Scheduled protocol programming

Dokter, K.P.C.

Chapter 5

Protocols as Constraints

Given the standardized specification of Reo connections in Treo developed in Chapter 4, we now proceed with the development of a compiler that accepts Treo as input. The first step in the construction of a compiler for Treo is the selection of an appropriate semantics of Treo components. The selected semantics has serious effects on the implementability and scalability of the resulting compiler, and this decision should therefore not be taken lightly. The current chapter proposes a semantics for Reo connectors whose intermediate representation is significantly smaller than the representation of existing semantics, without sacrificing performance[1]. As a result, our approach can compile Reo connectors for which it was previously infeasible to generate efficient code.

Over a decade ago, Baier et al. introduced *constraint automata* for the specification of interaction protocols [BSAR06]. Constraint automata feature a powerful composition operator that *preserves synchrony*: composite constructions not only yield intuitively meaningful asynchronous protocols but also synchronous protocols. Constraint automata have been used as basis for tools, like compilers and model checkers. Jongmans developed Lykos: a compiler that translates constraint automata into reasonably efficient executable Java code [Jon16]. Baier, Blechmann, Klein, and Klüppelholz developed Vereofy, a model checker for constraint automata [BBKK09, Klü12]. Unfortunately, like every automaton model, composition of constraint automata suffers from state space and transition space explosions. These explosions limit the scalability of the tools based on constraint automata.

To improve scalability, Clarke et al. developed a compiler that translates a constraint automaton to a first-order formula [CPLA11]. The transitions of the constraint automaton correspond to the solutions of this formula. At run time, a generic constraint solver finds these solutions and simulates the automaton. Since composition and abstraction for constraint automata respectively correspond to conjunction and existential quantification, the first-order specification does not suffer from state space or transition space explosion. However, the approach proposed by Clarke et al. only delays the complexity until run time: calling a generic constraint solver at run time imposes a significant overhead.

Jongmans realized that the overhead of this constraint solver is not always

[1]The work in this chapter is based on [DA18a]

necessary. He developed a commandification algorithm that accepts constraints without disjunctions (i.e., conjunctions of literals) and translates them into a small imperative program [JA16b]. The resulting program is a light-weight, tailor-made constraint solver with minimal run time overhead. Since commandification accepts only constraints without disjunction, Jongmans applied this technique to data constraints on individual transitions in a constraint automaton. Relying on constraint automata, his approach still suffers from scalability issues [JKA17].

We aim to prevent state space and transition space explosions by combining the ideas of Clarke et al. and Jongmans. To this end, we present the language of *stream constraints*: a generalization of constraint automata based on temporal logic. A stream constraint is an expression that relates streams of observed data at different locations (Section 5.2). We identify a subclass of stream constraints, called *regular (stream) constraints*, which is closed under composition and abstraction (Section 5.3). Regular constraints can be viewed as a constraint automata, and conjunction of reflexive regular constraints is similar to composition of constraint automata (Section 5.4).

A straightforward application of the commandification algorithm of Jongmans to regular stream constraints entails transforming a stream constraint into disjunctive normal form and applying the algorithm to each clause separately. However, the number of clauses in the disjunctive normal form may grow exponentially in the size of the composition. To prevent such exponential blowups of the size of the formula, we recognize and exploit symmetries in the disjunctive normal form. Each clause in the disjunctive normal form can be constructed from a set of basic stream constraints, which we call *rules*. This idea allows us to represent a single large constraint as certain combination of a set of smaller constraints, called the *rule-based form* (Section 5.5). We express the composition of stream constraints in terms of the rule-based normal form (Section 5.6), and show that, for *simple* sets of rules, the number of rules to describe the composition is only linear in the size of the composition (Section 5.7). The class of stream constraints defined by a simple set of rules contains constraints for which the size of the disjunctive normal form explodes, which shows that our approach improves upon existing approaches by Clarke et al. and Jongmans. We express abstraction on stream constraints in terms of the rule-based normal form and provide a sufficient condition under which the number of rules remains constant (Section 5.8). Finally, we conclude and point out future work (Section 5.10).

5.1 Related work

Representation of stream constraints in rule-based form is part of a larger line of research on symbolic approaches, such a symbolic model checking [BCM+92, BCH+97, KNSW07] and symbolic execution [CDE+07]. These approaches not only use logic (cf., SAT solving techniques [Kem12, Ehl10] for verification), but also other implicit representations, like binary decision diagrams [Bry86] and Petri nets [Mur89]. Petri nets offer a small representation of protocols with an exponentially large state space. While our focus is more on compilation, Petri nets have been studied in the context of verification. As inspiration for future work, it is interesting to study the similarities between Petri nets and stream constraints.

