a set of tuples that all contain the same attributes (the horizontal rows, often called records). A tuple is an unordered set of attribute values (the vertical columns, often called fields). An *n*-tuple is an unordered set of *n* attributes. An attribute is an ordered pair of an attribute name and a type name. Attributes are accessed by name, instead of by their position in the tuple. All of the attribute values (values in the same column) should be in the same domain, that is, they should be a valid value for the data type of the attribute, and they should obey the same constraints. Data types must be scalar, like an integer or a string, and cannot be compound, like a graph. Constraints provide a way of restricting the data that can be stored, either in tuples or in attributes. A relation is said to be *n*-ary iff it consists of a set of *n*-tuples. A special kind of constraint is a key. A key is an *m*-tuple of an *n*-ary relation, where $m < n$, which enforces the uniqueness of the combination of the *m* attribute values for each tuple. Key values are usually kept in an index table or hash table, which is stored in internal memory for fast access. By using keys, storage of duplicate data is prevented. The chance of duplicate data is further reduced by applying a set of normalization rules to the database structure. A 'relvar' (relational variable) is a named variable ranging over the set of tuples; as the result of a query, a subset of the set of tuples can be assigned to it, including the empty set.

The lexical entries of our lexicon are complex, non-atomic data structures. The Relational Model only allows atomic data types. It would be possible to pre-compile them into flat strings, which are atomic. Generally, however, flattening structured information is not a good idea when it comes to retrieval on the basis of some highly detailed substructure.

Lexical entries are directed acyclic graphs (dags), recursive data structures. Although it would be possible to pre-compile all feature-value paths of a dag to a number of tuples in different tables by means of a recursive procedure, it would be impossible to retrieve them since a relational database system does not provide for recursive processing (Hirao 1990). On the other hand, when we regard the lexicon as knowledge, and mark DELILAH as a knowledge-based system, we can linearize and store the graphs in a relational database, and use the powerful processing of recursion for inferences by Prolog. It is possible to successfully and easily store objects in a relational database (called 'Object-Relational mapping') by following a step-by-step procedure (Ambler 2000). A disadvantage is that quite a number of tables might be involved as graphs typically hold large numbers of features, while for semantic generation complete lexical specifications are required. Pre-compiling a lexical entry for a noun will typically yield a different number of tuples (in just as many tables) from pre-compiling a verbal entry. The top level, a main table, which is to represent complete lexical entries, has to span all the tuples transferring them into one large main tuple, in which each attribute represents a path, and where the attribute's value is either the value of the path or an ID that links to another table. This implies that there will be more than one top level, one table for each combination of attributes, and consequently more than one main table. As we do not want to impose a restriction on the internal dependencies of a graph, there is no restriction on the recursion depth of features. Consequently, the number of different main tables can be large in practice and infinite in theory. For practical purposes, e.g. retrieval by DELILAH's generator component, more than one main table means decreasing performance with orders of magnitude. Because of this, lexical entries cannot be regarded as single, homogeneous relations in the Relational Model; they cannot be retrieved as such.

Furthermore, lexical entries use variables, e.g. to co-index as yet unknown information between nodes in the graph. The Relational Model does not allow attribute values to be variables. A variable is not an atomic constant, and therefore not distinguishable from other values of the same attribute. As a consequence, an attribute that has a variable in its data domain cannot be indexed, which could lead to bad overall performance of the database. It would be possible to pre-compile variables into constants, and to de-compile them during retrieval. Calling meta-predicates, however, is generally rather time-consuming.

We conclude that the Relational Model, although appreciated for its economic storage, cannot efficiently accommodate (logic) variables, recursive features, and, consequently, the top level. Despite its efficient access, the Relational

Model does not meet high-performance demands on complex (recursive) data structures. The Relational Model is inappropriate for our purposes.

3.5.2.2 The Object-Oriented Model

In the Object-Oriented Model, information and control are represented in the form of interacting 'objects', as is well-known from the Object-Oriented Programming (OOP) paradigm. An object can be seen as a little information processor. It accepts commands (called 'messages') from other objects, processes data by executing procedures (called 'methods') that are stored in the object, and sends commands to other objects (called 'message passing') to be executed by those objects. It operates like a neuron in the brain, or a router in a computer network. By keeping data (properties) and procedures (operations) together in one local unit, an object holds all characteristics related to some concept. This implies that an object is a complex data structure. An object is independent of other objects; it has its own ID and its own role. These characteristics make an object attractive for representing 'behaviour'. OOP, then, is modelling a problem by distinguishing different (abstract) levels of objects (called 'classes' and 'subclasses', which are kept in a 'class hierarchy'), and defining their cooperation and interactions. A class defines the general characteristics of a concept in terms of the problem domain. An object is a particular instance of a class, from which it inherits all properties and methods, and to which it may add its own information, or overwrite inherited information. Inheritance may be seen as an 'is-a' relationship. Multiple inheritance means inheriting from more than one independent class, thereby combining properties and methods. Classes are the structuring elements (modules) in OOP, which hide the details of the code to be accessed by objects that stem from other classes.

Our linguistic objects are generated by deriving information from (one or more) generic classes of templates – 'constructions' – and by adding local information. After their creation, linguistic objects are independent of generic classes or other objects. This fits nicely in the OOP concept of objects that are constructed by a specialized 'constructor' method and by inheriting information from multiple classes. Our linguistic object, a complex, recursive graph, can easily be mapped onto an OOP object, which is a complex data structure. Shared variables, in fact, stand for a unification procedure, deferred until runtime. Encoding them by an OOP method is straightforward. Linguistic objects are independent information units, and, thus, uniquely identifiable, as are OOP objects. This makes an object, including all properties and methods, accessible by one ID, which is very important for efficient storage and access

by data-intensive processes such as semantic generation. ISO Prolog terms, being complex data types, are well equipped to represent OOP objects, including shared and singleton variables, recursion, and unique identification.
On the other hand, as linguistic objects are built from classes, and any combinatory difference is compiled out as a difference between objects, objects may differ from each other minimally, yielding a significant amount of overlap between objects, and introducing an exponential space factor. For example, the linguistic objects for the 2nd and 3rd person of a regular verb differ *only* in the person and phonological features.
Linguistic objects can adopt different states when they get involved in some linguistic process, like generation. When we talk about storing linguistic objects, we mean saving only their initial state. Objects whose states have been saved are called 'persistent'.
We conclude that the Object-Oriented Model provides a natural environment for representing linguistic objects. It does not suffer from the drawbacks of the Relational Model with respect to data modelling, and potentially facilitates fast access by unique identifiers. Its data redundancy is a small price to pay, given the considerable decrease in the price/performance ratio of hard disks each year. Future developments in runtime file (de-)compression techniques, as demonstrated on the level of the operating system, or the application, e.g. Java, might weaken this disadvantage. Prolog's term data type is suitable to represent linguistic data objects.

3.5.2.3 Object-Oriented Databases

An Object-Oriented database is a database which stores objects as created and modelled in OOP. Or, more strictly, an OO database system must satisfy two criteria: it should be a DBMS, and it should be an object-oriented system, i.e., to the extent possible, it should be consistent with the current crop of object-oriented programming languages (Atkinson *et al.* 1989).
OO databases arose after it was discovered that relational databases lacked high-performance processing on complex data structures like graphs. In the Relational Model, complex data, split and stored in several tables, have to be retrieved by searching these tables and combining pieces of data (called 'joining' in relational jargon), while in the Object-Oriented Model, a complete object is retrieved by its ID in one operation. In the Relational Model, the relational database is accessed by a declarative query language, typically SQL, which needs a procedural interpretation at runtime. The declarative query language permits general-purpose queries and transforms them into efficient retrieval procedures. In the Object-Oriented Model, lacking a standard query language, the OO database is accessed by means of a pre-compiled pointer

mechanism, which can be regarded as an optimized query answerer. In our application of a computational lexicon for semantic generation, we do need specialized queries, e.g. on syntactical type, semantic concept, and word form. Hence, an OO database is appropriate.

We come to the conclusion that, as our lexicon is static after it has been derived from generic templates, we do not need a full-blown object database management system (ODBMS) for persistent storage, but we can limit ourselves to implementing a simpler OO *lexicon*, containing fully-specified lexical entries, for retrieval only. We can represent objects by Prolog's native term data type, and manage them by Prolog's execution mechanism and pre-compiled pointers.

3.5.2.4 An example

We illustrate the operation of the generator by the following simplified example. In categorial grammar, a category consists of a head, and zero or more arguments to its left and/or right side. For generation purposes, the category can be regarded as an agenda. The generation algorithm keeps heads already produced in an unordered list, and arguments still to be produced in a stack. It handles them by inserting and deleting elements from an arbitrary position or from the top of the stack. It starts with a random semantic concept, and finds one of its realizations in the lexicon. The head of its category is inserted in the list. The arguments are shifted onto the stack, in reverse order. Each argument is produced either by reduction of some head in the list, or by reduction of a new category to be found in the lexicon. The topmost argument of the stack is replaced by the arguments of the new category, or removed completely, when it does not have arguments of its own. When the argument stack is empty, there are two possibilities. When the heads list does not consist of exactly one of the sentential categories s ('sentence') or q ('query'), the lexicon is consulted for a category that has the non-sentential category as an argument. Its head is inserted in the heads list, while its arguments are shifted onto the argument stack. If not, the algorithm stops. Categories, being complex symbols, are unified with each reduction step. The following example demonstrates the procedure, in line with (224).

1. search for random concept; find *betekenissen* 'meanings', cat=n; heads={n}, args={}
2. search for random entry with argument n; find *diepe* 'deep', cat=np/n
3. reduce arg n with head n in step 1; insert head np; heads={np}, args={}
4. search for random entry with arg np; find *ontdekt* 'discovers', cat=s\np1/np2
5. reduce arg np2 with head np in step 3; insert head s; shift arg np1; heads={s}, args={np1}

6. search for random entry with head np; find *die* 'that', cat=np/n
7. reduce arg np1 with head np in step 6; shift arg n; heads={s}, args={n}
8. search for random entry with head n; find *Nederlander* 'Dutchman', cat=n
9. reduce arg n with head n in step 8; heads={s}, args={}

Linearization information, kept in the complex symbols, is applied, yielding the final sentence *die Nederlander ontdekt diepe betekenissen* 'that Dutchman discovers deep meanings'.

3.5.3 Methods

We will describe methods that store our linguistic objects as objects in an OO lexicon, and methods, some borrowed from the Relational Model, that retrieve these objects efficiently and fast. The methods have been implemented in ISO Prolog. They should obey the Resource Principle, which is stated as: "Deploy working memory when performance is the key factor, and deploy external memory when storage is the main aspect". This principle is a practical phrasing of the insights that working memory will never be large enough to hold our current and planned number of linguistic objects, and that working memory is needed for real computation tasks, like semantic generation, and of the facts that working memory is faster than external memory, and that external disk space is abundant and cheap. We refrain from implementing interfaces to third-party products, e.g. the (semi-commercial) Berkeley DB library for external storage of terms (SICS 2014), or the Objects Package in SICStus Prolog (SICS 2014), because we prefer 'light', compatible, and manageable interfaces that are portable to other platforms.

3.5.3.1 Direct access and index tables

The 'Edinburgh' Prolog standard (Clocksin and Mellish 1984) offers sequential read and write access to files. On average, searching a record, given its number, out of N records will take time that is proportional to N/2. Although this is still efficient in complexity terms, for big N, however, searching will take an unacceptable amount of time. In ISO Prolog, the concept of a file has been improved on, and been replaced by the concept of a 'stream'. A stream is a file with random access, which means that each byte in the file can be located in constant time.
An index table is a table relating unique, simplex values to the positions, or context, where these values can be found. In our case, this translates to a table per feature that relates each feature's value to all occurrences of objects con-

taining that particular feature with that particular value. Internally, Prolog uses 'first-argument indexing' to locate the correct clause, given its first argument, which must be constant. This technique is not part of ISO Prolog, but it is regarded as an essential facility for interpreting Prolog programs efficiently, to be compared with the 'array' data type in other programming languages. It is implemented in every professional Prolog system on the market. An index table can be simply implemented by a collection of clauses, where each clause's first argument encodes the feature's values.
A fast lexicon for semantic generation hinges on the stream concept and indexing techniques.

3.5.3.2 Caching

Internal (working) memory can be accessed much faster than external memory. Applications often need the same data again and again. This has led to the development of cache memories. A disk cache is a storage mechanism in working memory that keeps the most recently read data plus the data of adjoining sectors in a buffer. As soon as the application asks for some data, the buffer is consulted first, saving access time to the hard disk. When the required data does not fit in the buffer, an extra disk access is necessary. To exploit disk caching, the data must be ordered in a way that corresponds to a relevant search criterion. Disk caching might be implemented at the application level or at the operating system level, or both. Caching at the application level, as advocated by Ceri *et al.* (1989), runs counter to the Resource Principle.

3.5.3.3 Hashing and compression

Hashing (Knuth 1973) is a method that converts an arbitrary, complex value to a simplex one (the 'hash' or 'key') by applying a hash function to it. Typically, hashing converts to an integer, because it is the most economical data type, and the easiest for a computer program to use. A hash value enables the use of an index table. Searching for an object in a collection of N objects, given an unhashed value of some constraint, will take $O(N)$ comparison operations. When the value is hashed, and the objects are indexed on the constraint by applying one and the same hash function onto their values, searching an arbitrary object only takes $O(1)$ time, assuming that the index table can be directly accessed. This is easy to implement in Prolog systems that can do first-argument indexing.
As each piece of data ever to be searched for is fixed, the lexicon is a collection of 'static search sets'. For such sets, a 'perfect hash function' (PHF) can be designed that is a function that will never assign the same hash value to

different data structures. However, depending on the complexity of the data structure, the hash can exceed the range of the integer data type on some computer platforms. A 'near perfect hash function' takes the integer range into account, but allows for duplicates ('collisions'). Duplicates increase space and time complexity. A 64-bit platform extends the integer range by several orders of magnitude compared to a 32-bit platform, enabling a PHF to be used, yielding zero collisions.

Number grouping is a compression technique which replaces a set of adjacent integers by a range starting with the lowest number and ending with the highest number. The space complexity for a range is constant instead of linear for a set of numbers. The time complexity for determining the subset of two ranges is constant, while it is linear for intersecting ordered sets.

3.5.4 Creating and accessing the object lexicon

The object lexicon, including retrieval methods, is generated by an off-line process which is not subjected to the Resource Principle. During generation, the objects are checked for well-formedness and validity. We will not describe the creation of the linguistic objects here; see section 3.4. Once they have been created, they are linearized into a series of bytes, and stored as Prolog terms in an output stream. We might store them in XML format as well, to make the lexicon application-independent.

A persistent object can be seen as a variable-length 'record', a concept from the Relational Model. The implicit 'record number' acts as the object's ID. The physical address of the object's first byte is kept in an external access table. The object lexicon and the access table have the same – large – number of entries. Storing them externally is in agreement with the Resource Principle. The access table holds records of a fixed size. As a consequence, it can be addressed by a function in working memory mapping an object with ID=N to the physical address of the Nth entry in the access table. The information to be found there in its turn refers to the physical location of the Nth object in the object stream. Thus, objects are retrieved by ID via an indirect addressing scheme in constant time, at the cost of accessing the disk one more time, of one function in working memory, and of one auxiliary access table in external memory. Prolog's built-in read predicate loads the objects as native Prolog terms.

Efficient data structures are the basis for efficient algorithms. Therefore, the objects need to be ordered by some criterion. A parser would benefit from an ordering on the phonological field for lexical look-up and tagging purposes, and would exploit a disk cache by accessing *all* objects with the same sur-

face form using a buffered read operation. A generator would benefit from an ordering on the most frequently used deep constraint for handling the agenda. As our semantic generator picks only *one* carefully selected object at a time, physical ordering does not seem to be beneficial. However, physical ordering by a criterion corresponds to a very compact index table for that criterion, because each entry can be represented by a range. Working memory is saved this way and intersection operations are sped up. It turns out that a physical ordering on syntactic type will prove helpful to the semantic generator. This exemplifies that a lexicon to be deployed both for a parser and for a generator needs to meet differing requirements. As the class of syntactical types is much smaller than the class of surface forms, a physical ordering on the former will yield fewer but bigger ranges, and as a consequence, a more compact index table than an ordering on the latter, saving more memory, in line with the Resource Principle.

3.5.4.1 Creating and accessing index tables for concept, type and phonology

For semantic generation, we require maximum performance on the retrieval of linguistic objects, specified by constraints on the features for semantic concept, syntactical type, and word form. For each linguistic object, its concept, type, and phonological features are stored in three auxiliary index tables in working memory. If an entry does not exist, it is created and added to the index table, together with the object's ID. If an entry already exists, only the ID is added. Number grouping is applied to the ordered (sub)sets of IDs as far as possible. Each combinatory type is hashed into a unique string instead of into an integer, because, theoretically, types are unlimited in size, which may yield too many collisions. Each word form and each semantic concept is hashed to a (non-unique) integer by applying a near-perfect hash function to their letters. The hash value is kept as small as possible by taking the letter frequency in Dutch into account (https://onzetaal.nl/taaladvies/advies/letterfrequentie-in-het-nederlands). This method is a variant of Huffman coding (Huffman 1952). Provisions are made for handling collisions which result from lexical ambiguities. In general, retrieving all IDs of objects that satisfy either the concept, type, or phonology constraint takes $O(1)$ time, when first-argument indexing is applied by the Prolog system to the respective index tables, and zero disk accesses. The index tables have a space complexity that is a linear function with a small factor – due to number grouping – of the number of templates. The index table on the feature that is used for ordering the objects has a space complexity that is a linear function of the number of

the feature's values. Three index tables are kept in working memory for fast access, in accordance with the Resource Principle.

