



Universiteit
Leiden
The Netherlands

Testing object Interactions

Grüner, A.

Citation

Grüner, A. (2010, December 15). *Testing object Interactions*. Retrieved from <https://hdl.handle.net/1887/16243>

Version: Corrected Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/16243>

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

Testing Object Interactions

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus prof. mr. P.F. van der Heijden,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 15 december 2010
klokke 13.45 uur

door

Andreas Grüner
geboren te Nettetal, Duitsland
in 1974

Promotiecommissie

Promoter: Prof. dr. Frank S. de Boer
Co-promoter: Dr. Marcello Bonsangue
Dr. Martin Steffen (*Universitetet i Oslo*)
Referent: Dr. Bernhard Aichernig (*Technische Universität Graz*)
Overige leden: Prof. dr. Farhad Arbab
Prof. dr. Joost N. Kok

CONTENTS

1	Introduction	1
1.1	Object-oriented programming languages	2
1.1.1	<i>Java</i>	3
1.1.2	<i>C#</i>	4
1.2	Testing in the software development life-cycle	4
1.3	Unit testing object-oriented software	7
1.3.1	<i>JUnit</i>	7
1.3.2	<i>jMock</i>	10
1.3.3	<i>JMLUnit</i>	14
1.4	Testing approach in this thesis	16
1.5	Structure of the thesis	17
1.6	Relation to my previous scientific work	18
I	Testing Sequential Components	19
2	Java-like programming language – <i>Japl</i>	23
2.1	Syntax	24
2.2	Static semantics	26
2.3	Operational semantics	31
2.4	Extension by components: the <i>Japl</i> language	37
2.4.1	Syntax	38
2.4.2	Static Semantics	40
2.4.3	Operational Semantics	41
2.5	Traces and the notion of testing	51
3	The test specification language	57
3.1	Extension by expectations	59
3.2	Syntax	64
3.3	Static semantics	66
3.4	Operational semantics	73
3.5	Example	76
3.6	Executability and input enabledness	79

3.7	Satisfiability and completeness	81
4	Code generation	83
4.1	Preprocessing	85
4.1.1	Labeling mechanism	85
4.1.2	Variable binding	93
4.2	<i>Japl</i> code generation	95
4.3	Generation of the test program.	102
4.4	Correctness of the code generation	103
4.5	Failure report and faulty specifications	111
5	Further possible extensions	115
5.1	Specification classes	115
5.2	Programming classes	122
5.3	Subtyping and inheritance	126
II	Testing Multi-threaded Components	135
6	Concurrent programming language – <i>CoJapl</i>	139
6.1	Syntax	139
6.2	Static semantics	140
6.3	Operational semantics	141
7	Specification language and code generation	151
7.1	Syntax	154
7.2	Static semantics	157
7.3	Operational semantics	158
7.4	Test code generation	159
8	Concluding remarks	163
	Bibliography	167
III	Proofs	173
	Appendices	175
A	Subject reduction	177
B	Compositionality	181

C Code generation	191
C.1 Preprocessing	191
C.2 Anticipation	198
C.3 Correctness of the generated code	207
Summary	223
Samenvatting	225
Curriculum Vitæ	227

LIST OF TABLES

2.1	Simple <i>Java</i> -like language: syntax	25
2.2	Simple <i>Java</i> -like language: type system (program parts up to stmts)	28
2.3	Simple <i>Java</i> -like language: type system (exprs)	30
2.4	Variable evaluation	33
2.5	Expression evaluation	33
2.6	Auxiliary notations	34
2.7	Simple <i>Java</i> -like language: operational semantics	36
2.8	<i>Japl</i> language : syntax	39
2.9	<i>Japl</i> language: type system (stmts)	40
2.10	Label check for incoming communication	45
2.11	Free variables	47
2.12	<i>Japl</i> language: operational semantics (ext.)	48
2.13	<i>Japl</i> language: traces	51
3.1	Specification language for <i>Japl</i> : syntax	65
3.2	Specification language for <i>Japl</i> : type system (stmts)	69
3.3	Specification language for <i>Japl</i> : operational semantics (external)	74
4.1	Preprocessing: labeling and anticipation ($prep_{out}$)	89
4.2	Preprocessing: labeling and anticipation ($prep_{in}$)	90
4.3	Initial method and constructor code	99
4.4	Code-generation: method extension	99
4.5	Generation of <i>Japl</i> code ($code_{out}$)	101
4.6	Generation of <i>Japl</i> code ($code_{in}$)	102
5.1	Extension by specification classes: syntax	116
5.2	Extension by specification classes: type system (stmts)	118
5.3	Extension by specification classes: operational semantics	119
5.4	Extension by programming classes: syntax	123
5.5	Extension by programming classes: type system (stmts)	125
5.6	<i>Japl</i> with subclassing: syntax	127
5.7	<i>Japl</i> with subclassing: type system (stmts)	129
5.8	<i>Japl</i> with subclassing: operational semantics (int.)	131
5.9	Example: cross-border inheritance	132

5.10	<i>Japl</i> with subclassing: operational semantics (ext.)	133
6.1	<i>CoJapl</i> language: syntax	140
6.2	<i>CoJapl</i> language: type system (stmts)	142
6.3	<i>CoJapl</i> language: type system (exprs)	143
6.4	<i>CoJapl</i> language: operational semantics (internal, part 1)	145
6.5	<i>CoJapl</i> language: operational semantics (internal, part 2)	146
6.6	<i>CoJapl</i> language: operational semantics (external)	148
6.7	<i>CoJapl</i> language: traces	150
7.1	Specification language for <i>CoJapl</i> : syntax	155
7.2	<i>CoJapl</i> example specifications	156
7.3	Specification language for <i>CoJapl</i> : type system (stmts)	157
7.4	Specification language for <i>CoJapl</i> : operational semantics (external)	159
7.5	<i>CoJapl</i> code generation: mutual exclusion	162
C.1	Anticipation-valid code (static)	199
C.2	Anticipation-valid configurations (dynamic)	201
C.3	Simulation relation for statements	207
C.4	Well-typedness of dynamic specification code mc_{sl}	211

LIST OF FIGURES

1.1	Software development process and testing levels	6
1.2	Novel testing approach	17
2.1	Notion of component	39
4.1	Testing framework	84