Since regular stream constraints correspond to constraint automata, we can view regular stream constrains as a restricted temporal logic for which distributed synthesis is easy. In general, distributed (finite state) synthesis of protocols is undecidable [PR89, PR90]. Pushing the boundary from regular to a larger class of stream constraints can be useful for more effective synthesis methods.

5.2 Syntax and semantics

The semantics of constraint automata is defined as a relation over *timed data streams* [AR02], which are pairs, each consisting of a non-decreasing stream of time stamps and a stream of observed (exchanged) data items. The primary significance of time streams is the proper alignment of their respective data streams, by allowing "temporal gaps" during which no data is observed. For convenience, we drop the time stream and model protocols as relations over streams of data, augmented by a special symbol that designates "no-data" item.

We first define the abstract behavior of a protocol C. Fix an infinite set X of variables, and fix a non-empty set of user-data $Data \supseteq \{0\}$ that contains a datum 0. Consider the *data domain* $D = Data \cup \{*\}$ of data stream items, where we use the "no-data" symbol $* \in D \setminus Data$ to denote the absence of data. We model a single execution of protocol C as a function

$$\theta : X \longrightarrow D^{\mathbb{N}} \tag{5.1}$$

that maps every variable $x \in X$ to a function $\theta(x) : \mathbb{N} \longrightarrow D$ that represents a stream of data at location x. We call θ a *data stream tuple* (over X and D). For all $n \in \mathbb{N}$ and all $x \in X$, the value $\theta(x)(n) \in D$ is the data that we observe at location x and time step n. If $\theta(x)(n) = *$, we say that no data is observed at x in step n (i.e., we may view θ as a partial map $\mathbb{N} \times X \rightharpoonup Data$). The behavior of protocol C consists of the set

$$\mathcal{L}(C) \ \subseteq \ (D^{\mathbb{N}})^X \tag{5.2}$$

of all possible executions of C, called the *accepted language* of C. We can think of accepted language $\mathcal{L}(C)$ as a relation over data streams. In this chapter, we study protocols that are defined as a *stream constraint*:

Definition 5.2.1 (Stream constraints)**.** A stream constraint ϕ is an expression generated by the following grammar

$$\begin{array}{rcl} \phi & ::= & \bot \mid t_0 \doteq t_1 \mid \phi_0 \wedge \phi_1 \mid \neg\phi \mid \exists x\phi \mid \Box\phi \\ t & ::= & x \mid \mathbf{d} \mid t' \end{array}$$

where $x \in X$ is a variable, $d \in D$ is a datum, and t is a stream term.

We use the following standard syntactic sugar: $\top = \neg\bot$, $\phi_0 \vee \phi_1 = \neg(\neg\phi_0 \wedge \neg\phi_1)$, $\Diamond\phi = \neg\Box\neg\phi$, $(t_1 \not\doteq t_2) = \neg(t_1 \doteq t_2)$, $(t_1 \doteq \cdots \doteq t_n) = (t_1 \doteq t_2 \wedge \cdots \wedge t_{n-1} \doteq t_n)$, $t^{(0)} = t$, and $t^{(k+1)} = (t^{(k)})'$, for all $k \geq 0$. Following Rutten [Rut01], we call $t^{(k)}$, $k \geq 0$, the k-th *derivative* of term t.

We interpret a stream constraint as a constraint over streams of data in $D^{\mathbb{N}}$. For a datum $d \in D$, $\mathbf{d}$ is the constant stream defined as $\mathbf{d}(n) = d$, for all $n \in \mathbb{N}$. The

operator $(-)'$, called *stream derivative*, drops the head of the stream and is defined as $\sigma'(n) = \sigma(n+1)$, for all $n \in \mathbb{N}$ and $\sigma \in D^{\mathbb{N}}$. Streams can be related by $\doteq$ that expresses equality of their heads: $x \doteq y$ iff $x(0) = y(0)$, for all $x, y \in D^{\mathbb{N}}$. The modal operator $\Box$ allows us to express that a stream constraint holds after applying any number of derivatives to all variables. For example, $\Box(x \doteq y)$ iff $x^{(k)}(0) = y^{(k)}(0)$, for all $k \in \mathbb{N}$ and $x, y \in D^{\mathbb{N}}$. Stream constraints can be composed via conjunction $\wedge$, or negated via negation $\neg$. Streams can be hidden via existential quantification $\exists$.