3.5.4.2 Creating and accessing a meta-index table

Additionally, the semantic generator may select objects that are specified by constraints on other features. We do not demand maximum performance on queries of this kind. For each linguistic object, each value of each feature ever to be retrieved (not being concept, type or phonology) is stored, with its full path, in one auxiliary index table in working memory. This 'meta-index table' spans more than one feature. The IDs of all objects that specify the same value for some feature are grouped by number and stored in a metadata table, which is a stream. Additionally, the set of IDs of objects that do not specify any value for the feature is calculated and stored. The storage address for each feature-value pair is kept in the meta-index table. Retrieval of all IDs of objects that match one of these constraints takes $O(1)$ time, when first-argument indexing is applied by the Prolog system, and one extra disk access. The space complexity of the meta-index table is a linear function of the number of unique feature-value combinations, occupying only a fraction of the working memory, consistent with the Resource Principle.

3.5.4.3 General retrieval

Retrieval of linguistic objects is performed by executing a search task, specified by one or more constraints. When the search task is a graph – typically, an argument subgraph of some object involved in the semantic generation process – it is flattened into a series of constraints (paths). Each constraint is looked up in the appropriate index table. We distinguish between strict and liberal constraints, which demand objects to match the constraint explicitly, or allow objects that are unspecified for the constraint, respectively. Liberal constraints follow from graphs that allow under-specification. If a strict constraint is a restriction on semantic concept, syntactical type, or word form, the set of IDs of the objects satisfying the constraint is found immediately in the respective index tables. For other features, including liberal constraints, the set of IDs is found after issuing an extra disk access to the metadata table. In all cases, a constraint is replaced by a set of ID numbers corresponding to objects that are not inconsistent with the constraint. The objects that satisfy all constraints are identified by IDs that result from intersecting all sets of IDs. As the sets of IDs are ordered, determining their intersection is a linear function of the size of the biggest set. The function factor can be very small when ranges are involved, because only the edges of a range need to be inspected.

In general, however, the ranges get fragmented after a few intersection operations. Only the IDs in the resultant set of IDs may be accessed and loaded. The semantic generator randomly picks one of the IDs. Disk access is performed only for this ID, in order to locate and retrieve a complete linguistic object for further processing.
By applying directly accessible pre-compiled access and index tables, searching is reduced to looking up, giving a great performance boost; this contrasts with Prolog's own depth-first backtracking search algorithm, which is inefficient. In summary, finding one object, given its record number, takes $O(1)$ time and one disk access. Finding the set of IDs of objects satisfying a constraint on concept, type, or phonology only, takes $O(1)$ time and zero disk accesses. Finding the set of IDs of objects satisfying some other constraint takes $O(1)$ time and one disk access. Finding the set of IDs of objects that satisfy N arbitrary constraints takes $O(N)$ time and zero disk accesses.

3.5.5 Perspectives

The sum of all external index and external auxiliary tables is at present under 5 % of the total (externally stored) lexicon, measured in bytes. In the future we will explore how our way of organizing access to the stored lexicon can be maintained when the lexicon increases in orders of magnitude. It is expected that Prolog's internal management of huge numbers of index clauses is challenged with respect to access times. We leave that technical matter to compiler builders. More importantly, the lexicon, although it may take very large proportions, always stays finite. As the indexing tables are derived from the full-blown lexicon, there is always the possibility of having sufficient working memory available for accommodating them. Further, we want to investigate to what extent cognitive insights are reflected by this organization of the lexicon – a question which would not have been raised if we had applied standard techniques or third-party products.

3.6 THE LEXICON WHILE PARSING

In the DELILAH system, parsing is a process with three distinct ordered stages. Firstly, a chart parser or a backtrack parser determines the optimal analysis of the sentence in compliance with the categorial syntax. This is the combinatorial stage. Secondly, unification is exploited to combine graphs following the pattern of the syntactical combinatorics. This is the unification stage. Thirdly, the stored logical form is compiled out: the semantic stage.
The lexicon as a database is called for in the first and the second stages. In the combinatorial stage, syntactical categories are collected and tested for each word. These categories are specified in every lexical graph. For the combinatorics to work properly it is not mandatory but it is certainly efficient to maintain a register of combinatorial categories per word form, as an index to the lexicon. Note that the relationship between combinatorial categories and lexical graphs is not bijective: distinct graphs may share word forms and categories. In the first stage, the categorial index is the only emanation of the lexicon that is required. One or more syntactical analyses are chosen and these are transferred to the second stage.

In the second stage, every selected analysis is taken as a guide to unification. In bottom-up mode, appropriate graphs are taken from the lexicon and tested for unification. The lexical graphs are selected by their phonological and syntactical/combinatorial properties only. Successful unifications are stored in parallel, if necessary. At this stage, it is mandatory to keep track of argument subtemplates which are candidates for unification at certain points in the derivation. A transitive verb, for example, has two nominal arguments, and subject and object had better not be confused in unification. For this purpose, argumentative subtemplates (marked ARG) are uniquely indexed, and this index is reflected in the combinatoric category. For example, in (418) the index of the subject-argument reenters in the combinatoric category and in the Stored Logical Form; the index is highlighted in the following excerpt:

(420) *re-entering*

```
ID:A+B
...
SLF:{{[I&(B+J)#K,
       L&(M+N)#O,
       P@some^Q^and(quant(Q,R),laugh~[Q],event~[Q],
       entails1(Q,S),and(P,entails(Q,T)))&(A+B)#U],[],[]},
     and(and(and(experiencer_of~[U,K],
     some^V^and(quant(V,some, and(and(event(V),move(V)),
     theme_of(V,K)),goal_of~[V,O],entails1(V,incr),cause(Q,V)),
     entails(V,incr)),attime(U,W)),tense(U,past))}
...
TYPE:s_vn\0~[pp^0#B+M, np^0#B+Z, np^0#B+Y,np^0#B+J]/0~[]

ARG: ID:B+J
        PHON:G
        SLF:I
        SYNSEM:CASE:nom
               CAT:np
               NUMBER:sing
               OBJ:subject_of(A+B)
               PERSON:C
               QMODE:X
               THETA:experiencer_of
```

The indices are variables in lexical templates, but become instantiated under unification. They are pairs of natural numbers N1+N2: N1 is the second member of the higher index, N2 the first index of embedded arguments. Hence, chains of instantiated indices represent argument structure for the sake of anaphor resolution. In this way, the bookkeeping device of indexing argumentative subtemplates provides additional structural information.

When parsing, access to the lexicon is in real time. Every graph that may be useful is identified and found instantaneously, as described in the preceding section. The lexicon is supposed to be complete and indexed when parsing starts. As soon as a parse is selected on syntactical grounds, the relevant templates from the lexicon are retrieved and ordered according to the derivation of the chosen parse. Each template is selected on the basis of its phonological head – the value HEAD:PHON in the template – and its combinatorial category – the value of TYPE. These values are fixed in every lexical template. The phonological head corresponds exactly to the word form occurring in the sentence that is being parsed. The category corresponds exactly to the combinatory category that is applied for that word by the syntax in the present deriva-

tion. Below is the parse scheme of a simple three-word sentence. The relevant combinatory categories at word level are in bold face.

(421) Parsing *Jeroen wil werken* 'Jeroen wants to work'

(a) for all words in the sentence, retrieve their syntactic categories and make a chart

(b) choose the optimal parse(s) from the chart:

```
1-3+s\wh~[]/1~[]  jeroen wil werken
      1-1+np\0~[]/0~[]  jeroen
      2-3+s\0~[np^wh]/1~[]  wil werken
            2-2+s\0~[np^wh]/0~[vp^6]  wil
            3-3+vp\0~[]/0~[]  werken
```

(c) retrieve all lexical templates

```
[ ... HEAD:PHON:werken... TYPE:vp\0~[]/0~[] ...]
[ ... HEAD:PHON:wil... TYPE:s\0~[np^wh]/0~[vp^6] ...]
[ ... HEAD:PHON:jeroen... TYPE:np\0~[]/0~[] ... ]
```

(d) try to unify, according to the parse tree(s) of stage (b)

1. every retrieved *werken* template with every retrieved *wil* template
2. every unification resulting from step 1 with every retrieved *jeroen* template

(e) select the optimal unification result

Many of the lexical templates, though retrieved by their heads, will introduce constructions in the sense that the category and the template invoke other constituents, like the *wil* templates in (421). The procedure up to the selection of the optimal unification result is indifferent to whether the final semantics of these templates is compositional or not. Since every template carries its semantic receipt with it, unification is all that counts. For example, the light verb form *heb* 'have' of type `s\0~[np^wh]/0~[np^0]` introducing the *be hungry* construction unifies with the fully-fledged template for the NP *honger* 'hunger': its semantics comes along but is not represented at the top level. Step (421)(c), therefore, is perfectly general: in all cases, the constraints of phonology and applied combinatoric category suffice to retrieve all and only the relevant templates.

4. GRAMMAR: the reward of incompleteness

4.1 THE THREE DUALS OF GRAMMAR

The insight implicitly emerging from the previous chapters is that structure and meaning are non-compliant in a fundamental and principal way. The insight lies sheltered under the idea that, somehow, interpretation of well-formed sentences and assignment of well-formed sentences to given meanings must be computable, even if the ultimate semantic representation is no longer the result of derivation. The present chapter, however, sails under a different perspective. Here, we try to find out what the overall characteristics of the resulting formal system turn out to be. Here, grammar will be identified as incomplete *per se*, in a logical, almost Gödelian sense. Logical incompleteness will be acknowledged as a valuable feature of formal grammar, and as a mark of soundness. In order to ensure computability however, the lexicon will once more be constructed as a treasury of oracles that connect structures to meanings. As a main line, this chapter argues that constituents cannot derive entailments, or the other way round. Hence, some true statements about the relationship between a structured sentence and its interpretation cannot be proven within grammar. This particular gap between truisms about form and meaning and theorems on this relationship constitutes the incompleteness of the grammar. Moreover, the chapter says that logical incompleteness is a desirable property of formal grammar. Grammars that do not enjoy incompleteness are bound to underspecify syntax, semantics, or both.

In daily linguistics, syntax and semantics are different games, to almost every grammarian. Across frameworks, and also in the DELILAH system, syntax identifies and manipulates *constituents*. Constituents are the sentence's proper parts and are taken to be complex bundles of properties – they are complex

symbols or signs. Their interaction is modeled by a certain comparison of these bundles: *unification*. Sentential semantics is about the identification of *entailments*, or semantic consequences – Aristotle's heritage. The semantic contributions of constituents are taken to be functions; they are *composed* in order to provide the logical base for entailments.

Talking about a grammar as a whole, we have to deal with three duals. Firstly, we have to account for the balance between syntax and semantics, the *arts* of grammar. Secondly, we have to compare the *modes of operation* of syntax and semantics, to wit, unification of signs and composition of functions, respectively. Thirdly, we must evaluate the relationship between the *objects* of syntax and semantics: constituents and entailments. The present chapter is on the clash between the arts, the operation modes and the objects of grammar as a formal system. There is no doubt, however, that the theatre of the clash is restricted to grammar, and does not extend to human language.

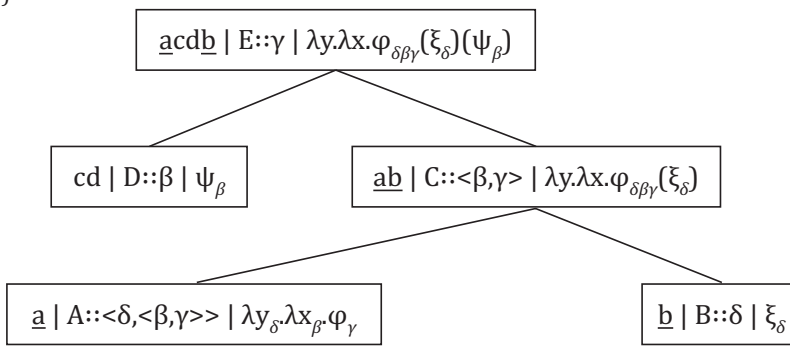
4.2 THE CONSERVATIVITY OF SYNTAX

4.2.1 The yield of syntax

Syntax serves a few good purposes. Firstly, it defines the proper parts of well-formed phrases, marking their roles: the *constituents*. Undoubtedly, they are the keys to grasping language. Secondly, syntax labels constituents, by classifying constituents into equivalence classes, the *categories*. Thirdly, syntax provides the *linearization* regime, inevitable since languages consist only of meaningful sequences of symbols. And, finally, syntax provides some grouping or storage of semantic terms: an *underspecified semantic representation* (cf. Bunt 2008). Below, you will find a picture of a syntactical derivation, which sketches the simultaneous computation of strings, categories and types and meanings.

For rules of grammar C *concatenates* A *and* B and E *infixes* D *in* C and a data triple *<string | category::type| meaning>*, the derivation of a discontinuous string *acdb* of category E is schematized by

(422)



In every step in the derivation, linear order, categories and types and storage of meaning are taken care of. This is why syntax deserves our permanent affection and attention.

4.2.2 The restrictedness of unification

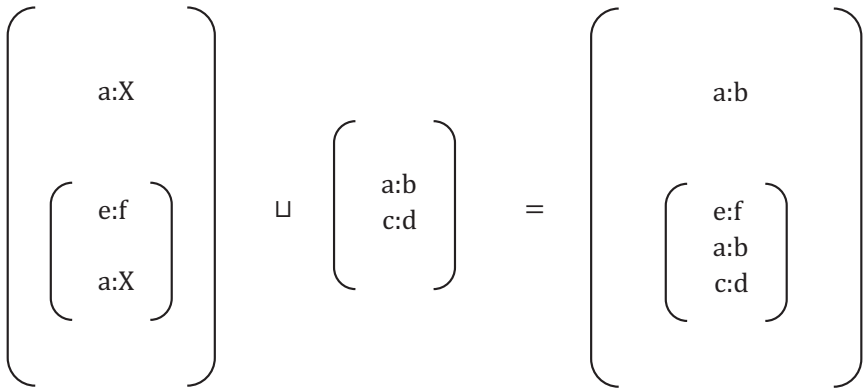
Syntax determines constituency by unifying complex symbols or signs. Unification is a well-defined operation (see *e.g.* Pereira and Shieber 1987) checking the compatibility of graphs or terms by seeking to create one graph or term out of two. Without loss of generality, below, we restrict our attention to graphs.
In the grammar of natural language, applying unification makes sense if four conditions apply:

(a) *binarity:* unification is an operation involving two graphs, yielding a derived graph in case of success
(b) *locality:* everything that is worth unifying is specified in the operanda
(c) *antisymmetry or strong typing:* for each two signs, there is a unique way of integrating two signs
(d) *recursion:* the output of one round of unification is the input of another.

Binarity is an economic principle. The unification of more than a pair of graphs is costly and possibly undecidable. Nothing in natural language calls for such a complication. Locality has to guarantee that all relevant information is available and can be exploited at crucial moments in the derivation of the covering structure. It is a principle of information transfer. Antisymmetry ensures different roles for each graph in the unification. These roles are in the graphs themselves, as part of the specification; therefore, these graphs are strongly typed: the type of a graph determines its role in a unification check-up. Recursion, finally, expresses that unification is all there is to syntax – the outcome

of one unification is the input of another instance of the same process. Below, the process is illustrated as the generalization of two structures; the process was discussed extensively in chapter 3 (cf. (395)).

(423) *Unification of complex signs*



One structure checks another by binarity and antisymmetry. The checking graph has a relevant substructure which is specified for features *e* and *a*. The first has value *f*, the second is undetermined or variable. The value of *a* in the substructure is bound to the value of *a* at the top level of the checking structure; this relationship reflects locality. The secondary structure – the one to be checked – is specified for features *a* and *c,* with definite values *b* and *d* respectively. The specifications for the substructure and the secondary structure are compatible: *e:f* and *c:d* do not affect each other, both occurring in just one sign, and the value X of *a* in the checking sign is instantiated by *b* in the secondary sign. Because no incompatibilities arise, the resulting unified structure specifies all information in one new graph, which is open to unification with other graphs, by recursion. The resulting structure is at least as specific as either of the two operands.

As described here, unification is a standard tool in all lexicalist approaches to grammar (*e.g.* Sag, Wasow and Bender 2003, Bouma 1993, Kay and Fillmore 1999). These approaches differ mainly as to the embedding and organization of unification and – of course – the nature and the logic of the features specified. Unification, as Sag *et al.* note, is not the theory of grammar – it is its chariot.

4.2.3 The generality of unification

Syntax is both constructive and conservative. It specifies how phrases combine to make new phrases in such a way that the phrases entering the combination survive as proper parts in the whole. In syntax, phrases are preserved under combinatorics. Syntax does not delete – it builds. For that reason, the logic of categorial grammar, for example, is intuistionistic, constructionist and resource-sensitive (Van Benthem 1991). For that reason, syntax does not have a complement type of operation. One cannot nullify structure.

Combining phrases syntactically amounts to checking their properties, finding some linearization and computing the properties of the whole out of the properties of the proper parts. It is an essential feature of all major syntactical frameworks that the combinatorics does not create structure independent of information carried by the operands. This is even true for those syntaxes that embody theories of language, like minimalism (Chomsky 1995). The definition of its single (or main) syntactical process *merge* is like (424) rather than like (425), according to Chomsky (1995), Collins and Stabler (2011) and Stabler (2010).

(424) merge(α, β) = {β, {α, β}}
(425) merge(α, β) = {γ, {α, β}}.