Each stream term t evaluates to a data stream in $D^{\mathbb{N}}$. Let $\theta : X \longrightarrow D^{\mathbb{N}}$ be a data stream tuple. We extend the domain of θ from the set of variables X to the set of terms $T \supseteq X$ as follows: we define $\theta : T \longrightarrow D^{\mathbb{N}}$ via $\theta(\mathbf{d}) = \mathbf{d}$ and $\theta(t') = \theta(t)'$, for all $d \in D$ and terms $t \in T$.

Next, we interpret a stream constraint ϕ as a relation over streams.

Definition 5.2.2 (Semantics)**.** The language $\mathcal{L}(\phi) \subseteq (D^{\mathbb{N}})^X$ of a stream constraint ϕ over variables X and data domain D is defined as

1. $\mathcal{L}(\bot) = \emptyset$;
2. $\mathcal{L}(t_0 \doteq t_1) = \{\theta : X \longrightarrow D^{\mathbb{N}} \mid \theta(t_0)(0) = \theta(t_1)(0)\}$;
3. $\mathcal{L}(\phi_0 \wedge \phi_1) = \mathcal{L}(\phi_0) \cap \mathcal{L}(\phi_1)$;
4. $\mathcal{L}(\neg\phi) = (D^{\mathbb{N}})^X \setminus \mathcal{L}(\phi)$;
5. $\mathcal{L}(\exists x\phi) = \{\theta : X \longrightarrow D^{\mathbb{N}} \mid \theta[x \mapsto \sigma] \in \mathcal{L}(\phi)$, for some $\sigma \in D^{\mathbb{N}}\}$;
6. $\mathcal{L}(\Box\phi) = \{\theta : X \longrightarrow D^{\mathbb{N}} \mid \theta^{(k)} \in \mathcal{L}(\phi)$, for all $k \geq 0\}$,

where $\theta[x \mapsto \sigma] : X \longrightarrow D^{\mathbb{N}}$ is defined as $\theta[x \mapsto \sigma](x) = \sigma$ and $\theta[x \mapsto \sigma](y) = \theta(y)$, for all $y \in X \setminus \{x\}$; and $\theta^{(k)} : X \longrightarrow D^{\mathbb{N}}$ is defined as $\theta^{(k)}(x) = \theta(x^{(k)})$, for all $x \in X$.

Let ϕ and ψ be two stream constraints and $\theta : X \longrightarrow D^{\mathbb{N}}$ a data stream tuple. We say that θ *satisfies* ϕ (and write $\theta \models \phi$), whenever $\theta \in \mathcal{L}(\phi)$. We say that ϕ *implies* ψ (and write $\phi \models \psi$), whenever $\mathcal{L}(\phi) \subseteq \mathcal{L}(\psi)$. We call ϕ and ψ *equivalent* (and write $\phi \equiv \psi$), whenever $\mathcal{L}(\phi) = \mathcal{L}(\psi)$.

Example 5.2.1. One of the simplest stream constraints is $\mathsf{Sync}(a, b)$, which is defined as $\Box(a \doteq b)$. Constraint $\mathsf{Sync}(a, b)$ encodes that the data streams at a and b are equal: $\theta(a)(k) = \theta(b)(k)$, for all $k \in \mathbb{N}$ and all $\theta \in (D^{\mathbb{N}})^X$. Therefore, $\mathsf{Sync}(a, b)$ synchronizes the data flow observed at ports a and b.

Conjunction $\wedge$ and existential quantification $\exists$ provide natural operators for composition and abstraction for stream constraints. For example, the composition $\mathsf{Sync}(a, b) \wedge \mathsf{Sync}(b, c)$ synchronizes ports a, b, and c. Hiding port b yields $\exists b(\mathsf{Sync}(a, b) \wedge \mathsf{Sync}(b, c))$, which is equivalent to $\mathsf{Sync}(a, c)$. $\Diamond$

Example 5.2.2. Recall that $x^{(k)}$, for $k \geq 0$, is the k-th derivative of x. We can express that a stream x is periodic via the stream constraint $\Box(x^{(k)} \doteq x)$, for some $k \geq 1$. For $k = 1$, stream x is constant, like $\mathbf{0}$ and $*$. $\Diamond$

Example 5.2.3. The stream constraint $\mathsf{FIFO}(a, b, m)$ defined as $m \doteq * \wedge \Box((a \doteq m' \doteq \mathbf{0} \wedge b \doteq m \doteq *) \vee (a \doteq m' \doteq * \wedge b \doteq m \doteq \mathbf{0}) \vee (a \doteq b \doteq * \wedge m' \doteq m))$ models a 1-place buffer with input location a, output location b, and memory location m that can be full ($m \doteq \mathbf{0}$) or empty ($m \doteq *$). $\Diamond$

Example 5.2.4. Recall that $*$ models absence of data. Stream constraint $\Box\Diamond(a \neq *)$ expresses that always eventually we observe some datum at a. A constraint of such form can be used to define fairness. $\Diamond$

5.3 Regular constraints

We identify a subclass of stream constraints that naturally correspond to constraint automata. We first introduce some notation.

To denote that a string s *occurs* as a substring in a stream constraint ϕ or a stream term t, we write $s \in \phi$ or $s \in t$, respectively.

Every stream constraint ϕ admits a set $\mathrm{free}(\phi) \subseteq X$ of *free variables*, defined inductively via $\mathrm{free}(\bot) = \emptyset$, $\mathrm{free}(t_0 \doteq t_1) = \{x \in X \mid x \in t_0 \text{ or } x \in t_1\}$, $\mathrm{free}(\phi_0 \wedge \phi_1) = \mathrm{free}(\phi_0) \cup \mathrm{free}(\phi_1)$, $\mathrm{free}(\neg\phi) = \mathrm{free}(\Box\phi) = \mathrm{free}(\phi)$, and $\mathrm{free}(\exists x\phi) = \mathrm{free}(\phi) \setminus \{x\}$.

For every variable $x \in X$, we define the *degree of x in ϕ* as

$$\deg_x(\phi) \;=\; \max(\{-1\} \cup \{k \geq 0 \mid x^{(k)} \in \phi\}),$$

and the *degree of* ϕ as $\deg(\phi) = \max_{x \in X} \deg_x(\phi)$. Note that for $x \notin \phi$ we have $\deg_x(\phi) = -1$. For $k \geq 0$, we write $\mathrm{free}^k(\phi) = \{x \in \mathrm{free}(\phi) \mid \deg_x(\phi) = k\}$ for the set of all free variables of ϕ of degree k.

We call a variable x of degree zero in ϕ a *port variable* and write $P(\phi) = \mathrm{free}^0(\phi)$ for the set of port variables of ϕ. We call a variable x of degree one or higher in ϕ a *memory variable* and write $M(\phi) = \bigcup_{k \geq 1} \mathrm{free}^k(\phi)$ for the set of memory variables of ϕ.

Definition 5.3.1 (Regular)**.** A stream constraint ϕ is regular if and only if $\phi = \psi_0 \wedge \Box\psi$, such that $\Box \notin \psi_0 \wedge \psi$ and $\deg_x(\psi_0) < \deg_x(\psi) \leq 1$, for all $x \in X$.

For a regular stream constraint $\phi = \psi_0 \wedge \Box\psi$, we refer to ψ_0 as the *initial condition* of ϕ and we refer to ψ as the *invariant* of ϕ. Stream constraints $\mathsf{Sync}(a, b)$ and $\mathsf{FIFO}(a, b, m)$ in Examples 5.2.1 and 5.2.3 are regular stream constraints.

A regular stream constraint ϕ has an operational interpretation in terms of a labeled transition system $[\![\phi]\!]$. States of the transition system consist of maps $q : M(\phi) \longrightarrow D$ that assign data to memory locations, and its labels consist of maps $\alpha : P(\phi) \longrightarrow D$ that assign data to ports. We write $Q(\phi)$ for the set of states of ϕ and $A(\phi)$ for the set of labels of ϕ.