The result of combining two things syntactically is projected from the objects combined, to wit, α and β. The resulting structure is not projected out of the blue. It is not surprising, then, that independently formulated phrase structure grammars with production rules have an equivalent formulation in a more lexicalist framework, like categorial type logic: if a category A expands to B and C according to some phrase structure grammar, one can equivalently state that members of category B map category C to category A, making the category B combinatorially obsolete. Phrase structure rule (426) expresses the very same syntactical information expressed by functional application rule (427).

(426) φ A ψ → φ B C ψ
(427) φ A ψ ⇐ φ A/C C ψ

In (426), the top label A is not necessarily associated with the daughters' labels. In (427), the top label is introduced by one of the daughters, by definition. The relation between phrase structure grammars and categorial or type-logical grammars has been studied since Bar-Hillel (1953). Pentus (1993) proved the (weak) equivalence of context-free phrase structure grammar and

the Lambek-calculus – restrictions of type-0 grammars and of type-logical grammars respectively. The mapping between minimalistic and categorial grammar has been studied by Retoré and Stabler (2004) and Vermaat (1999). What matters here is that in (424) and (427), the top node – the new constituent, the unification, the whole – does *not* introduce new information. The labeling of the new constituent comes with its proper parts.

Grammars do not describe the manipulation of categories, though, but the manipulation of richly structured phrases, containing all kinds of information. Let us call these phrases of a certain category *complex symbols* or *signs*, as was suggested by, among others, Friedman and Bredt (1968), Gazdar, Klein, Pullum and Sag (1985), Morrill (1994), and many others. Under that view, common to almost all present approaches to syntax, rules of grammar with formalizations like (424) or (427) specify how complex symbols arise from other complex symbols. Linguistic theories differ considerably – though not fundamentally – in the specification of complex symbols, as to the nature and the amount of information stored in the complex symbols, and as to the alleged impact of the representations. Discourse representation theory (Kamp and Reyle 1993), for example, is much more specific on anaphoric effects than are other approaches to grammar. Yet, the basic process that lives on all these resources is *unification.* Unification is a costly, but finite procedure to check point by point whether the requirements that one constituent imposes on its linguistic environment are met by candidates for that environment. The division of labour between the signs is that one sign specifies conditions for the other, while the other specifies values in the checking sign. If all relevant specifications match point by point, the unification succeeds. If one specification runs counter to the other, unification fails. If specifications do not interfere, they go along with the process. Information is neither destroyed, nor left behind. At most, some general information is replaced by more specific information of the same sort. In particular, if A is a constituent of B and unified with it, subsequent unification of B with another constituent *cannot* destroy the position of A with respect to B. Essentially, the unification of A and B is at least as informative or specific as both components; it is conservative.

Note that unification does not impose any restriction on the nature of the constraints. It is important to see that even semantic information can be checked and transferred this way. However, the level of semantic representation that can be reached by applying unification is by conjecture exactly the level that is called *underspecified semantics* in computation (Bunt 2008). It is a kind of storage of typed semantic contributions where the ultimate inter-

action between these contributions is not specified. Similar approaches to delayed application have been proposed by Cooper (1983), Janssen (1986), Bos (2001), Copestake *et al.* (2005) and Jäger (2005).

Syntax is conservative, because all its aims can be fully achieved by unification and unification does not destroy any information. Successful unification yields a conservative merge of complex symbols. Information is added to other information; no information is lost. Not all information will be available at all levels of embedding, but that is not the point: information is not expelled by other information, and the level of specificity is monotonously increasing: with each unification new information comes in. For this reason, we can call syntax an *additive* process. It is the grammatical instantiation of addition and union. It defines an algebra in which the smaller units are ordered below the larger ones. If two complex signs unify, both define proper constituents of the resulting phrase. In the following, the symbol $\sqsubseteq$ reads as 'is a constituent of '; this notion is antisymmetric.

(428) If A and B are complex signs and $A \sqcup B$ is their unification, then both $A \sqsubseteq A \sqcup B$ and $B \sqsubseteq A \sqcup B$.

Lemma (428) we call *additivity* for syntax, as it compares syntax to set union and partial (transitive, antisymmetric) ordering over sets, as well as to addition over $\mathbb{R}+$ and its ordering:

(429) If A and B are positive real numbers and A + B is their addition, then both $A \leq A+B$ and $B \leq A+B$.
If A and B are sets and $A \cup B$ is their union, then both $A \subseteq A \cup B$ and $B \subseteq A \cup B$.

In additivity for syntax, the empty complex symbol is the unit, but the grammar will never schedule the empty complex symbol for unification, as it does not represent any syntactical object. The syntactical representation of a sentence with constituents A_1 .... A_n is therefore $(...(A_1 \sqcup A_2) \sqcup ..... \sqcup A_n)$ or $\bigsqcup_{i=1..n} A_i$. It represents a conservative and preserving merge of information, scheduled by rules of grammar.

4.3 THE DESTRUCTIVITY OF SEMANTICS

4.3.1 Divorcing constituents and entailments

Natural language semantics west of the Indus starts with Aristotle. The *Organon* describes how two independent sentences logically determine a third one. Here is, as an example, the syllogism dubbed *Cesaro,* combining a negative (e), a universal (a) and an existential (o) quantifier. A negated existentially quantified sentence and a universal quantified sentence jointly entail an existentially quantified sentence with predicate negation. The conclusion takes its predicate negation from the negative quantifier in the first premise, its left predicate or middle term from the second premise, the positive version of its right predicate from the first premise, and its quantifier out of the syntactical blue. The abstract syntactical structure in Germanic languages for this syllogism is indicated below.

(430)	No humans have hooves	$[_{dp}$ Q2 P] M
	All horses have hooves	$[_{dp}$ Q1 S] M
	∴ Some horses are not humans	$[_{dp}$ Q3 S] Q2-P

The *conclusion* is entailed by the premises – one cannot consistently assert the premises and deny the conclusion. That is, all speakers of a language will confirm that they cannot think of a situation in which the following complex description in their language would be true, that is, in which it could be anything other than a desperate expression of nonsense.

(431) No humans have hooves and all horses have hooves but some horses are humans.

Natural languages – even Pirahã, we must assume – share these syllogistic three-tuples, because all languages have the partial ordering coming with a universal quantifier (Zwarts 1983) and they have negation, and that should do. In most languages, these quantifiers are conservative, restricting the domain of quantification to the nominal property, and go with that property syntactically, rather than bridging between the expressions of that property. And in all languages, the single conclusion is built from several syntactically specified but unrelated constituents. Despite all mediaeval classifications of syllogisms in term of subjects, predicates and midterms, however, Aristotle's is not a theory on the syntax-semantics interface. From the constituent structure of the premises one cannot predict the conclusion. From the conclusion

one cannot deduce the structure of the premises. The relationship between constituency and entailment is very loose. It is not possible to map constituents and entailments functionally.
From its very inception as a field of empirical study, semantics went beyond syntax.

4.3.2 Entailments as products of composition and deduction

Semantics is invasive and destructive, and its operations may not even be computable (Lasersohn 2009). The semantic representation of a sentence has to take care of all those properties of a sentence that are not determined, or are under-determined, by its structure. It does so by bringing certain terms under control of operators outside that term, thereby affecting the independence of the semantic contribution of the term *per se*. The intensional independence of the term is reduced to denotational dependency when the term is absorbed in the proposition. That is what the 'extra-grammatical' meaning postulates of Montague (1971) are about. The destructive nature of semantic composition emanates in two related but distinguishable phenomena: scopal control – affecting the interpretation of operator-sensitive terms – and phrasal or structural ambiguity – the fact that one syntactical structure may come with several distinct interpretations. Scopal control changes the way in which a phrase contributes to the meaning of the whole. Ambiguity of the kind that occurs in many *extended lexical units* where a certain grouping may or may not introduce a specialized reading can overrule the individual lexical elements completely: in one reading of *John kicked the bucket,* there is neither a bucket, nor does any kicking occur. To the extent that syntactical structure and categorization is unaffected by the outcome of the interpretation, scopal control and ambiguity turn semantic terms into dynamic warriors, striving after reign instead of just preserving *part-of*-ness, as in syntax. If two terms φ and ψ each contain an operator, and if these two operators can dominate each other, the possible relations between the operators have to be checked when φ and ψ are applied to each other. If a phrase [A B] in *X A B Y* can contribute, for example, the application of the meaning of A to the meaning of B, it has to be checked whether it had not rather contribute some other meaning, not arising from functional application. A classic example here is the *way* construction (Goldberg 1995). In Dutch (cf. Poß 2010), intransitive verbs can productively introduce a causative construction when combined with expressions of path and direction and a reflexive – this construction was already mentioned in sections 3.1.4 and 3.4.4. Informally, a Dutch intransitive verb

V creates a fully-fledged paradigm {*V+reflex+een weg naar DP}*, meaning: by V-ing achieve to end up in DP. Here, we repeat a slightly modified example from chapter 3, with intransitive *golfen* 'play golf' as verbal head.

(432) Elke bankier had *zich een weg naar* een Raad van Bestuur kunnen *golfen*
Every banker had himself a way to some Board of Directors been-able play-golf
'Every banker could have succeeded in moving into some Board of Directors by playing golf'

Note that instead of *golfen,* any other intransitive verb could occur, maintaining the interpretative framework. In this sentence, *[een weg naar een Raad van Bestuur]* is a constituent. Whether and, if yes, how this constituent contributes to the meaning of the whole, and also whether both occurrences of *een* are interpreted as quantifiers at all, entertaining scopal relations with the universal quantifier and modal, for example, completely depends on the details of the construction's embedding, and the way the relevant meanings are stored.
The interpretation of a sentence is both kingmaker and exterminator. The semantic representation of a sentence – its Logical Form (LF) – is accountable for all and only its *entailments*. Among grammarians, this concept of Logical Form has already been revealed in Higginbotham (1985). Entailments are the logical consequences of a sentence, that is: those consequences that are independent of the context of utterance or knowledge of the world. That is why they live in the domain of grammar. Entailments are intensionalized or necessitated implications, induced by knowledge of language. Aristotle's syllogisms are the prototypical examples of entailments. The semantic representation of *All human beings are mortal* and *Socrates is a human being* must be such that from their conjunction the proposition *Socrates is mortal* can be derived with purely formal, logical manipulations. For a sentence like (433), the semantic representation must be such that each of the independent sentences in (434) can be derived from it, by logical manoeuvres and grammar alone. At the same time, the semantic representation or logical form must resist the deduction of any of the sentences in (435).

(433) Not every farmer beat his donkey
(434) Some farmer owned a donkey
Some donkey was not beaten
Some donkey that relates to a farmer was not beaten
Some farmer did not beat a donkey that was related to him
Someone did not beat
Some creature was not beaten
Some donkey was not beaten by any farmer to whom it relates
...

(435) No donkey was beaten
Every donkey was beaten
Some donkey was not beaten by any farmer
…

The sentences in (434) are simultaneous consequences of (433). Of course, these consequences are relative to some well-defined calculus of propositions. To be a semantic consequence is to be derivable by some logic. The formal tendency in modern semantics is motivated only by the fact that formal logics and algebras provide mechanic deduction: the theory of grammar needs these mechanics to justify the inevitability of entailments observed by Aristotle. Therefore, the logical form of a sentence can be defined with respect to its entailments. Its ideology is captured by the following statements, requesting some consistent logic.

(436) *Logical Form of S*
The logical form of a sentence S is that representation ⟦S⟧ for which a logic L exists such that S entails P iff $⟦S⟧ \vDash_L ⟦P⟧$ and $⟦S⟧ \vdash_L ⟦P⟧$

(437) *Meaning of S*
The meaning of a sentence S is the conjunction of its entailments derivable from its Logical Form

These definitions have several edges. Firstly, they relate logical form to linguistic empiricism. Entailments are those aspects of sentential meaning on which language users converge. Entailments, therefore, are sentences, not formulas. With his syllogisms, Aristotle did not discover new cognition – he just made explicit the oldest insights of talking heads. Human beings may disagree on the meanings of words, but entailment is a decidable property among linguistic laymen, as is claimed correctly by Chierchia and McConnell-Ginet (2000).
Secondly, the statements define a lower bound to logical form. It invokes formal reasoning to connect entailments to logical form, by assuming that some entailments are so elementary that they do not have any other entailments themselves. Logical form must be so rich that formal deduction of entailments is possible, according to some logic.
Thirdly, we can use (436) and (437) to define an upper bound to the notion of entailment. Entailments are those propositions that are connected to sentences by judgements of language users. We assume that language users only have (linguistic) judgements about propositions that are conceptually – say, lexically – related to the base. Few language users are likely to acknowledge entailments outside the lexical realm of the base – only the trained logicians among them are in danger of doing so. We assume that all the material in entail-

ments of a sentence is delivered by the lexical components of the sentence. Every entailment is built completely from semantic material available in the sentence. Because of this – quite realistic – restriction, we can neglect entailments produced by purely logical combinatorics, like the logical equivalence of *p* and *p & p, p → p, p & (p ∨ ¬p)* and so on. We define an entailment of A as the shortest representative of its logical equivalence class within A's lexical realm.

A sentence entails all of its entailments simultaneously. As a matter of fact, definition (436) gives rise to the following conjecture:

(438) If P_1, ... P_n are all entailments of S, the logical form ⟦S⟧ is equivalent to the conjunction ⟦P_1⟧ & .... & ⟦P_n⟧

Since entailments may be complex propositions themselves, the explicit conjunction of all entailments will be redundant. Still, the shorter and more economical representation is basically conjunctive. That is, if every ⟦P_i⟧ itself is spelled out as a conjunction $p \,\&\, ... \,\&\, p_n$, the representation ⟦S⟧ is the conjunction of all those propositions p_j that occur in the representation of at least one P_i. The logical form, then, is a kind of conjunctive union of all entailments. The idea that logical form is basically conjunctive is advocated by Pietroski (2006, 2011) from a purely linguistic point of view, and by Reckman (2009) from a computational-linguistic point of view. It is advocated and integrated in the DELILAH semantics in chapter 2 of this monograph as Flat Logical Form.

For the logical form to be the source of all entailments, it combines semantic contributions of phrases by applying one meaning to another, bringing one semantic contribution under the logical influence of another. In particular, semantic contributions can be excluded from inference by wrapping them into intensional domains, or be made referential dependent upon other semantic terms. Even if a phrase is a proper part of a sentence, its semantic contribution to the sentence might be non-existent, mutilated or only partial. In exactly this sense, semantic combinatorics is *multiplicative* and *selective*. It behaves like intersection, in accumulating interpretative conditions on the individual terms. From the consideration above and the conjunctive nature of the logical form, it follows immediately that the notion of *semantic consequence* ⊰ induces an antisymmetric and transitive ordering between the logical form and any entailments of the sentence; in the definition, ". ⊰ ." abbreviates the conjunction of ". ⊨ ." and ". ⊢.".

(439) If ⟦S⟧ is the logical form of S and S entails P, then ⟦S⟧ ⊰ ⟦P⟧

In this ordering, the whole – the conjunction of entailments – is ordered 'below' the proper part – the individual entailment. Its antisymmetry follows from the restriction mentioned above, that entailments are built from lexical material in the sentence and that they are the 'shortest' representatives of their equivalence class.
With respect to the ordering, it is of some importance to note that the notion of semantic consequence is not conservative in the sense in which unification is (cf. (423)). In particular, extension of the logical form of a sentence does not preserve semantic consequences the way extension of the syntactical form preserves constituency.

(440) *Non-preservation of semantic consequences*
If $[\![S]\!]$ is the logical form of S, S entails P and $[\![S]\!] \dashv [\![P]\!]$, there are embeddings $S \sqcup E$ of S such that $[\![S \sqcup E]\!] \dashv [\![P]\!]$ is not valid.
If $\mathbf{P_S}$ is $\{P|[\![S]\!] \dashv [\![P]\!]\}$ and $\mathbf{Q_E}$ is $\{Q|[\![E]\!] \dashv [\![Q]\!]\}$, then $\mathbf{R_{S \sqcup E}} \subseteq \mathbf{(P_S \cup Q_E)}$

Among these syntactically conservative embeddings that do not preserve semantic consequences we place all those phrases which are called *plugs* in the literature on presuppositions and their projections. A ready example is a propositional frame introduced by an intensional verb, taking a proposition as its complements. Embedding a proposition under an intensional verb may not preserve its presuppositions, which would have been entailed otherwise. Even more transparent is the operation of disjoining sentences: whatever is entailed by S is not entailed by *S or P* unless it is also entailed by P. For this reason – a conservative embedding of the whole does not preserve entailments – the ordering imposed by the notion of entailment is the opposite of the ordering imposed by constituency.

The conjunction of entailments of a sentence is not projected from its syntactic structure. Nothing in the syntax gives rise to flat representations. What is more, not every constituent, not even every constituent at a certain level of derivation, gives rise to any particular entailment. An argument of a verb, for example, does not induce any entailments of its own, although it is generally the prototypical constituent. Quite the opposite is true. Entailments – testable entailments – are almost exclusively produced by combining the semantic contributions of *several* constituents. In order to produce entailments, the semantic contributions of constituents apply to each other. The agenda for these applications is dictated by the constituent structure – the syntax – but the outcome is not dictated by syntax. The result of applying two semantic contributions to each other is solely defined by their internal structure, since the application itself – mostly modeled by lambda conversion – is too gen-

eral to calibrate entailments. Since Montague (1972), we have been taking the lambda terms themselves as introducing enough structure to guarantee a structured outcome.

Therefore, we must assume that some non-conservative but very restrictive process of applying semantic contributions is active in order to produce the conjunction of entailments. This process is *functional composition*.

(441) If f_i is the lambda term associated with constituent c_i of S, i.e. $f_i = [\![c_i]\!]$, there is some protocol according to which the composition $\prod_{i=1..n} f_i$ or $(\ldots (f_1 \circ f_2) \circ \ldots \circ f_n)$ is arranged such that the result of applying that protocol to the composition is the logical form $[\![S]\!]$.