Definition 5.3.2 (Operational semantics)**.** The operational semantics $[\![\phi]\!]$ of a regular stream constraint $\phi = \psi_0 \wedge \Box\psi$ consists of a labeled transition system $(Q(\phi), A(\phi), \rightarrow, Q_0)$, with set of states $Q(\phi)$, set of labels $A(\phi)$, set of transitions $\rightarrow = \{(q_\phi(\theta), q_\phi(\theta'), \alpha_\phi(\theta)) \mid \theta \in \mathcal{L}(\psi)\}$, and set of initial states $Q_0 = \{q_\phi(\theta) \mid \theta \in \mathcal{L}(\psi_0 \wedge \psi)\}$, where

1. $q_\phi(\theta) : M(\phi) \longrightarrow D$ is defined as $q_\phi(\theta)(x) = \theta(x)(0)$, for $x \in M(\phi)$; and

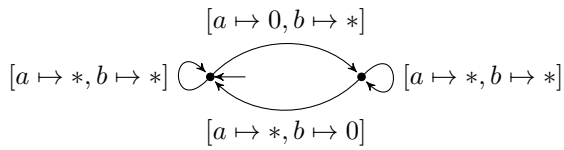


Figure 5.1: Semantics of $\mathsf{FIFO}(a, b, m)$ over the trivial data domain $\{0, *\}$.

2. $\alpha_\phi(\theta) : P(\phi) \longrightarrow D$ is defined as $\alpha_\phi(\theta)(x) = \theta(x)(0)$, for $x \in P(\phi)$.

and θ' is defined as $\theta'(x)(n) = \theta(x)(n+1)$, for all $x \in X$ and $n \in \mathbb{N}$.

Example 5.3.1. Consider the regular stream constraint $\mathsf{FIFO}(a, b, m)$ from Example 5.2.3. Note that in this example, the set of ports equals $\text{free}^0(\mathsf{FIFO}) = \{a, b\}$ and the set of memory locations equals $\text{free}^1(\mathsf{FIFO}) = \{m\}$. The semantics of $\mathsf{FIFO}(a, b, m)$ over the trivial data domain $D = \{0, *\}$ consists of 4 transitions:

1. $([m \mapsto *], [m \mapsto 0], [a \mapsto 0, b \mapsto *])$;
2. $([m \mapsto 0], [m \mapsto *], [a \mapsto *, b \mapsto 0])$; and
3. $([m \mapsto d], [m \mapsto d], [a \mapsto *, b \mapsto *])$, for every $d \in \{*, 0\}$.

Figure 5.1 shows the semantics of FIFO over the trivial data domain. ◊

Equivalent stream constraints do not necessarily have the same operational semantics. We are, therefore, interested in operational equivalence of constraints:

Definition 5.3.3 (Operational equivalence)**.** Stream constraints ϕ and ψ are operationally equivalent ($\phi \simeq \psi$) iff $\phi \equiv \psi$ and $\text{free}^k(\phi) = \text{free}^k(\psi)$, for $k \geq 0$.

Example 5.3.2. Let ϕ be a stream constraint, let t be a term and let $x \notin t$ be a variable that does not occur in t. Then, we have $\exists x(x \doteq t \wedge \phi) \equiv \phi[t/x]$, where $\phi[t/x]$ is obtained from ϕ by substituting t for every free occurrence of x. Observe that $\exists x(x \doteq t \wedge \phi)$ and $\phi[t/x]$ may admit different sets of free variables: if ϕ is just $\top$ and t is a variable y, the equivalence amounts to $\exists x(x \doteq y) \equiv \top$. To ensure that the free variables coincide, we can add the equality $t \doteq t$ and obtain the operational equivalence $\exists x(x \doteq t \wedge \phi) \simeq \phi[t/x] \wedge t \doteq t$. ◊

Operational equivalence of stream constraints ϕ and ψ implies that their operational semantics are identical, i.e., $[\![\phi]\!] = [\![\psi]\!]$. It is possible to introduce weaker equivalences by, for example, demanding that $[\![\phi]\!]$ and $[\![\psi]\!]$ are only weakly bisimilar. Such weaker equivalence offer more room for simplification of stream constraints than operational equivalence does. As our work does not need this generality, we leave the study of such weaker equivalences as future work.

The most important operations on stream constraints are composition ($\wedge$) and hiding ($\exists$). The following result shows that regular stream constraints are closed under conjunction and existential quantification of degree zero variables.

Theorem 5.3.1. *For all stream constraints ϕ and ψ and variables x, we have*

1. $\Box\phi \wedge \Box\psi \equiv \Box(\phi \wedge \psi)$; *and*

2. $\exists x \Box \phi \equiv \Box \exists x \phi$*, whenever* $\deg_x(\phi) \leq 0$ *and* $\Box \notin \phi$*.*

Proof. For assertion 1, $\mathcal{L}(\Box\phi \wedge \Box\psi) = \{\theta \in (D^{\mathbb{N}})^X \mid \forall k \geq 0 : \theta^{(k)} \models \phi \wedge \psi\} = \mathcal{L}(\Box(\phi \wedge \psi))$ shows that $\Box\phi \wedge \Box\psi \equiv \Box(\phi \wedge \psi)$.