The protocol that is assumed accounts for both the syntactical structure and the invariants of the underspecified semantic representation. Being sensitive to syntactical structure as well as the underspecified semantic representation, the protocol embodies the compositionality of interpretation. Since constituents and entailments do not match, the protocol accounting for the relationship between the two is neither a syntactic nor a semantic computation, and it is not restrictive at all. Such a protocol was alluded to in chapter 2, taking care of the transformation of Stored Logical Form into Applied Logical Form and Flat Logical form. Note however that the conjunctive result of the composition is not caused by the protocol, but to the internal structure of the applied terms themselves. The result of the composition is a product, with all kinds of terms restricting each other. The nature of this composition is best understood by considering that an entailment cannot be traced back to a constituent and that all entailments are simultaneously entailed by the whole.

4.4 THE DENIAL OF STRUCTURE

In the previous section, interpretation was defined as a procedure to extract the necessarily underspecified semantic representation from the unified syntactical structure, to identify the contributions per constituent and to apply these contributions to each other. The result must be a conjunction of propositions – the entailments. This leads to the following overall characterization of the 'syntax-semantics interface.' The interpretation of an additive constituent structure implies the multiplication of the interpretation of the constitu-

ents, by the protocol referred to in (441); the latter 'spells out' a conjunction of propositions, by pure

(442) *Interpretation*
$[\![(...(A_1 \sqcup A_2) \sqcup ..... \sqcup A_n)]\!] \Rightarrow (...([\![A_1]\!] \sqcap [\![A_2]\!]) .... \sqcap [\![A_n]\!]) \Leftrightarrow p_1 \,\&\, ... \,\&\, p_k$

It shows that the interpretation inverts the additive algebraic structure of syntactical form – the additive, conservative, highly structured unification of complex signs – into the multiplicative logical form – an invasive, flat chaining of small clauses, conjoining to fully-fledged entailments.
The directionality of the interpretation process does not mean that in grammar the inverse process does not exist. In fact, we identified the process of generating sentences from independent meanings as a form of 'backpropagation' of this kind; the operation is referred to by the sign $\langle\!\langle . \rangle\!\rangle$.

(443) *Generation*
$\langle\!\langle\, p_1 \,\&\, ... \,\&\, p_k \,\rangle\!\rangle \Rightarrow \langle\!\langle (...([\![A_1]\!] \sqcap [\![A_2]\!]) .... \sqcap [\![A_n]\!]) \,\rangle\!\rangle \Rightarrow$
$(...(\langle\!\langle A_n \rangle\!\rangle \sqcup \langle\!\langle A_2 \rangle\!\rangle) ... \sqcup \langle\!\langle A_1 \rangle\!\rangle)$

Given some simultaneous semantic constraints, generation retrieves some composition of meanings compatible with these constraints and a unification of the associated complex symbols. This picture is misleading, however, in that functional composition is not procedurally 'in-between' the conjunction of entailments and unificational structure. Rather, the composition arises from unification, and is supposed to convert to the original conjunction. Generation is about finding out whether there is some composition to provide the original, structure-independent conjunction.

(442) and (443) define the grammatical cycle from form to meaning and back as the consecutive application of the operations $[\![.]\!]$ and $\langle\!\langle . \rangle\!\rangle$. The first operation assigns meanings to forms; the second assigns forms to meanings. Their composition is antimorphic: it maps addition onto multiplication and *vice versa*. That is what negation does in the calculus of propositions, according to the De Morgan laws, or what complementation does on sets.

(444) $\neg(p \vee q) \quad \leftrightarrow \quad \neg p \,\&\, \neg q$
$-(A \cup B) \quad = \quad -A \cap -B$

Negation and complementation are both antimultiplicative and antiadditive. Neither syntax nor semantics, however, are complete Boolean algebras, hosting both upward and downward operations and complementation. At best,

syntax and semantics are monoids, generated by one operation. Syntax is defined by a conservative, addition-type operation (say: concatenation-by-unification), for which complementation is not defined. Semantics is defined by an invasive, multiplication-type operation – function composition – for which complementation and identity are defined. Interpretation is some mapping from one additive monoid onto another multiplicative monoid. Thus, interpretation, like negation and complementation, is antiadditive. That is why we can say that semantics accounting for inference denies unification structure, while observing compositionality. In the same vein, generation is antimultiplicative, by letting structure negate meaning. Because generation is even less understood than interpretation, for the rest of this chapter we will stick with interpretation.

Let us refer to (442) as the *antiadditivity of full interpretation*. The antimorph nature of full interpretation has consequences for various approaches to natural-language grammar and natural-language processing. The main consequence is this. The concepts with which we structure syntax are quintessentially different from the concepts that buy us semantic inference. There is no monotony when going from unification to entailments, or the other way round: from meanings to sentences. This affects shallow processing, for example. Shallow processing with ambitions beyond tagging phrases lives on the idea that somehow aspects of meaning arise from counting and recounting and modeling co-occurrences. The means by which we can make educated guesses on the form of sentences up to the identification of referential objects, however, cannot be monotonously extended to cover entailments. They serve different algebras. The antimorphism of full interpretation also affects those grammatical theories that appeal to *deep* syntactical operations for meaning to come about, like the 'classical' theory of logical form in generative grammar.

4.5 THE MISMATCH OF STRUCTURE AND MEANING

The idea that the meaning of a sentence is determined by its construction is generally attributed to Frege, but wrongly so, according to Janssen (1983). Yet, as a mathematician, Frege must have been well aware that the reference of a complex phrase somehow lives on the way in which its parts contribute to that reference, and that this contribution is delivered by syntax. For it makes little sense to assume that the referential potential of a sentence is not somehow

dependent on its construction. This does not imply, however, that the building blocks of syntactical construal match with the jewels of semantics. As a matter of fact, it is not difficult to show that in many constructions entailments and constituents diverge seriously. A main source of incongruence is type incongruence: entailments are propositions, but most constituents do not project propositional meanings. Take, for example, DPs. Most DPs have meanings that are only remotely related to propositions. Nominalizations, however, may introduce propositions, which can even be entailed. Below, three examples of embedded nominalizations in the relatively independent subject position are given, with an infinite scheme for a proposition projected by it.

(445) The destruction of the city by the enemy is solicited by our own army.
(446) The king welcomed the destruction of the city by the enemy as a blessing.
(447) The destruction of the city by the enemy was finally avoided by a simple trick.
(448) The enemy destruct-X the city.

For two reasons, a possible entailment based on the nominalization is not independent. Firstly, all virtual entailments based on the nominalized subject need tense, which can only be derived from scrutinizing the tense and the aspect of the matrix. And, secondly, if tense and aspect can be borrowed for some enemy-destroying-city-type of proposition, the veridicality profile of the matrix completely determines whether this proposition – or even its denial – is factually entailed. Therefore, not even a propositionally inclined DP projects 'its' entailments autonomously.

In this section, we will consider three constructions in which the mismatch of constituents and entailments is evident, although sentences and propositions seem to be available. In exception phrases, entailments pop up which almost look opaque if one focuses on the lexical material. In comparatives, entailments are suppressed that seem almost inevitable. In ellipsis, entailments are evident without constituency available to carry that load. It is not accidental that these sample constructions are all related to coordination. Recall that entailments are somehow – covered – coordinates of logical form. When the syntactical configuration gives rise to its own explicit coordination, the clash between form and meaning can hardly be overlooked. In Cremers (1993a), it was argued that the procedures for reconstructing free explicit coordination are in fact meta-grammatical (see also section 1.7.4).

4.5.1 The mismatch in exception phrases

In exception phrases, the referent of a constituent that is licensed by an *exception*-like coordinator is somehow set apart with respect to a predication. They come in two modes. Either the constituent is excluded from predication over a class, or the constituent is included in the predication. Here are Dutch examples of the first (exclusive) and the second (inclusive) mode, respectively.

(449) Op een historische roman na heb ik Marie haar boeken allemaal gelezen
'except for a historical novel have I Marie her books all read'
I read all Mary's books with the exception of a historical novel.
(450) Henk heeft behalve met Dylan (ook) veel opgetreden met The Boss
'Henk has except with Dylan (also) much performed with The Boss'
Apart from Dylan, Henk (also) performed a lot with The Boss

Exception phrases have some intriguing syntactical-semantic details. The set of items that can introduce the excepted constituent is limited. Often, they look like prepositions, but semantically, they behave like coordinators. In particular, they are sensitive to the polarity of one constituent in the main clause, which acts as a counterpart to the constituent that is excepted. If this constituent is upward entailing with respect to its internal domain, the exception phrase is interpreted inclusively. Otherwise – for example when the counterpart is a universally quantified phrase – the exception phrase is interpreted exclusively. In both modes, the exception phrases can occur in almost all positions in which adverbial adjuncts can occur, without change of meaning, though with possible consequences for information structure. Moreover, the complement to the excepting operator can be an elliptical phrase, which complicates the analysis considerably. Both the positional freedom and the elliptical option are demonstrated below.

(451) Ik heb geen filosoof alcohol durven schenken, behalve Arie wijn.
'I have no philosopher alcohol dare present, except Arie wine'
I did not dare to present alcohol to any philosopher, except for wine to Arie
(452) Ik heb behalve Arie wijn geen filosoof alcohol durven schenken.
(453) Behalve Arie wijn heb ik geen filosoof alcohol durven schenken.
(454) Ik heb, behalve Arie wijn, geen filosoof alcohol durven schenken.

To appreciate the mismatch between constituency and entailments, we focus on a simple exception sentence in an exclusive mode, assuming some bracketing which indicates syntactic form while abstracting from the labelling. The structured alternatives are listed in (455) - (456), with the except phrase occurring post-verbally, pre-verbally and adnominally, respectively.

(455) Alle filosofen waren dronken behalve Socrates.
All philosophers were drunk except for Socrates.
(456) [[[Alle filosofen] [waren dronken]] [behalve Socrates]]
(457) [[Behalve Socrates] [waren [alle filosofen] [dronken]]]
(458) [[[Alle filosofen] [behalve Socrates]] [waren dronken]]

Each of the syntactical alternatives has three prominent entailments, presented here in Dutch.

(459) [[Niet alle filosofen] [waren dronken]]
Not all philosophers were drunk
(460) [Socrates [was [een filosoof]]
Socrates was a philospher
(461) [Socrates [was [niet] dronken]]
Socrates was not drunk

In former days, one could have maintained that for the simple reason that three differently structured sentences share their semantics, the three sentences must have a common syntactical ground. In modern syntax, this reduction would involve some operations like *extraction-from*-DP or *insertion-into*-DP, which hardly have an independent motivation and require a serious complication of the combinatorial equipment, beyond control. In any case, the common derivation would be inspired by semantic similarities, not by syntactic processes – there are no other structures in which a proper part of a DP (as in (458)) ends up or begins as a leftmost specifier (as in (457)), without semantic consequences.
Since it is unlikely, therefore, that in serious syntax the three options are in each other's derivations, we had better look at the projection of the syntax on the entailments. Note, firstly, that none of the entailments is entailed by the matrix sentence (without the exception phrase) clause alone. The constituent *Alle filosofen waren dronken*, though a proposition as such, does not give rise to any independent entailment. This already shows that being a constituent, or even being a constituent of the right type at top level, is not a sufficient condition for providing entailments.
Constituency is not even a necessary condition for a phrase to provide, or to contribute to, an entailment. Scrutinizing the major parts of the entailments, it is evident that none of the following phrases quintessentially occurring in the entailments can be derived syntactically from proper constituents of the original, for example by reversed unification or decomposition. The following constituents of the entailments are syntactically untraceable.

(462) Niet alle filosofen *not all philosophers*
was niet dronken *was not drunk*
was een filosoof *was a philosopher*
niet *not*

Remarkably, *none* of the immediate entailments can be constructed as an amalgam of proper constituents of one of the entailing sentences. As a consequence, we can safely state that there is no transparent relation between constituents of an exception sentence and its prominent, non-trivial entailments.

4.5.2 The mismatch in comparatives

Comparatives are common constructions among the languages of the world. Though the syntactical variety is huge and subtle at the same time, the characteristic overall pattern can be captured by a configuration in which something is predicated over something with a certain degree that is implicitly compared to the degree to which the predicate applies to something else. It is a common feature of comparatives that there is only one explicit grade phrase in the sentence which is in a construction with some comparison particle. The second grade phrase is predominantly absent – a phenomenon known as subdeletion. The interpretation of these constructions, however, involves three propositions: the original graded predication, the secondary graded predication and the proposition containing the comparison (Von Stechow 1984). Below you will find the syntactical and semantic schemes of a simple Dutch example:

(463) [[... grade1...predication1] [comparison phrase ... predication2]]
[[Jan wandelt sneller] [dan Kees fietst]]
Jan walks faster than Kees bikes

(464) ⟦grade1 predication⟧ and ⟦grade2 predication⟧ and ⟦comparsion of grades⟧ ≈
⟦John walks at a certain level of speed⟧ and
⟦Kees bikes at a certain level of speed⟧ and
⟦the level of speed at which John walks is higher than the level of speed at which Kees bikes⟧

The most intriguing aspect of comparatives is that neither of the two sentences – elliptic or full – involved is entailed as it stands. The first sentence contains an expression of grade – in the example above, a morpheme – that is not part of the proposition induced as an entailment by that first sentence. The sentence does not entail that Jan walks fast. The second sentence does not contain a grade expression but it is not entailed either. The third proposi-

tion is a derivative of the relation that the comparison phrase imposes on the degrees to its left and its right – present or not – but is as such not projected by any major constituent of the original sentence. The fact that three different propositions are entailed by the construction and that these entailments are partly induced by sentential-looking proper parts of the constructions underlines the analysis of comparatives as coordination, rather than subordination, by Hendriks (1995) and others.
For some comparative constructions, it is even difficult to maintain that they have an underlying clausal structure.

(465) You are better than no one (and no one is better than you)
(from Bob Dylan's *To Ramona*)

This line is meaningful – expressing that everybody is at least as good you are – but no clausal extension is.

(466) # You are better than no one is / has ever been / could ever be

The meaningful occurrence of the negative quantifier complies with Hoeksema's (1983) observation that non-clausal comparatives do not license negative polarity items, where clausal comparatives do. Notwithstanding its non-clausal syntax, its interpretation still involves a conjunction of comparison clauses.

(467) ⟦You are better than no one⟧ = ⟦you are good to some degree⟧ and ⟦everybody is good to at least that degree⟧

Clearly, the second clause cannot be derived by syntax from the *than* phrase.

What we observe here is that there is some evident underlying sentential structure in the construction, but that these sentences quintessentially deviate from entailments in syntactically almost inaccessible but semantically prominent ways. The deviation is even more prominent in simple adjectival predication. Sentence (468) entails neither that Jan is smart nor that Marie is rich.

(468) [[Jan is slimmer] dan [Marie rijk is]]
'Jan is smarter than Marie rich is'
Jan is smarter than Marie is rich

The incongruence between sentential structure and entailments – propositional by nature – is not an accidental feature of comparatives in Germanic languages. It is widespread, and it is even a defining property of the construc-

tion in the languages of the world. Here too, syntax almost misleads semantics, and semantics contains few clues as to syntactical structure.

4.5.3 The mismatch in ellipsis

Gapping is the prototypical source of incomplete sentences, interpretable only in heavily conditioned, language-dependent verbal contexts. The verbal context is typically provided by coordination-like ordering. If these conditions are met, the interpretation is perfect. Here are some typical examples of gapped structures, indicated by italic type.

(469) Ik weet dat de jongens geen boeken willen en *de meisjes geen films*
'I know that the boys no books want and the girls no movies'
I know that the boys don't want books and the girls don't want movies

(470) Komt Henk vandaag met de trein? Nee, *morgen en met de fiets*
'Comes Henk today with the train? No, tomorrow and with the bike'
Does Henk come by train today? No, tomorrow, and by bike

(471) Niemand heeft voor Aruba gekozen, en *bijna iedereen voor Bonaire*
'Nobody has for Aruba chosen, and almost everyone for Bonaire'
Nobody choose Aruba, and nearly everyone Bonaire

In the study of grammar, numerous analyses have been proposed to ensure that the non-sentential proposition could be reconstructed. In Neijt (1980), gapping was even hailed as a specimen of core syntax. Thereafter, ellipsis was completely ignored by grammarians, or treatments were proposed with *ad hoc* mechanisms, culminating in a dedicated licensing construction in Merchant (2001). The interpretative approach of Dalrymple, Pereira and Shieber (1991) goes beyond syntax in order to stress interpretability. The problem with syntactical reconstruction is that the missing parts of the ellipsis are not available as a constituent – this was the basic observation that made Neijt (1980) move from reconstructing elided material to defining the relationship between remnants. The real problem is that normal syntax cannot provide constituency because anything can be a remnant, in particular parts that are not or even cannot be a target for reconstruction rules, like non-finite verbs.

(472) Kees is naar Amsterdam gefietst, en Jan gelopen.
'Kees is to Amsterdam biked-PART and Jan walked-PART.
Kees biked to Amsterdam and Jan walked.

In sentences like these, any effort to produce a structure that can provide the interpretation for the second sentence by general syntax is in vain. Such an

effort is bound to call upon dirty operations on non-constituents, as there are not enough moves available to save the day. Yet, language users do not have serious problems in interpreting ellipsis. And because explicit and implicit conjunction is a very generous source of ellipsis, language users do not have problems computing entailment. Every speaker of Dutch will recognize that (472) entails both propositions in

(473) Kees is naar Amsterdam gefietst.
(474) Jan is naar Amsterdam gelopen.

The second entailment clearly stems from the elliptical phrase. It is equally clear, however, that the entailed proposition is not provided by unificational syntax.

4.5.4 A conjecture on the interface

An open-minded look at the four types of examples of the separation of constituency and entailment and numerous other mismatches leads to the idea that this separation is neither incidental nor accidental. We might conjecture that, except for some full sentences, constituents do not project any entailment, and that entailments cannot be traced back to a non-sentential constituent. This conjecture is made explicit below.

(475) *Dissociation conjecture*
If A is a proper constituent of S and S entails P then ⟦A⟧ does not entail P and P is not equivalent to ⟦A⟧.

With a proper constituent we mean every constituent that is not a full sentence, implicitly or explicitly coordinated with other full sentences. The conjecture excludes any naive mapping of structure on meaning, of syntax on semantics and of unification on composition. It says that ultimately decent grammar does not enjoy compositionality.
According to (475), meaning does not come for free with quantifier movement and the like. Syntactical operations may get us somewhere, but not to the semantic finish. For deep interpretation, grammar has to get rid of structure. According to (475), the *syntax-semantics interface* calls for methods beyond merge, unification, feature checking, and the like. These methods may be type-0 in the Chomsky hierarchy, and not subject to elegant principles of locality, economy, type-shifting, binding or whatever. They may appeal to that part of cognition that is beyond strong artificial intelligence, to pick up on

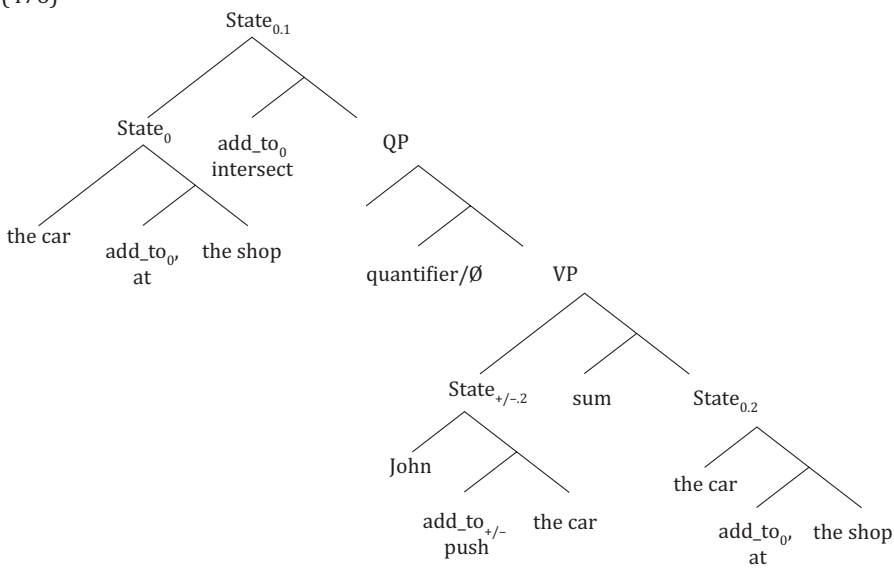
Roger Penrose's (1989) lead: they may be computable, but this computation does not necessarily reflect established principles of grammar. In that sense, (475) conjectures that syntax and semantics are not functionally related but, rather, they are complementary branches of linguistic analysis. To get to meaning, you have to go beyond unification. To compute entailments, forget about constituency.

4.6 THE LEXICON AS AN ORACLE: THE CASE OF *BEHALVE*

In the preceding sections, and even in all the preceding chapters, it was argued that syntax and semantics, unification and composition cannot carry the full load of deriving linguistic truisms. Again, we will invoke the lexicon to provide the bits and pieces of the final theorems – those truisms that explicitly connect texts by dubbing one text an entailment of the other. In addition, we need algorithms which live neither on unspecific unification nor on nervous, typed composition to assemble the bits and pieces offered by the lexicon as input into the final act of constructing meaning as the simultaneous presence of relevant entailments.

The idea that the lexicon stores specific linguistic knowledge is quite common among computational and other linguists. This view can even be called traditional: the lexicon/constructicon contains all information that cannot be derived by other means or is unpredictable. The idea that the lexicon feeds detailed semantic structure into the derivation can be traced back to Montague (1972), the protagonist of event semantics for verbs, and – for broad-minded lovers of history – to Russell's theory of definite descriptions. The essence of storing rich, very rich semantic structure in the lexicon is that it does not interfere with type-logical complexity – the semantic contribution of a phrase is type-driven but not type-structured. A verb, for example, can come with extensive aspectual and pre-suppositional structure, but it may also introduce a simple property. An achievement will bring with it more semantic material than a simple event verb (see, *e.g.* Arsenijevic 2006). Below you will find the semantic representation that Arsenijevic (2006) claims for sentences with perfective telic verbs, like *John has pushed the cart to the shop.* It involves at least three states, connected by relations between propositions.

(476)



Let us now focus on a few of those elements that, like no other, influence the inferential structure of a sentence: the functional heads. Montague (1972) awoke linguistics by assigning (477) to the simple determiner *the.*

(477) $\lambda_{\wp_{<ssett>}} \lambda\Re_{<ssett>} \forall x_{<se>} [\wp\{x\} \Rightarrow \Re\{x\}]$.

The determiner was stored as a function of type <<ssett, ssett>, t>, that is, a relationship between properties-in-intension, introducing a partial ordering – as an implication or subsumption of sets – in the semantic structure of any sentence which it was part of.

In the same vein, we might ask what a functional head like *behalve* 'except' must look like in order to ensure the entailment pattern for exclusive readings that was established in section 4.5.1. It is repeated here.

(478) Behalve Socrates waren alle filosofen dronken
Except for Socrates, all philosophers were drunk
entails
Socrates was een filosoof — *Socrates was a philosopher*
and
Niet alle filosofen waren dronken — *Not all philosophers were drunk*
and
Socrates was niet dronken — *Socrates was not drunk*

All of these entailments are induced by the exception phrase or, better, by its head *behalve*. None of the entailments, however, can be traced back to any single constituent. Therefore, we must rely on *behalve*'s lexical specification for providing the entailment pattern: if the constituent structure cannot identify entailments, the functions-to-be-composed must do so. The phrasal semantics is the only input to the identification of entailments when syntax and unification cannot deliver. The three entailments of exclusive *behalve* are tantamount to conjuncts at the high-level logical form of a sentence governed by *behalve*.
In order to specify the entailment pattern as a part of the lexical semantics, we have to determine some essential anchors in the sentential structure. First, it must be noted that the complement of *behalve* needs some co-typed counterpart in the matrix. This counterpart must be downward entailing with respect to its internal domain for *behalve* to get an exclusive reading. Moreover, the internal domain itself is needed for its contribution to a predicative statement about the complement of the exception phrase. Also, the negation of the main clause's quantificational structure must be constructable, and this again may depend on the semantics of the counterpart phrase. Secondly, temporal and aspectual characteristics of the main clause must be isolated and targeted in order to get the aspects of the entailments right. Thirdly, the semantics of *behalve* must guarantee the simultaneous validity of at least the three entailments; they are in conjunction with each other. Summarizing, the semantics of exclusive *behalve* in the constructicon for Dutch must informally be as follows.

(479) *Behalve* denotes that function ***f*** from a proposition P and an object B to propositions such that
if P is equivalent to R(S), R is of type of B and equivalent to Q(N) with Q downward entailing in N,
then ***f***(P, B) is the conjunction of
(i) the denial of P
(ii) the denial of the application of B to S
(iii) the assertion that B is N according to the tense of P

The definition assumes that in the main sentence the semantics of the counterpart to *behalve*'s complement is detectable. Its detection, however, will require some non-unificational algorithm, although the detection must be syntactically valid: the counterpart is supposed to be a proper constituent of the main clause to which the *behalve* phrase has been attached. The detection, therefore, requires some post-derivational inspection of the structure. Seen in this way, part of the semantics of *behalve* is a dedicated algorithm to determine the target constituent. In the formulation above, this algorithm

– too dirty to spell out here – is absorbed in the *if*-clause. It is as double-layered as Pythian talk.
More formally, the semantics of exclusive *behalve* can be stated this way:

(480) $f_{\text{behalve-exc}}(Q_{\beta\alpha}(N_{\beta})(S)_{\alpha\tau}, B_{\alpha}) = B(N) \,\&\, \textit{not}(Q(N)(S)) \,\&\, \textit{not}(B(S))$

When fully applied, the composition of this function with relevant meanings in its environment yields a conjunction of logical forms. One may hope that a generator can turn these logical forms into decent Dutch, or decent Swahili, for that matter. If so, the constructicon helps to overcome the predictable and built-in limits of good grammar.

4.7 THE INCOMPLETENESS OF GRAMMAR

Gödel's Incompleteness Theorem is both a fairy tale and rocket science. It repositions human intellectual analysis and understanding. It says that we are able to understand some very complex processes, without being able to fully compile them, to abstract from them completely. In his criticism of claims amounting to the perspective of creating extra-human supra-intelligence, Penrose (1989) over and over refers to Gödel's magnificent result, and with good reason: a system cannot prove its consistency, even if we know that it is consistent. And what is best: Gödel *proved* this to be the case – at least he convinced human intellect.

This chapter is not about turning grammar into mathematics. There is always that quotation from Daniel Kehlmann's *Die Vermessung der Welt* "Measuring the World" according to which Gauss tells Wilhelm von Humboldt, who claims to be a researcher too, that linguists long for mathematics but are not smart enough and deal with home-cooked poor logic. Still, grammar running on computers is a formal system of the type that is subject to Gödel's Incompleteness Theorem. It is, however, dealing with more complicated, multi-layered and more intensional objects than numbers. It is crucial to Gödel's Incompleteness that the class of theorems beyond proof of rejection is well-defined, and intrinsically restricted. Gödel actually proved that in arithmetic a two-place predicate *X is a proof of Y* introduces inconsistency. He proved that for a system to be consistent certain theorems are bound to be beyond computation. He proved that the set of arithmetically provable sentences

is a proper subset of the set of arithmetic truisms, and that the difference between these is well-defined.

For computational grammar, that is, for grammar that operates by computation unsupervised, certain theorems conflict with the internal mechanisms of grammar. Certain theorems are incompatible with doing syntax syntactically and semantics semantically. Our suggestion is that these theorems are exactly those that connect the yield of syntax to the yield of semantics. Let us write [S] for the syntactical analysis or the constituent structure of a sentence S, according to everything that syntax buys us. Let us assume that we have a decent calculus of entailments – we do not have such a calculus yet, but there is no reason why we could not find one – and let us simply call the relevant relation *entails*. Then we can state the incompleteness of grammar as the incompatibility of a statement about entailment between structured sentences with both unification and composition.

(481) *Incompleteness of grammar*
For [S] and [P] being consistent analyses of S and P respectively, and if *S entails P* means that their logical forms are related by $[\![S]\!] \prec [\![P]\!]$,
the proposition *[S] entails [P]* can neither be proven nor disproven by unification or composition.

The idea is that syntactical means – based on unification – are insufficient and inadequate to identify [P] as an entailment, and that semantic means – based on composition – are insufficient and inadequate to reconstruct [S] as an entailer of [P]. From a processing point of view, syntax is not strong enough to bring us to entailments, and composition is too invasive to bring us back to form. Together, they do not suffice to span the grammatical cycle mechanically. Since a homomorphism between unification and composition cannot be constructed, an attempt to prove theorems of the form *[S] entails [P]* is bound to fail. This failure identifies the class of theorems as filling the gap between grammatical statements that can be deduced and linguistic truisms that cannot. S may entail P, but *[S] entails [P]* cannot be proven by grammatical means. It is essential to acknowledge that incompleteness (481) is not a defect of grammar, but rather an asset. It qualifies the art of interpreting natural language in one of those domains that can be formalized without expanding to a totalitarian, holistic philosophy of everything. Though language is everything, not everything is language, Van Benthem claimed, and he is right. Even the grammar of natural language cannot justify itself. It is not the last word on how it is. The grammarian and her constructicon will always be needed for the final touch.

4.8 THE FRUIT OF INCOMPLETENESS

Several strategies are known for repairing the non-monotony of syntax and semantics and, thus, for avoiding incompleteness. Here are four ways to free your favourite grammar from incompleteness.

(482) (1) stick with underspecified semantics
(2) abstract from logical form
(3) abstract from constituent structure
(4) abstract from grammar

The first strategy is practised in HPSG and Head-Driven Construction Grammar (Boas and Sag 2012) and part of a well-defined computational model. The second strategy is shared by Generative Grammarians and Construction Grammarians; it is not confessed, but just practiced. Montagovianists and Lambek-style Categorial Grammar coined the third strategy structural completeness. The fourth strategy is common in Bayesian processing, as a survival strategy for robust parsing.

One way of implementing the third strategy is enriching syntactical derivation, up to mimicking lambda conversion in the syntax. This was practiced by Montague (1972), and – on a more principled basis – by many categorial grammarians after him (Moortgat 1988, Carpenter 1997, Morrill 1994 and 2011, Hendriks 1993). Under this strategy, syntactical derivation is made responsible for scoping effects and other forms of semantic dependency. It leads to the blurring of syntactical structure and derivation, based on the structural completeness of certain categorial calculi: if a sentence can be proven to be interpretable, it is so under any bracketing or grouping of constituents. By structural completeness, many derivational options are created that have no effect at all on constituency or linear order. The result is spurious ambiguity, which has to be fought afterwards (Dowty 1988). Yet, the distribution of cases and roles in *Every man loves some women* is not affected by which women are loved by which man in a model for that sentence. The strategy to flexibilize the derivation makes syntactical structure obsolete. It relinquishes the beautiful and intriguing linguistic generalization that syntactical structure is essentially independent of denotation – the autonomy of syntax, a major fact about natural language as a cognitive system. P and notP do not differ as fundamentally in syntax as they do in semantics.

The other real strategy to deal with the antimorphic tension between structure and interpretation is to be satisfied with underspecified semantics: that level of semantic specificity that can be reached by pure unification. This is the dominant attitude among deep processing computationalists (Copestake *et al.* 2005, Bunt 2008, Egg 2010), and maybe among logical-form theoreticians. It refrains from invading the syntax, and it keeps the grammar as a clean and computable as possible. Like Egg (2010), Boas and Sag (2012:95) even sees advantages to it:

> *"Scope underspecification is the heart and soul of M(inimal) R(ecursion) S(emantics), which was originally developed for computational purposes. Scope resolution ... is a difficult and currently unsolved research problem. Hence, having semantic representations that do not require full scope resolution is a significant advantage in much computational work, e.g. machine translation. However, ..., it may be useful in psycholinguistics, as well."*

Underspecified semantics avoids incompleteness, but does not identify linguistic meaning as an independent object in its own right, living in a realm in which syntax is no longer needed. It adheres to a representation that is meaningful only in relation to a syntax-semantics interface. Underspecified semantics does not present a logic, with computable operations on meaning. Underspecification implies raising the question whether meaning exists.

In generative grammar and construction grammar, the nature of logical *form* is not a big thing. Croft (2001:212), for example, presents the semantic scheme for a Tzotzil sentence as a graphic map of the annotated sentence on some predication-like sequence, without accounting for the details of the mapping. Construction grammarians tend to represent aspects of meaning in graphics, giving little attention to compositional or formal matters. With the possible exceptions of highly theoretical approaches like those of Kracht (2004) and Stabler (1992), generative grammarians only rarely pay tribute to the precise derivation of generalized logical form. It is unlikely, however, that generative grammarians would consider logical form trivial. Broekhuis and Dekkers (2000) suggest that in a grammar model with split derivation for phonological and logical form, the former but not the latter may be subject to optimalization, thereby maintaining the 'classic' set of parameters and principles for logical form. In neither grammatical tradition are the semantic representations chosen (pictures, graphs, formulas) explicitly linked to constituent structure. Logical form is mostly anecdotic, derived stepwise. Of course, incompleteness is avoided this way, but logical form as a product of grammar

is not achieved. Again, independent identification of a semantic object is not sought after. Quite the contrary, grammarians of generative and constructional persuasions give the impression that they do not consider linguistic meaning as an independent object.

In Bayesian approaches to natural language processing, it is becoming more and more accepted to claim that grammar as such can be circumvented (*e.g.* Frank, Bod and Christiansen 2012). In this realm, grammar is at best a necessary evil, and particular grammatical specifications as such are hardly a topic of debate. The journals of computational linguistics are mostly about processing on the basis of some given annotation, and not about which grammatical annotation best serves computational goals. As a consequence, the internal organization of the grammar is seen as external to processing. Incompleteness in the sense of (481) stays out of sight.

It pays to see the struggle for computing towards and from meaning as the price for incompleteness. Paying this price frees syntax from semantic overload, and frees semantics from contingent syntactical bookkeeping. It prevents us from computing distinct syntactical derivations to accommodate purely semantic diversity. It makes semantics transparent, as it does not have to reflect syntactical accidents. Syntax brings meaning to a certain point – Stored Logical Form, as we called it in chapter 2. After that point, syntax is only *exploited* by meta-algorithms to compute fully specified meaning, but syntax is no longer in charge. In this sense, the overall computational properties of grammar are beyond the properties of syntax. The meaning of a sentence is Cognition with a capital. The fact that human beings are able to act as if sentences are meaningful marks an astonishing convergence of biological and social evolution. Yet, the fact that we dynamically assign meanings to sequences of symbols does not mean that this competence is modeled by *computable* mappings (Lasersohn 2009). We had better acknowledge that one single formal system couldn't operate on all levels of explanation of our linguistic minds. This is an instance of the paradox that Gödel discovered: systems of knowledge are formal and consistent only if they are not holistic and totalitarian. If artificial intelligence is bound to stay within the limits of mathematics, grammar certainly is too. Grammar, even computational grammar, can compute a lot, but not everything that is true of grammatical objects. The study of language is not witchcraft but normal business, after all.