For assertion 2, suppose that $\deg_x(\phi) \leq 0$ and $\Box \notin \phi$. We show that $\theta \in \mathcal{L}(\Box\exists x\phi)$ if and only if $\theta \in \mathcal{L}(\exists x \Box \phi)$, for all $\theta \in (D^{\mathbb{N}})^X$. By Definition 5.2.2, this equivalence can be written as

$$\theta^{(k)}[x \mapsto \mu_k] \models \phi \quad \Leftrightarrow \quad (\theta[x \mapsto \sigma])^{(k)} \models \phi, \tag{5.3}$$

for all $k \geq 0$, $\sigma \in D^{\mathbb{N}}$, and $\mu_k \in D^{\mathbb{N}}$ such that $\mu_k(0) = \sigma^{(k)}(0)$.

To prove Equation (5.3), we proceed by induction on the length of ϕ:

Case 1 ($\phi := \bot$): Since $\mathcal{L}(\bot) = \emptyset$, Equation (5.3) holds trivially.

Case 2 ($\phi := t_0 \doteq t_1$): Observe that, since $\deg_x(\phi) \leq 0$, for all terms t, we have $x \in t$ iff $t = x$. We conclude Equation (5.3) from $\mu_k(0) = \sigma^{(k)}(0)$ and

$$\theta^{(k)}[x \mapsto \mu_k](t)(0) = \left\{ \begin{matrix} \mu_k(0) & \text{if } t = x \\ \theta^{(k)}(t)(0) & \text{if } t \neq x \end{matrix} \right\} = (\theta[x \mapsto \sigma])^{(k)}(t)(0).$$

Case 3 ($\phi := \psi_0 \wedge \psi_1$): By the induction hypothesis, Equation (5.3) holds for ψ_0 and ψ_1. By conjunction of Equation (5.3), we conclude Equation (5.3) for ϕ.

Case 4 ($\phi := \neg\psi$): By the induction hypothesis, Equation (5.3) holds for ψ. By contraposition of Equation (5.3), we conclude Equation (5.3) for ϕ.

Case 5 ($\phi := \exists y \psi$): If $y = x$, then $x \notin \text{free}(\phi)$ and both sides in Equation (5.3) are equivalent to $\theta^{(k)} \models \phi$. Hence, Equation (5.3) holds for $y = x$. Suppose $y \neq x$. Then, $\theta^{(k)}[x \mapsto \mu_k] \models \phi$ is equivalent to $(\theta[y \mapsto \tau])^{(k)}[x \mapsto \mu_k] \models \psi$, for some $\tau \in D^{\mathbb{N}}$. Applying the induction hypothesis for θ equal to $\theta[y \mapsto \tau]$, we conclude that $\theta^{(k)}[x \mapsto \mu_k] \models \phi$ is equivalent to $(\theta[y \mapsto \tau][x \mapsto \sigma])^{(k)} \models \psi$, for some $\tau \in D^{\mathbb{N}}$. Since $y \neq x$, we conclude that Equation (5.3) holds.

We conclude that the claim holds for all ϕ with $\deg_x(\phi) \leq 0$ and $\Box \notin \phi$. □

5.4 Reflexive constraints

Conjunction of stream constraints is a simple syntactic composition operator with clear semantics: a data stream tuple θ satisfies a conjunction $\phi_0 \wedge \phi_1$ if and only if θ satisfies both ϕ_0 and ϕ_1. In view of the semantics of regular stream constraints in Definition 5.2.2, it is less obvious how $[\![\phi_0 \wedge \phi_1]\!]$ relates to $[\![\phi_0]\!]$ and $[\![\phi_1]\!]$. The following result characterizes their relation when no memory is shared.