REFERENCES

Ades, A. and M. Steedman. On the Order of words. *Linguistics and Philosophy 4.* 1982. pp 517-558

Aho, A.V. Indexed grammars: An extension of the context-free grammars. *Journal of the ACM* **15**. 1968. pp 641-671

Ajdukiewicz, K. Die syntaktische Konnexität. *Studia Philosophica* **1**. 1935. pp 1-27

Alshawi, H. (ed.) The Core Language Engine. The MIT Press. 1992

Alshawi, H., D. Carter, M. Rayner, and B. Gambäck. Translation by Quasi Logical Form Transfer, *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*. ACL. 1991. pp 161–168

Ambler, S.W. Mapping objects to relational databases, *IBM DeveloperWorks Magazine*. 2000

Aronoff, M. In the beginning was the word. *Language 83:4*. 2008. pp 803-830

Arsenijević, B. *Inner aspect and telecity*. LOT. 2006

Asher, N. *Reference to Abstract objects in Discourse.* Kluwer. 1993

Atkinson, M., F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik (1989). The Object-Oriented Database System Manifesto, *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*. Kyoto. Elsevier Science Publishers. 1990. pp 223-240

Bach, E. Problominalization*, Linguistic Inquiry* 1. 1970. p 121

Bach, E. The algebra of events. *Linguistics and Philosophy 9*. 1986. pp 5-16

Baldridge, J. and G.-J.M. Kruijff. Multi-Modal Combinatory Categorial Grammar. *Proceedings of 10th Annual Meeting of the European Association for Computational Linguistics.* Budapest. 2003. pp 211-218

Bar-Hillel, Y. A Quasi-Arithmetical Notation for Syntactic Description. *Language 29*. 1953. pp 47-58

Bar-Hillel, Y., C. Gaifman, and E. Shamir. On Categorial and Phrase Structure Grammars. *Bulletin of the Research Council of Israel*, *9F*. 1960. pp 1-16

Barwise, J. and R. Cooper. Generalized Quantifiers and Natural Language. *Linguistics and Philosophy 4:1*. 1981. pp 159-219

Bennis, H. *Gaps and Dummies*. Foris. 1986

Berners-Lee, T.B., J. Hendler and O. Lasilla. The Semantic Web. *Scientific American*. 2001

Bloom, P. *How children learn the meaning of words.* CUP. 2002

Boas, H. C. and I.A. Sag. *Sign Based Construction grammar.* CSLI. 2012

Bobrow, D., C. Condoravdi, R. Crouch, V. de Paiva, L. Karttunen, T. H. King, R. Nairn, L. Price, and A. Zaenen. Precision focused Textual inference. *Proceedings of the workshop on textual entailment and Paraphrasing*. ACL. 2007. pp 16-21

Bos, J. DORIS 2001: Underspecification, Resolution and Inference for Discourse Representation Structures. *In:* Blackburn, P. and M. Kohlhase (eds). *ICoS-3. Inference in Computational Semantics*. Buxton. 2001. pp 117– 124

Bos, J. *Underspecification and resolution in Discourse Semantics.* Universität des Saarlandes. 2002

Bos, J. Computational Semantics in Discourse: Underspecification, Resolution, and Inference. *Journal of Logic, Language and Information 13:2*. 2004. pp 139–157

Bos, J. and K. Merkert. When logical inference helps determining textual entailment (and when it doesn't). *Proceedings 2nd Pascal RTE Challenge Workshop*. 2006

Bouma, G. *Nonmonotonicity and Categorial Unification Grammar*. Rijksuniversiteit Groningen. 1993

Bouma, G. and G. van Noord. Adjuncts and the processing of lexical rules. *Proceedings 15th Conference on Computational Linguistics.* 1994

Bouma, G. and G. van Noord. Word Order Constraints on Verb Clusters in German and Dutch. 1996 *http://odur.let.rug.nl/~vannoord/papers/complex.pdf*.

Bouma, G., G. van Noord, and R. Malouf. Alpino: Wide-coverage Computational Analysis of Dutch. *Computational Linguistics in the Netherlands 2000*. Rodopi. Amsterdam. 2001

Brandt Corstius, H. *Computer-taalkunde*. Coutinho. 1978

Broekhuis, H. and J. Dekkers. The minimalist program and optimality theory. Derivations and Evaluations. *In:* Dekkers, J. *et al.* (eds). *Optimality theory: Phonology, Syntax and Acquisition*. OUP. 2000. pp 386-422

Bunt, H. Semantic underspecification: which technique for which purpose? *In:* Bunt, H. and R. Muskens (eds). *Computing meaning, Volume 3.* Springer. 2008. pp 55-85

Buszkowski, W. Compatibility of a categorial grammar with an associated category system. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, *28*. 1982. pp 229-237

Carpenter, B. *The logic of type feature structures*. CUP. 1992

Carpenter, B. *Type-Logical Semantics*. MIT Press. 1997

Carroll, J., A. Copestake, D. Flickinger, and V. Poznànski. An Efficient Chart Generator for (semi-)Lexicalist Grammars. *In: Proceedings of the 7th European Workshop on Natural Language Generation (EWNLG'99)*, 1999. pp 86-95

Carrouges, M. *Les machines célibataires.* Alfieri. 1975

Ceri, S., G. Gottlob, and G. Wiederhold. Efficient Database Access from Prolog, *IEEE Transactions on Software Engineering* **15**(2), 1989. pp 153-164

Chaffin, R. and D.J. Herrmann. The nature of semantic relations: a comparison of two approaches. *In:* Evans, M.W. (ed). *Relational models of the lexicon.* CUP. 1988. pp 289-335

Chamberlin, D.D. and R.F. Boyce. SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*. Association for Computing Machinery. 1974. pp 249-264

Chambers, N., D. Cre, T. Grenager, D. Hall, C. Kiddon, B. MacCartney, M. de Marneffe, D. Ramage, E. Yeh, and C. Manning. Learning Alignments and Leveraging Natural Logic. *Proceedings of the workshop on textual entailment and Paraphrasing*. ACL. 2007. pp 165-170

Chierchia, G. and S. McConnell-Ginet. *Meaning and Grammar*. MIT Press. 2000

Chomsky, N. Remarks on Nominalization. *In:* Jacobs, R. and P. Rosenbaum (eds). *Readings in English Transformational Grammar.* Ginn. 1970. pp 184-221

Chomsky, N. *Lectures on Government and Binding*. Foris. 1981

Chomsky, N. *The Minimalist program*. MIT Press. 1995

Clocksin, W.F. and C.S. Mellish. *Programming in Prolog*, 2nd ed. Springer-Verlag, Berlin. 1984

Clocksin, W.F. and C.S. Mellish. *Programming in Prolog Using the ISO Standard*. Springer. 2003

Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM 13(6)*. 1970. pp 377–387

Collins, C. and E. Stabler. *A formalization of minimalist syntax.* Ms. 2011. *http://ling.auf.net*

Cooper, R. *Quantification and Syntactic Theory*. Reidel. 1983

Cooper, R., D. Crouch, J. van Eijck, C. Fox, J. van Genabith, J. Jaspers, H. Kamp, D. Milward, M. Pinkal, M. Poesio, and S. Pulman. *Using the Framework*. LRE 62-051. 1996

Copestake, A. and D. Flickinger. An Open Source Grammar Development Environment and Broad-coverage English Grammar Using HPSG. *Proceedings of the 2nd International Conference on Language Resources and Evaluation.* 2000

Copestake, A., D. Flickinger, C. Pollard, and I.A. Sag. Minimal Recursion Semantics: An Introduction. *Research on Language and Computation 3:4.* 2005. pp 281-332

Cormack, A. and N. Smith. Where is a sign merged? *GLOT International 4:6.* 1999. p. 20

Cremers, C. On two types of infinitival complementation in Dutch. *In:* Heny, F. and B. Richards (eds). *Linguistic Categories: Auxiliaries and related Puzzles. Volume 1*. Reidel. 1983. pp 169-221

Cremers, C. Over een lineaire kategoriale ontleder. *TABU 19: 2*. 1989. pp 76-85

Cremers, C. *On parsing coordination categorially*. PhD thesis. Leiden University. 1993a

Cremers, C. Coordination as a Parsing Problem. *In:* Sikkel, K. and A. Nijholt (eds.). *Proceedings of the 6th Twente Workshop on Language Technology* (TWLT6, ACL/SIGPARSE). 1993b. pp 139-143

Cremers, C. A Note on Categorial Grammar, Disharmony and Permutation. *In: Proceedings EACL 1999*. ACL. 1999. pp 273-274

Cremers, C. ('n) Betekenis berekend. *Nederlandse Taalkunde 7:4*. 2002. pp 375-395

Cremers, C. Why plurality does not mean a thing. *In*: Bogaards, P., J. E. C. V. Rooryck and P. J. Smith (eds.). *Quitte ou Double Sens. Articles sur l'ambiguïté offerts à Ronald Landheer*. Rodopi. 2002

Cremers, C. Modal Merge and Minimal Move for Dislocation and Verb Clustering. *Research on Language and Computation 2:1*. 2004. pp 87-103

Cremers, C. NL from Logic: Connecting Entailment and Generation. *In:* Aloni, M., H. Bastiaanse, T. de Jager, and K. Schulz (Eds). *Logic, Language and Meaning*. Springer. 2010. pp 94-103

Cremers, C. and M. Hijzelendoorn, *Counting Coordination Categorially*, Computation and Language archive, 1996 *http://arxiv.org/abs/cmp-lg/9605011*

Cremers, C. and M. Hijzelendoorn. Pruning Search Space for Parsing Free Coordination in Categorial Grammar. *Proceedings 5th International Workshop on Parsing Technologies* (IWPT97, ACL/SIGPARSE). 1997a. pp 42-53

Cremers, C. and M. Hijzelendoorn. Filtering Left Dislocation Chains in Parsing Categorial Grammar. *In:* Landsbergen, J., J. Odijk, K. van Deemter, and G. Veldhuijzen van Zanten (eds.). *Papers from the 7th CLIN Meeting*. 1997b. pp 41-56

Cremers, C. and H. Reckman. Exploiting logical forms. *In:* Verberne, S., H. van Halteren, and P.A. Coppen (eds). *Computational Linguistics in the Netherlands 2007*. LOT, 2008. pp 5-21

Croft, W. *Radical Construction Grammar*. OUP. 2001

Crouch, R. and J. van Genabith. Context Change, Underspecification and the Structure of Glue Language Derivations. *In:* Dalrymple, M. (ed.). *Semantics and Syntax in lexical Functional Grammar: the Resource Logic Approach.* MIT Press. 1999. pp 117-190

Daelemans, W. and A. van den Bosch. *Memory-Based Language Processing*. CUP. 2005

Dalrymple, M., F. Pereira, and S. Shieber. Ellipsis and higher-order unification. *Linguistics and Philosophy 14.* 1991. pp 399-453

Dalrymple, M., A. Kehler, J. Lamping, and V. Saraswat. The semantics of resource sharing in lexical-functional grammar. *Proceedings 7th EACL*. 1995. pp 31-38

Date, C.J. *Introduction to Database Systems*. 8th ed. Addison-Wesley. 2003

Davidson. D. The logical form of action sentences. *In:* Rescher, N. (ed.). *The logic of decision and action.* University of Pittsburgh Press. 1967

De Hoop, H. *Case Configurations and NP Interpretation*. Rijksuniversiteit Groningen. 1992

De Vries, M.H., M.H. Christiansen, and K.M. Petterson. Learning Recursion: Multiple Nested and Crossed Dependencies. *Biolinguistics 5:1-2.* 2011. pp 10-35 *http://cnl.psych.cornell.edu/pubs/2011-dcp-bioling.pdf.*

Dekker, P. *Transsentential meditations; ups and downs in dynamic semantics*. Universiteit van Amsterdam. 1993

Den Besten, H., J. Rutten, T. Veenstra, and J. Veld. *Verb Raising, Extrapositie en de Derde Constructie*. Unpublished. University of Amsterdam. 1988

Deransart, P., A. Ed-Dbali, and L. Cervoni. *Prolog: The Standard,* Springer-Verlag. 1996

Diesing, M. and E. Jelinek. Distributing arguments. *Natural Language Semantics 3:2.* 1995. pp 123-176

Dowty, D. Type raising, functional composition, and non-constituent coordination. *In:* Oehrle, R. *et al* (eds). *Categorial grammars and natural language structures*. Kluwer. 1988. pp 153-198

Ebeling, C.L. *Syntax and Semantics*. Brill. 1978

Ebert, C. *Formal Investigations of Underspecified Representations*. University of London. 2005

Egg, M. Semantic Underspecification*. Language and Linguistics Compass 4:3*. 2010. pp 166-181

Eisner, J. Efficient Normal-Form parsing for Combinatory Categorial Grammar. *In: Proceedings of the 34th Annual Meeting of the ACL*. 1996. pp 79-86

Everaert, M. and H. van Riemsdijk. *The Blackwell Companion to Syntax*. Blackwell. 2006

Evers, A. *The Transformational Cycle in Dutch and German*. Universiteit Utrecht. 1976

Fannee, M. *Over het implementeren van het coördinatie-algoritme in de Chart-parser van Delilah*. MSc. thesis, Dept. of Linguistics, Leiden University. 2006

Flynn, M. A Categorial Theory of Structure Building. *In:* Gazdar, G., E. Klein and G.K. Pullum. *Order, Concord and Constituency*. Foris. 1983. pp 139-174

Flickinger, D., A. Copestake, and I. Sag. HPSG Analysis of English. *In:* A. Wahlster (ed). *Verbmobil. Foundations of Speech-to-Speech technology*. Springer. 2000. pp 254-263

Fodor, J.D. and I. Sag. Referential and Quantificational Indefinites. *Linguistics and Philosophy 5:3*. 1982. pp 355-398

Frank, A. and U. Reyle. Principle-based semantics for HPSG. *Arbeitspapiere des Sonderforschungsbereich 340*. Universität Stuttgart. 1994

Frank, S.L., R. Bod, and M.H. Christansen. How hierarchical is language use? *Proceedings of the Royal Society B*. doi:10.1098/rspb2012.1741

Friedman, J. and T. H. Bredt. *Lexical Insertion in transformational Grammar*. Technical Report, Stanford University. 1968

Gärdenfors, P. *Conceptual Spaces: The geometry of Thought*. MIT Press. 2000

Gardner, M. *Logic Machines and Diagrams*. McGraw-Hill. 1958

Gazdar, G. *Applicability of indexed grammars to natural languages*. CSLI. 1985

Gazdar, G., E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Blackwell. 1985

Goldberg, A. E. *Constructicons. A constructicon grammar approach to argument structure.* Uni. Chicago Press. 1995

Goddard, C. The search for the shared semantic core of all languages. *In:* Goddard, C. and A. Wierzbicka (eds). *Meaning and Universal grammar – Theory and Empirical Findings. Volume 1*. John Benjamins. 2002. pp 5-40

Grégoire, N. DuELME: a Dutch electronic lexicon of multiple word expressions. *Language Resources and Evaluation 44:1-2.* 2010. pp 23-39

Grootveld, M. *Parsing coordination generatively*. Holland Institute of Linguistics. 1994

Hankamer, J. and I. Sag. Deep and Surface Anaphora. *Linguistic Inquiry 7*. 1976. pp 391-428

Harbusch, K. and G. Kempen. Elleipo: A module that computes coordinative ellipsis for language generators that don't. *EACL-2006. Conference Companion.* EACL. pp 115-118

Harrison, M. A. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, 1978

Hauser, M.D. and W.T. Fitch. What are the uniquely human components of the language faculty? *In*: Christiansen, M.H. and S. Kirby (eds.). *Language Evolution*. OUP. 2003. pp 158-181

Heim, I. and A. Kratzer. *Semantics in Generative Grammar*. Blackwell. 1998

Heine, B. and T. Kuteva. *The Genesis of Grammar*. *A reconstruction*. OUP. 2007

Hendriks, H. *Studied Flexibility*. University of Amsterdam. 1993

Hendriks, P. *Comparatives and categorial grammar*. Rijksuniversiteit Groningen. 1995

Hepple, M. *The grammar and processing of order and dependency*. Edinburg. 1990

Hepple, M. A Compilation-Chart Method for Linear Categorial Deduction. *Proceedings Coling 1996*. Center for Sprogteknologi, Copenhagen. 1996

Hepple, M. and G. Morrill. Parsing and derivational equivalence. *In: Proceedings of the 4th Conference of the EACL*. 1989. pp 10-18

Higginbotham, J. On Semantics. *Linguistic Inquiry 16:4*. 1985. pp 547–593

Higginbotham, J. Linguistic theory and Davidson's program in semantics. *In:* Le Pore. E. (ed.). *Truth and Interpretation*. Blackwell. 1986. pp 29-48

Hirao, T. (1990). Extension of the relational database semantic processing model. *IBM Systems Journal 29(4)*. pp 539-550

Hoeksema, J. Negative polarity and the comparative. *Natural Language and Linguistic Theory 1*. 1983. pp 403-434

Hoeksema, J. *Categorial Morphology*. Rijksuniversiteit Groningen. 1984

Hoeksema, J. The Semantics of Non Boolean "and". *Journal of Semantics 6:1*. 1988. pp 19-40

Hoeksema, J. and R. Janda. Implications of Process Morphology for Categorial Grammar. *In:* Oerhle, D., E. Bach, and D. Wheeler (eds), *Categorial Grammar and natural Language*. Kluwer. 1988. pp 199-248

Hollebrandse, B. and T. Roeper. *Recursion and propositional exclusivity*. Unpublished. 2007

Honcoop, M. *Dynamic Excursions on Weak Islands*. Universiteit Leiden. 1998

Hopcroft, J.E., R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languges and Compuation*. Addison Wesley. 2001

Houtman, J. *Coordination and Constituency*. Rijksuniversiteit Groningen. 1994

Huffman, D.A. *A Method for The Construction of Minimum Redundancy Codes*, *Proceedings of the Institute of Radio Engineers 40(9*). 1952. pp 1098-1101

Huijbregts, M.A.C. *Overlapping Dependencies in Dutch*. Utrecht Formal Papers in Linguistics. 1976

Icke, V. *The Force of Asymmetry*. CUP. 1995

Jackendoff, R. *Foundations of Language*. OUP. 2002

Jacobson, P. Comment 'Flexible Categorial Grammars.' *In*: Levine, R. (ed.). *Formal grammar: theory and implementation*. OUP. 1991. pp 129 - 167

Jäger, G. On the Generative Capacity of Multi-modal Categorial Grammars. *Research on Language and Computation 1*. 2003. pp 105-125

Jäger, G. *Anaphora and Type-logical Grammar*. Springer. 2005

Janssen, T.M.V. *Foundations and applications of Montague Grammar*. CWI. 1986

Janssen, T.M.V. Compositionality. *In:* Van Benthem, J. and A. ter Meulen (eds). Handbook of Logic and Language. North Holland. 1997. pp 417-474

Jaspers, D. *Operators in the Lexicon.* Leiden University. 2005

Jech, T.J. About the Axiom of Choice. *In:* Barwise, J. (ed). *Handbook of Mathematical Logic*. North Holland Pub Cy. 1977. pp 345-370

Johannesen, J. B. *Coordination*, OUP. 1997

Joshi, A.K. How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars. *In:* Dowty, D., L. Karttunen, and A. Zwicky (eds.), *Natural language parsing: Psychological, computational and theoretical perspectives*. CUP. 1985. pp 206-250

Joshi, A.K., K. Vijay-Shanker, and D. Weir. The Convergence of Mildly Context-Sensitive Grammar Formalisms. *In*: P. Sells, S.M. Shieber, and T. Wasow (eds.), *Foundational Issues in Natural Language Processing*. MIT Press, 1991. pp 31 - 82

Kamp, H. and U. Reyle. *From Discourse to Logic*. Springer. 1993

Karttunen, L. *Radical lexicalism*. CSLI. 1986

Kasami, T. An efficient recognition and syntax algorithm for context-free languages. *Air Force Cambridge Research Laboratory*. 1965

Kasper, R. and W. Rounds. The logic of unification grammar. *Linguistics and philosophy 13*. 1990. pp 33-58

Kay, M. The MIND system. *In:* Rustin, R. (ed). *Natural Language Processing*. Algorithmics Press. 1973. pp 155-188

Kay, Martin. Algorithm schemata and data structures in syntactic processing. *In:* Grosz, Barbara J., Karen Sparck Jones, and Bonnie Lynn Weber (eds.), *Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, California, 1980, pp 35-70

Kay, M. *Unification Grammars*. Xerox. 1981

Kay, P. and C. Fillmore. Grammatical constructions and linguistic generalizations: The what's X doing Y? construction. *Language 75*. 1999. pp 1–33

Kayne, R.S. *The Antisymmetry of Syntax.* MIT Press. 1994

Kempen, G. Clausal coordination and coordinative ellipsis in a model of the speaker. *Linguistics.* 2008

Kešelj, V. and N. Cercone. *A Graph Unification Machine for NL Parsing.* Univ of Waterloo. 2002

Khalaily, S. *One Syntax for All Categories – Merging Nominal Atoms in Multiple Adjunction Categories*. Holland Institute of Linguistics. 1997

Kneale, W. and M. Kneale. *The Development of Logic*. OUP. 1962

Knuth, D. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading MA. 1973

Koller, A. and S. Thater. An improved redundancy elimination algorithm for underspecified representations. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*. 2006. pp 409–416

Komagata, N. Efficient Parsing for CCGs with Generalized Type-Raised Categories. *Proceedings 5th International Workshop on Parsing Technologies (IWPT97, ACL/SIGPARSE)*. 1997. pp 135-146

Komagata, Nobo N., Nobo N. Komagata, I.S. Kulick, and R. Prasa. *A Computational Analysis of Information Structure using Parallel Expository Texts in English and Japanese*. Univ. of Pennsylvania. 1999

König, E. *Der Lambek-Kalkül. Eine Logik für lexikalische Grammatiken*. Universität Stuttgart. 1990

Kracht, M. *The Mathematics of Language*. Mouton de Gruyter. 2004

Krifka, M. Four Thousand Ships Passed through the Lock: Object-Induced Measure Functions on Events. *Linguistics and Philosophy* 13. 1990. pp 487-520

Kripke, S. Naming and Necessity. *In:* Davidson, D. and G. Harman (eds). *Semantics of natural language.* Reidel. 1972

Lakoff, G. and M. Johnson. *Metaphors we live by*. Chicago Univ Press. 1980

Lambek, J. The mathematics of sentence structure. *American Mathematical Monthly 64:3*. 1958. pp 154-170

Lasersohn, P. Compositional Interpretation. *In:* Hinrichs, E. and J. Nerbonne (eds). *Theory and Evidence in Semantics*. CSLI. 2009. pp 133-158

Levelt, W. *Speaking*: *from intonation to articulation*. The MIT Press. 1989

Levinson, S.C. *Presumptive Meanings.* The MIT Press. 2000

Link, G. The Logical Analysis of Plural and Mass Terms: A Lattice Theoretical Approach. *In:* Bäuerle, R., C. Schwarze and A. von Stechow (eds). *Meaning, Use and the Interpretation of language.* Walter de Gruyter. 1983. pp 302-323

Marcus, G. *The Algebraic Mind*. MIT Press. 2003

Marcus, M. *A Theory of Syntactic Recognition for Natural Language*. MIT Press. 1980

Maxwell, J. T. and R. M. Kaplan. The interface between phrasal and functional constraints. *Computational Linguistics 19:4*. 1993. pp 571–590

McCawley, J.D. Concerning the base component of a transformational grammar. *Foundations of language 4*. 1968. pp 243-269

McHale, B. Poetry as Prothesis. *Poetics Today 21:1*. 2000. pp 1-32

Mel'čuk, I. and A. Zholkovsky. *Explanotory Combinatorial Dictionary of Modern Russian*. Wiener Slawistischer Almanach. 1984

Merchant, J. *The syntax of silence: Sluicing, Islands and the Theory of Ellipsis*. OUP. 2001

Meyer, B. *Object-Oriented Software Construction*. Prentice Hall. 1997

Mitkov, R., S. Lappin, and B. Boguraev. Introduction to the Special issue on Computational Anaphora resolution. *Computational Linguistics 27:4*. 2001. pp 473-477

Montague, R. The Proper Treatment of Quantification in Ordinary English. *In:* Davidson, D. and G. Harman (eds). *Semantics of Natural Languages*. Reidel. 1972

Moortgat, M. *Categorial Investigations*. Foris. 1988

Moortgat, M. Categorial Type Logics. *In:* Van Benthem, J. and A. ter Meulen (eds). Handbook of Logic and Language, pp 93-177. Elsevier and The MIT Press. 1997

Morley, M. Homogeneous sets. *In:* Barwise, J. (ed.). *Handbook of Mathematical Logic*. North Holland Pub Cy. 1977. pp 181-196

Morrill, G. *Type Logical Grammar. Categorial Logic of Signs*. Kluwer. 1994

Morrill, G. *Categorial grammar: logical syntax, semantics and processing*. OUP. 2011

Munn, A. Topics in the syntax and semantics of coordinate structure. University of Maryland. 1993

Neijt, A. *Gapping*. *A contribution to sentence grammar*. Foris. 1980

Parsons, T. Underlying states and time travel. *In:* Higginbotham, J. *et al.* (eds). *Speaking of Events*. OUP. 2000. pp 81-93

Partee, B. H. Syntactic categories and semantic type. *In:* Rosner, M. and R. Johnson (eds). *Computational linguistics and formal semantics*. CUP. 1992. pp 97-126

Partee, B. and M. Rooth. Generalized Conjunction and Type Ambiguity. *In:* Bäuerle, R., C. Schwarze and A. von Stechow. *Meaning, Use, and the Interpretation of Language.* De Gruyter. 1983. pp 361-383

Payne, J. Complex phrases and complex sentences. *In:* Shopen, T. (ed.). *Language typology and syntactic description: complex constructions. Volume 2*. CUP. 1985

Penrose, R. *The Emperor's New Mind*. OUP. 1989

Pentus, M. Lambek Grammars are Context Free, *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*. 1993. pp 429-433

Pereira, F.C.N. and D.H.D. Warren. Parsing as Deduction. *Proceedings 21st Annual Meeting of the ACL*. MIT, Cambridge. MA. 1983

Pereira, F.C.N and S.M. Shieber. *Prolog and Natural language Analysis.* CSLI. 1987

Pietroski, P. *Events and Semantic Architecture.* OUP. 2006

Pietroski, P. Minimal semantic instructions. *In:* C. Boeckx (ed). *The Oxford handbook of linguistic minimalism*. 2011. pp 472-498

Poeder, J. *Over de verwerking van extra-grammaticale invoer door een natuurlijke-taalverwerkingssysteem*. MSc. Thesis, Leiden University. 1994

Pollard, C. *Generalized Phrase Structure Grammar, Head Grammars and Natural Languages*. Stanford University. 1984

Pollard, C. and I.A. Sag. *Information-Based Syntax and Semantics*. *Volume 1: Fundamentals*. CSLI. 1988

Pollard, C. and I.A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press. 1994

Poß, M. *Under Construction*. LOT. 2010

Poß, M. and T. van der Wouden. Extended Lexical Units in Dutch. *In:* Van der Wouden, T., M. Poß, H. Reckman, and C. Cremers (eds). *Computational Linguistics in the Netherlands 2004*. LOT. 2005

Postma, G.J. *Zero Semantics.* HIL. 1995

Pullum, G.K. and G. Gazdar. Natural languages and context-free languages*. Linguistics and Philosophy 4:4*. 1982. pp 471-504.

Pustejovski, J. *The Generative Lexicon: A Theory of Computational Lexical Semantics*. MIT Press. 1993

Ramchand, G. *First phase syntax*. Ms. Oxford University. 2002

Ramsay, A. *Extended Graph Unification*. 1989

Reckman, H. *Flat but not shallow*. PhD. Leiden University. 2009

Reckman, H. and C. Cremers. Concepts across categories. *Proceedings ICoS-5.* pp 97-106. 2006 *http://www.aclweb.org/anthology-new/W/W06/W06-3910.pdf*

Reinhart, T. Quantifier scope: how labor is divided between QR and choice functions. *Linguistics and Philosophy 20*. 1997. pp 335-397

Reiter, E. and R. Dale. Building Applied Natural Language Generation Systems. *Natural Language Engineering 3*. 1997. pp 57-87

Retoré, C. and E. Stabler. Generative grammars in resource logic. *Research on Language & Computation 2:1*. 2004. pp 3-25

Roorda, D. *Resource-logics: proof-theoretical investigations*. Univ. of Amsterdam. 1991

Rosetta, M.T. *Compositional Translation*. Kluwer. 1994

Ross, J.R. *Constraints on Variables in Syntax.* MIT. 1967

Rounds, W.C. Feature Logics. *In*: Van Benthem, J. and A. ter Meulen (eds). *Handbook of Logic and Language*. Elsevier and MIT Press. 1997. pp 475-535

Rozwadowska, B. Derived Nominals. *In:* Everaert, M. and H. van Riemsdijk (eds). *The Blackwell Companion to Syntax. Volume II.* Blackwell. 2006. pp 24-55

Russell, B. *The Philosophy of Logical Atomism*. University of Minnesota, Department of Philosophy. 1949 (Reprinted as: *Russell's Logical Atomism*. Fontana/Collins. 1972)

Ruys, E.G. Unexpected Wide-Scope Phenomena. *In:* Everaert, M. and H. van Riemsdijk (eds). *The Blackwell Companion to Syntax. Volume V*. Blackwell. 2006. pp 175-228

Sag, I.A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger. Multiword expressions: A pain in the Neck for NLP. *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics*. 2002. pp 1–15

Sag, I.A., T. Wasow, and E.M. Bender. *Syntactic Theory. A Formal Introduction.* CSLI. 2003

Scha, R. Distributive, Collective and Cumulative Quantification. *In:* Groenendijk, J., T.M.V. Janssen and M. Stokhof (eds). *Formal Methods in the Study of Language*. Univ of Amsterdam. 1981. pp 483-512

Schein, B. *Plurals and Events*. MIT Press. 1993

Seuren, P.A.M. *Western Linguistics. An Historical Introduction*. Blackwell. 1998

Seuren, P.A.M. The natural logic of language and cognition. *Pragmatics 16:1.* 2006. pp 103-138

Shadbolt, N., W. Hall, and T.B. Berners-Lee. *The Semantic Web revisited*. IEEE Intelligent Systems. 2006

Shieber, S. M. Separating Linguistic Analysis from Linguistic Theories, *In:* Reyle, U. and C. Rohrer (eds). *Natural Language Parsing and Linguistic Theory*. Reidel. 1988. pp 33-68

Shieber, S. M. The Problem of Logical-Form Equivalence. *Computational Linguistics 19:1*, 1993. pp 179-190

SICS. Documentation for SICStus Prolog 4. *http://sicstus.sics.se/documentation.html.* 2014

Sowa, J. *Conceptual structures: Information processes in Mind and Machine.* Addison-Wesley. 1984

Stabler, E. P. *The logical approach to syntax: foundations, specifications, and implementations of theories of government and binding*. MIT Press. 1992

Stabler, E. P. Varieties of crossing dependencies: structure dependence and mild context sensitivity. *Cognitive Science 28*. 2004. pp 699-720

Stabler, E.P. Computational perspectives on minimalism. *In:* C. Boekx (ed). *Handbook of Linguistic Minimalism.* OUP. 2010. pp 616-641

Steedman, M. Gapping as Constituent Coordination. *Linguistics and Philosophy 13:2*. 1990. pp 207-264

Steedman, M. *Surface Structure and Interpretation*. MIT Press. 1996

Steedman, M. *The Syntactic Process*. The MIT Press. 2000

Szabolcsi, A. Strong vs. Weak Islands. *In:* Everaert, M. and H. van Riemsdijk (eds.). *The Blackwell Companion to Syntax. Volume IV.* Blackwell. 2006. pp 479-531

Szabolcsi, A. and M. den Dikken. Islands. *Glot International 4:6*. 1999. pp 3-8

Tatu, M. and D. Boldovan. COGEX at RTE3. *Proceedings of the workshop on textual entailment and Paraphrasing*. ACL. 2007. pp 22-27

Turner, R. Two Issues in the Foundation of Semantic Theory. *In:* Chierchia, C., B. H. Partee, and R. Turner (eds). *Properties, Types and Meaning*. *Volume 1.* Kluwer. 1989. pp 63-84

Van 't Veer, M. *Solve.pl*. Unpublished paper. Leiden University, Dpt. of Linguistics. 2007.