Theorem 5.4.1. *Let* ϕ_0 *and* ϕ_1 *be regular stream constraints such that* $\text{free}(\phi_0) \cap \text{free}(\phi_1) \subseteq P(\phi_0 \wedge \phi_1)$*, and let* $(q_i, q_i', \alpha_i) \in Q(\phi_i)^2 \times A(\phi_i)$*, for* $i \in \{0, 1\}$*. The following are equivalent:*

1. $q_0 \xrightarrow{\alpha_0} q_0'$ *in* $[\![\phi_0]\!]$*,* $q_1 \xrightarrow{\alpha_1} q_1'$ *in* $[\![\phi_1]\!]$*, and* $\alpha_0|_{P(\phi_1)} = \alpha_1|_{P(\phi_0)}$*;*

2. $q_0 \cup q_1 \xrightarrow{\alpha_0 \cup \alpha_1} q_0' \cup q_1'$ *in* $[\![\phi_0 \wedge \phi_1]\!]$*,*

where $|$ *is restriction of maps, and* $\cup$ *is union of maps.*

Proof. Write $\phi_i = \psi_{i0} \wedge \Box\psi_i$, with $\Box \notin \psi_{i0} \wedge \psi_i$ and $\deg_x(\psi_{i0}) < \deg_x(\psi_i) \leq 1$, for all $x \in X$. Then, $\text{free}^k(\phi_i) = \text{free}^k(\psi_i)$, for all $i, k \in \{0, 1\}$.

Suppose that assertion 1 holds. By Definition 5.2.2, we find, for all $i \in \{0, 1\}$, some $\theta_i \in \mathcal{L}(\psi_i)$ such that $q_i = q_{\phi_i}(\theta_i)$, $q'_i = q_{\phi_i}(\theta'_i)$, and $\alpha_i = \alpha_{\phi_i}(\theta_i)$. Define $\theta : X \longrightarrow D^{\mathbb{N}}$ by $\theta(x) = \theta_i(x)$, if $x \in \text{free}(\phi_i)$, and $\theta(x) = *$, otherwise. Since $\text{free}(\phi_0) \cap \text{free}(\phi_1) \subseteq P(\phi_0 \wedge \phi_1)$ and $\alpha_0|_{P(\phi_1)} = \alpha_1|_{P(\phi_0)}$, we have that $\theta_0(x) = \theta_1(x)$, for all $x \in \text{free}(\phi_0) \cap \text{free}(\phi_1)$. Hence, θ is well-defined. By construction, $\theta \models \psi_0$ and $\theta \models \psi_1$. By Definition 5.2.2, we have $\theta \models \psi_0 \wedge \psi_1$. By Theorem 5.3.1, we have $\phi_0 \wedge \phi_1 = \psi_{00} \wedge \psi_{10} \wedge \Box(\psi_0 \wedge \psi_1)$. Since $q_0 \cup q_1 = q_{\phi_0 \wedge \phi_1}(\theta)$, $q'_0 \cup q'_1 = q_{\phi_0 \wedge \phi_1}(\theta')$, and $\alpha_0 \cup \alpha_1 = \alpha_{\phi_0 \wedge \phi_1}(\theta)$, we conclude assertion 2.

Suppose that assertion 2 holds. We find some $\theta \in \mathcal{L}(\psi_0 \wedge \psi_1)$, such that $q_0 \cup q_1 = q_\theta$, $q'_0 \cup q'_1 = q_{\theta'}$, and $\alpha_0 \cup \alpha_1 = \alpha_\theta$. Then, we conclude assertion 1, for $q_i = q_{\phi_i}(\theta)$, $q'_i = q_{\phi_i}(\theta')$, and $\alpha_i = \alpha_{\phi_i}(\theta)$. □

Stream constraints ϕ_0 and ϕ_1 without shared variables ($\text{free}(\phi_0) \cap \text{free}(\phi_1) = \emptyset$) seem completely independent. However, Theorem 5.4.1 shows that their composition $\phi_0 \wedge \phi_1$ admits a transition only if ϕ_0 and ϕ_1 admit respective local transitions (q_0, q'_0, α_0) and (q_1, q'_1, α_1), such that $\alpha_0|_{P(\phi_1)} = \alpha_1|_{P(\phi_0)}$. Since ϕ_0 and ϕ_1 do not share variables, the latter condition on α_0 and α_1 is trivially satisfied. Still, for one protocol ϕ_i, with $i \in \{0, 1\}$, to make progress in the composition $\phi_0 \wedge \phi_1$, constraint ϕ_{1-i} must admit an idling transition.

To allow such independent progress, we assume that ϕ_{1-i} admits an *idling* transition (q, q, τ), where τ is the silent label over $P(\phi_{1-i})$. The *silent label* over a set of ports $P \subseteq X$ is the map $\tau : P \longrightarrow D$ that maps $x \in P$ to $* \in D$. If such idling transitions are available in every state of ϕ_1, we say that ϕ_1 is *reflexive*:

Definition 5.4.1 (Reflexive)**.** A stream constraint ϕ is reflexive if and only if $q \xrightarrow{\tau} q$ in $[\![\phi]\!]$, for all $q \in Q(\phi)$.