Van Benthem, J. Questions about quantifiers. *Journal of Symbolic Logic 49:2.* 1984. pp 443-466

Van Benthem, J.F.A.K. *Essays in Logical Semantics*. Reidel. 1986

Van Benthem, J.F.A.K, *Language in Action*. North-Holland. 1991

Van Deemter. K. *Not exactly: In praise of vagueness*. OUP, 2010

Van de Woestijne, C.E. *A formal characterisation of the Delilah system*. MSc. thesis, LIACS, Leiden University. 1999

Van Dreumel, S. and P.-A. Coppen. Surface analysis of the verbal cluster in Dutch. *Linguistics 41:1*. 2003. pp 53-81

Van Noord, G. *Reversibility in natural language processing.*Rijksuniversiteit Utrecht. 1993

Van Noord, G. FSA utilities: A toolbox to manipulate finite-state automata. *In: Automata Implementation.* Springer. 1997. pp 87-108

Van Riemsdijk, H. *A Case Study in Syntactic Markedness: The Binding Nature of Prepositional Phrases*. The Peter de Ridder Press. 1978

Verkuyl, H.J. *On the compositional nature of the aspects.* Reidel. 1972

Verkuyl, H.J. *A theory of aspectuality*. CUP. 1993

Verkuyl, H.J. *Binary Tense.* CSLI. 2008

Vermaat, W. *Controlling Movement: Minimalism in a deuctive perspective*. Utrecht University. 1999

Vijay-Shanker, K. and D. J. Weir. Polynomial Time Parsing of Combinatory Categorial Grammars. *Proceedings of the 28th Meeting of the ACL*. 1990. pp 1-8

Von Stechow, A. Comparing semantic theories of comparison. *Journal of Semantics 3*. 1984. pp 1-77

Vosse, Th. and G. Kempen. Syntactic structure assembly in human parsing: a computational model based on competetive inhibition in a lexicalist grammar. *Cognition 75.* 2000. pp 105-143

Vossen, P., K. Hofmann, M. de Rijke, E. Tjong Kim Sang, and K. Deschacht (2007). The Cornetto Database: Architecture and User-Scenarios. *Proceedings of 7th Dutch-Belgian Information Retrieval Workshop* DIR, University of Leuven. 2007. pp 89-96

Wahlster, W. (ed.) *Verbmobil: Foundations of Speech-To-Speech Translation*. Springer. 2000

Wetzer, H. *Nouniness and verbiness: a typological study of adjectival predication*. Nijmegen University. 1995

Wheeler, D.W. *Aspects of a categorial theory of phonology.* Univ of Massachusetts at Amherst. 1981

White, M. and J. Baldridge. Adapting Chart Realization to CCG. *Proceedings Ninth European Workshop on Natural Language Generation.* Budapest. 2003

Williams, W.C. *Selected Essays of William Carlos Williams*. New Directions. 1969

Winter, Y. *Flexibility principles in Boolean Semantics*. MIT press. 2001

Wittenburg, K. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism*. Univ. of Texas, Austin. 1986

Wurmbrand, S. Verb Clusters, Verb Raising and Restructuring. *In:* M. Everaert and H. van Riemsdijk (eds). *The Blackwell Companion to Syntax*. *Volume V.* Blackwell. 2006. pp 229-344

Younger, D.H. Recognition and parsing of context-free languages in time n^3. *Information and Control 10(2).* 1967. pp 189-208

Zinn, Peter. *Categoriale Grammatica's en de Chomsky-hiërarchie. Een onderzoek naar de kracht van verschillende categoriale grammatica's*. MSc. thesis, Dept. of Computer Science, Leiden University. 1993

Zwart, J.W. *Dutch Syntax. A Minimalist Approach*. Rijksuniversiteit Groningen. 1993

Zwarts, F. Negatief polaire uitdrukkingen I. *Glot 4:1*. 1982

Zwarts, F. Determiners: a relational perspective. *In:* Ter Meulen, A.G.B. (ed), *Studies in Modeltheoretic Semantics.* Foris. 1983. pp 37-62

Zwarts, F. *Categoriale grammatica en algebraïsche semantiek*. Rijksuniversiteit Groningen. 1986

INDEX

A

adjacency · 51, 62, 119, 127, 129, 144
adjuncts · 6, 31-32, 61, 86, 88, 90-93, 108, 122-123, 131, 181, 272, 318
algebra · 5, 21, 25, 41, 53, 57-58, 60, 110, 140-141, 144, 168-169, 182, 188, 190-191, 254, 307
  additivity · 307
  multiplicativity · 57
algorithm · 17, 106, 113-115, 123, 126-131, 135-137, 141, 147-149, 151, 160, 170, 172-173, 198, 210, 220, 224, 254, 285, 291, 297, 326
anaphora · 29, 112, 157, 166-167, 180, 258
antimorphism · 316
argument list · 40, 47-51, 56, 68, 71-89, 91-92, 95, 97-99, 103-104, 119-122, 141
associativity · 130-132
asymmetry · 6, 50, 68-70, 84, 227
attribute-value matrix · 199, 245, 247, 271
automorphism · 122
auxiliary inversion · 85
axiom · 52-53, 63, 65, 92, 109, 119, 178, 180-181, 189

B

binarity · 303-304

C

collocational effects · 241, 265, 285
comparative · 321
complex symbol · 143, 149, 220, 231, 240, 246, 307
compositional complexity · 205-206
compositionality · 15, 19, 22, 26, 110-111, 154-156, 170-171, 174, 205, 218, 253, 264, 272, 314, 316, 323
compression · 290, 293-294
computability · 5, 12-16, 19, 22, 92, 141, 197, 242, 301
computation · 12-13, 16, 19-20, 28-29, 47, 62, 72, 75, 79, 83, 106, 115, 135, 155, 171, 197, 213, 259-260, 292, 302, 306, 314, 324, 327-328
computational linguistics · 12, 21-22, 114-115, 197, 203, 234, 237-238, 284, 331
concatenation · 19-20, 31, 34, 54, 140-142, 144, 193, 245, 316
conjunction · 12, 16, 20, 23, 107-109, 111-114, 136-137, 161, 174, 191, 194, 209, 214, 217, 226, 310-315, 321, 323, 326-327
conservativity · 8, 263, 302
constituency · 31, 70, 74, 130, 138-139, 237, 303, 309, 313, 317-318, 322-324, 329
constructicon · 22, 324, 326-328
coordination · 6, 32, 94, 106-115, 119, 136-137, 139, 317, 321-322
crossing dependency · 77, 98

D

database · 7, 22, 229-230, 240-241, 264, 284, 286-288, 290-291, 298
data structure · 126, 235, 240, 246, 259, 289, 294
deduction · 8, 53, 56-58, 64-65, 92, 105, 124, 196, 309-311

Delilah · 7, 12, 22, 30-32, 35, 41, 43, 71, 74-75, 82, 84, 91, 103, 105-106, 115, 133-135, 138-139, 145, 151, 154, 158, 160, 162, 167, 170-172, 174-175, 183, 189, 195, 197-199, 203-204, 207-208, 217-218, 220, 224, 234-237, 240, 242, 244-245, 252-254, 258-259, 263, 272, 278-280, 284-285, 288, 298, 301, 312
derivability · 52, 64-65, 67
determinism · 141, 226
disambiguation · 128, 203
discontinuity · 5-6, 25, 31-32, 55, 62-63, 73, 75, 78, 87, 92, 94-100, 103-106, 134, 139, 144, 237, 243

E

ellipsis · 8, 32, 112-114, 137, 157, 317, 322-323
embedding
  extensional · 175
  intensional · 158, 160, 171, 175, 222
entailment · 7, 15, 157-158, 163, 196, 210-211, 213-215, 217, 219-220, 226-227, 309, 311-314, 317, 319-320, 323-326, 328
essential separation · 191, 193-195
event · 29, 31, 36, 154-155, 157, 159-162, 173, 176-177, 179, 181-195, 200-201, 208-209, 212-213, 215, 221, 224, 232, 241, 248, 254, 256-257, 262, 273, 276-277, 279, 281-282, 299, 324
exception phrase · 318-319, 326
extended lexical unit · 190, 204-205, 207, 279
extraction · 55, 75, 90, 319
extraposition · 93

F

features · 35, 62, 100-101, 121, 151, 158, 198, 200, 219, 231, 234-235, 239-240, 248-249, 252, 268, 271, 277, 284, 286, 288, 290, 295-296, 304
finite verb · 31-32, 35, 38, 43, 184, 200, 269-270

G

generalized quantifiers · 18, 21, 168-169, 211, 258
generator · 11, 13, 17-18, 23, 138, 144-145, 149, 151, 174, 218, 220-221, 223, 226, 228, 230, 239, 285-286, 288, 291, 295-297, 327
grammar · 1, 5-9, 12-16, 20-23, 26-31, 33-34, 39, 41-45, 48-53, 55-62, 65-66, 69-71, 73-75, 77, 84, 91-92, 94, 97, 100-106, 108, 112, 114-115, 118-122, 125-132, 134, 138-139, 141-144, 149-151, 155-156, 170, 196-199, 203-204, 214, 220-221, 223, 230-231, 234, 236-237, 241-242, 244, 246-247, 254-255, 259-263, 268-270, 274, 278, 283, 285, 291, 301-307, 310-311, 315-316, 322-324, 327-331
  categorial grammar · 16, 21, 23, 31, 42-43, 55, 58, 61, 65, 69-70, 94, 100, 105, 108, 114, 118, 127, 130, 134, 170, 196, 223, 236-237, 255, 259, 291, 305-306
  categorial list grammar · 73, 237
  Chomsky hierarchy · 6, 28, 61, 114, 124-125, 323
  Construction Grammar · 203, 236, 329
  context-free grammar · 119
  Discourse Representation Theory · 156
  formal grammar · 22-23, 28, 50, 203, 241, 301
  generative capacity · 6, 119-125, 129, 197
  GPSG · 70
  HPSG · 7, 35, 70, 114, 199, 203, 233-234, 236-237, 240, 260, 265, 283-284, 329
  MCSL · 124-126
  meta-grammatical · 317
  minimalism · 100, 305
  Montague Grammar · 157-158, 199
  natural semantic metalanguage · 154
  TAG · 203
grammatical cycle · 315, 328
graph · 7, 22, 35, 37, 199, 217, 220, 225-226, 228, 231, 233-236, 239-240, 246, 260-264, 266, 268-271, 278-279, 283-285, 287-289, 296, 298-299, 303-304

H

hashing · 293
higher-order · 46, 93, 158, 165, 170, 254-255
homomorphism · 23, 57-58, 144, 254, 328

I

incompleteness · 8, 171, 301, 327-331
  incompleteness theorem · 327
infinitive *pro participio* · 81
intensionality
  de dicto · 177-178
  de re · 177-178, 239
islands · 32-33, 75-76, 79, 82, 90, 129, 131, 210, 222

L

lambda calculus · 19, 57, 70, 105
lambda conversion · 20, 257, 313, 329
lambda term · 105, 194, 199, 204, 314
language processing · 12, 115, 197, 203, 210, 214, 220, 238, 316, 331
lemma · 20, 64-65, 71, 119, 233, 240-241, 244, 265, 271-274, 277, 280, 284-285
lexicalist hypothesis · 230
lexicon · 7-8, 16, 21-23, 25-27, 34-35, 43, 45, 71, 80, 82, 89-92, 102, 118, 120-123, 130, 132, 134-135, 143, 151, 154-155, 160, 184, 191, 193, 197, 200, 203, 206-208, 214, 219-220, 223-225, 231, 233-246, 248-249, 251-253, 258-259, 263, 265-266, 269-270, 272, 274, 278-280, 283-288, 291-295, 297-299, 301, 324
linearization · 6, 41, 46, 51, 56-57, 59, 61, 68-70, 77, 95, 98-100, 102, 138, 140, 143-145, 147, 233, 237, 242-244, 277, 286, 302, 305
linear ordering · 40, 69, 138-139, 242
linguistic engineering · 15
locality · 75, 129, 235, 283, 303-304, 323
logic · 5, 11-12, 41, 56-58, 61, 100, 105, 109-110, 117, 145, 153, 157, 160, 162-163, 169-172, 180, 196, 198, 209, 213-214, 217, 220, 222, 236, 258, 260-261, 286, 288, 304-305, 311, 327, 330
logical form · 7, 22, 132-133, 135, 138, 149, 157, 160, 179-180, 186, 190, 195-198, 205, 208, 210-214, 218-220, 222, 225-227, 284, 298, 310-317, 326, 329-330
  applied logical form · 161, 172, 176, 197-198, 208-214, 217, 228
  flat logical form · 172, 176-177, 197-198, 208-222, 224-228
  normal logical form · 176
  stored logical form · 36, 132, 135, 159, 172, 184, 197-199, 201-210, 217-219, 226, 228, 232-233, 240, 248, 255-257, 266-267, 275-277, 281-282, 298-299

M

meaning · 1, 6, 8, 12-16, 18-23, 26-27, 29, 34, 109-112, 130, 153-160, 168, 170-171, 184, 189, 196-197, 199, 203-206, 208, 211, 215, 218-220, 222, 229-231, 241, 249-254, 256, 258, 264, 266, 269, 279, 301-303, 309-312, 315-318, 323-324, 330-331
meta-index · 296
model · 12, 14-15, 18, 29, 52-53, 63, 70, 91, 96, 99, 115, 153, 157, 175, 178, 190-191, 223, 229, 234, 236-238, 253, 258, 286-287, 329-330
morphology · 171, 227, 242, 244
movement · 32, 63, 87, 100, 118, 139, 323
multiplicative · 57, 312, 315-316

N

negation · 13, 57-58, 60, 162, 188-189, 210, 260, 279, 308, 315-316, 326
nominal constituent · 27, 42, 168, 184
nominalization · 317

O

object-oriented · 194, 290
operator · 40, 48, 54, 75, 118, 139, 162, 173-174, 186, 200, 209, 214, 222, 309, 318

P

parser · 6, 16-18, 23, 42, 115, 126-134, 136, 144, 149-151, 197, 220, 230, 294-295, 298
  backtracking · 126, 133, 150-151, 219, 221, 228, 297
  bottom-up · 127-128, 132-133, 228, 268, 298
  chart · 74, 92, 115, 126-133, 135-136, 298, 300
  polynomial time · 123-127, 137
  top-down · 127-128, 228, 268-269
partial recursive functions · 16
pluractionality · 190-191, 195
post-derivational · 17, 134-135, 158, 160, 167, 171, 210, 326
probabilistic · 127-128
Prolog · 80, 118, 123, 138-139, 143, 181, 257, 285-288, 290-297
pronouns · 32-33, 90, 121-122, 144, 185, 259
proposition · 14, 17, 19, 23, 52-53, 60, 63-64, 111-112, 145, 153-155, 157, 168, 175-179, 188, 207, 211, 215, 258, 261, 309-310, 313, 317, 319-320, 322-323, 326, 328

Q

qualia · 234, 239
quantification · 6, 58, 157, 159-163, 165, 167, 169-170, 178-181, 185-187, 189-190, 195, 224, 308

R

recursion · 5, 19-20, 25, 28-29, 33-34, 51, 139, 150, 171, 288, 290, 303-304
referentiality · 29, 181, 187-188, 249
relational database · 286-288, 290
resource sensitivity · 92, 109
robustness · 128, 130, 132-134, 136

S

scope · 6, 13, 17, 106, 139, 153, 160, 166-167, 170, 172-174, 178-182, 186, 188-189, 194-195, 198-199, 210-212, 222, 226, 235, 249, 254, 330
semantics · 8, 14-16, 20-22, 34, 43, 66, 108, 110, 117, 128, 131-132, 135, 138, 145, 156-158, 160, 162, 168, 171-172, 179, 182-184, 188, 190-191, 195-197, 199, 203-204, 206-207, 212-213, 217-222, 224, 230, 233, 238, 241-242, 248, 251-253, 258, 266, 270, 277, 279, 284, 287, 300-302, 306, 308-309, 311-312, 314-317, 319, 322-324, 326-331
Skolemization · 6, 180-181, 186-189
small clause · 89, 174, 214, 273, 278
spurious ambiguity · 20, 128, 130-132, 135, 329
stack · 40, 45, 51, 71-72, 80, 87, 89, 92, 118, 121-122, 124, 150, 165, 291
string algebra · 57
syntax · 5-6, 8, 15, 19, 21-22, 25, 28-31, 33-35, 37-39, 41, 43-46, 54-55, 57, 60, 62, 70-71, 74, 84, 90, 92, 102, 114-117, 122-123, 128, 133, 138-139, 151, 157, 166, 171, 179, 185, 190, 196-199, 203, 218-222, 230, 234, 239, 242, 245-246, 249, 252-253, 265-266, 273, 298-299, 301-303, 305-309, 313-316, 319, 321-324, 326, 328-331
syntax-semantics interface · 157, 218, 308, 314, 323, 330

T

template · 35-39, 42, 49, 82, 143, 146-149, 173, 184, 192, 199, 202-203, 231-232, 235, 242-243, 245-248, 253, 255-257, 259, 261, 264, 266-269, 271-277, 280-282, 284, 299-300
  subtemplate · 38-39
tense · 36, 49, 147, 157, 159-161, 176-177, 184-185, 202, 208-209, 221-222,

227, 232-233, 245, 248, 256-258, 262, 275-277, 282, 299, 317, 326
truth-functional · 13, 177-178
Turing machine · 14, 28
type · 16-19, 22, 27, 31, 36-38, 42-43, 45-47, 50, 52-55, 58-59, 68-69, 71-72, 79, 89-92, 94, 99-101, 105-106, 108-111, 117-119, 122, 124, 126-127, 129-130, 132-135, 137, 144, 146, 150-151, 154, 156, 165, 167, 169, 172, 175, 178, 180-181, 183-184, 189, 192, 200-201, 204, 228, 231, 233-234, 236, 238, 241, 246, 251, 253-254, 257-258, 266, 274-275, 279-280, 286-287, 290-291, 293-297, 300, 302-303, 305-306, 316-317, 319, 322-327
cancellation · 39-41, 47-56, 66, 70, 72-73, 76, 79, 82, 84, 87-89, 91, 95, 102, 109, 118, 120-122, 141-142, 144, 146-147, 150-151, 221
cancellation mode · 39-40, 89, 95, 121, 142
count invariance · 65, 109, 119, 123, 129
disharmonic composition · 54, 56, 93
flag · 38, 40-41, 48-49, 65, 68, 72-79, 81-86, 88-89, 91, 95, 98-99, 103-104, 118-119, 121, 187, 275-276
left rule · 52
merge mode · 41, 45, 48-50, 66, 68, 76, 79-82, 87, 91, 97, 102, 117, 122, 140
primary category · 39-41, 46, 48-49, 54, 59, 68-69, 72, 75-89, 91-92, 95-99, 103-105, 122-123, 140, 145
right rule · 52
secondary category · 39-40, 46, 48-49, 54, 56, 68-69, 72, 75-79, 81-92, 96-99, 103-104, 117, 122, 129, 140-141
type coercion · 234, 251

U

unbounded dependencies · 236
undecidable · 187, 211, 213, 251, 258, 303
underspecification · 6, 133, 137, 170-172, 206-207, 212, 221-223, 225, 330
unification · 7-8, 22-23, 34-35, 37-44, 133-134, 139, 143-147, 149, 157, 170, 196, 198-199, 205, 207, 217, 220, 222-223, 230-231, 233, 235-236, 239, 242, 244-247, 258, 260-266, 268, 270-273, 289, 298-300, 302-307, 313, 315-316, 319, 323-324, 326, 328, 330
graph unification · 199, 217, 239, 260

V

verbal constituent · 27
verb cluster · 30, 76, 81, 85, 90, 125, 129, 131
verb-second · 31, 43, 87, 274
veridicality · 317

W

wh · 36, 38, 72-75, 80, 82-84, 87-88, 92, 100, 103-104, 118, 121-122, 125, 129, 131, 140-143, 146, 202, 232, 243, 247, 256-257, 262, 282, 300
word order · 43, 70, 121, 221
worst case · 105, 126