For regular constraints, we can define reflexiveness also syntactically, for which we need some notation. For a variable $x \in X$ and an integer $k \in \mathbb{N} \cup \{-1\}$, we define the predicate $x\dagger_k$ (pronounced: "x is blocked at step k") as follows:

$$x\dagger_k \;:=\; (x^{(k)} \doteq x^{(k-1)}), \qquad \text{with } x^{(k)} \doteq *, \text{ for all } k < 0.$$

Predicate $x\dagger_{-1} \equiv \top$ is trivially true. Predicate $x\dagger_0 \equiv (x \doteq *)$ means that we observe no data flow at port x. Predicate $x\dagger_1 \equiv (x' \doteq x)$ means that the data in memory variable x remains the same.

We now provide a syntactic equivalent of Definition 5.4.1 for regular constraints.

Lemma 5.4.2. *A regular stream constraint $\phi = \psi_0 \wedge \Box\psi$ is reflexive if and only if $\bigwedge_{x \in X} x\dagger_{d(x)} \models \psi$, where $d(x) = \deg_x(\phi)$, for all $x \in X$.*

Proof. Since $d(x) = -1$, for all but finitely many $x \in X$, the stream constraint $\bigwedge_{x \in X} x\dagger_{d(x)}$ is well-defined. By definition, $\bigwedge_{x \in X} x\dagger_{d(x)} \models \psi$ if and only if, for all $q \in Q(\phi)$, there exists some $\theta \in \mathcal{L}(\psi)$, such that $q_\theta = q_{\theta'} = q$ and $\alpha_\theta = \tau$. □

Example 5.4.1. The stream constraint $\mathsf{Sync}(a, b) := \Box(a \doteq b)$ from Example 5.2.1 is reflexive, because $\bigwedge_{x \in X} x\dagger_{d(x)} = a \doteq * \wedge b \doteq *$ implies $a \doteq b$. The stream constraint FIFO from Example 5.2.3 is reflexive, because $\bigwedge_{x \in X} x\dagger_{d(x)} = a \doteq * \wedge b \doteq * \wedge m' \doteq m$ is one of the clauses of FIFO. ◊

Theorem 5.4.1 suggests a composition operator $\times$ on labeled transition systems, satisfying $[\![\phi_0]\!] \times [\![\phi_1]\!] = [\![\phi_0 \wedge \phi_1]\!]$. For reflexive constraints ϕ_0 and ϕ_1, composition $\times$ simulates composition of constraint automata [BSAR06]. Constraint automata also feature a hiding operator that naturally corresponds to existential quantification $\exists$ for stream constraints. We leave a full formal comparison between stream constraints and constraint automata as future work.

5.5 Rule-based form

The commandification algorithm developed by Jongmans accepts only constraints without disjunction (i.e., conjunctions of literals) [JA16b]. To apply commandification to the invariant ψ of an arbitrary regular stream constraint $\psi_0 \wedge \Box\psi$, we can first transform ψ into disjunctive normal form (DNF). However, the number of clauses in the disjunctive normal form may be exponential in the length of the constraint. In this section, we introduce an alternative to the disjunctive normal form that prevents such exponential blow up, for a strictly larger class of stream constraints. Our main observation is that the clauses of the disjunctive normal form may contain many symmetries, in the sense that we may generate all clauses from a set of stream constraints R, called a *set of rules*. A *rule* is a stream constraint ρ, such that $\deg(\rho) \leq 1$ and $\Box \notin \rho$.

Definition 5.5.1 (Rule-based form)**.** A reflexive stream constraint ϕ is in rule-based form iff ϕ equals

$$\mathrm{rbf}(R) \;=\; \bigwedge_{x \in \mathrm{free}(R)} \left(x\dagger_{d(x)} \vee \bigvee_{\rho \in R : x \in \mathrm{free}(\rho)} \rho \right) \tag{5.4}$$

with R a finite set of rules, $\mathrm{free}(R) = \bigcup_{\rho \in R} \mathrm{free}(\rho)$, and $d(x) = \max_{\rho \in R} \deg_x(\rho)$. A stream constraint ϕ is defined by R iff $\phi \simeq \mathrm{rbf}(R)$.

We provide some intuition behind Definition 5.5.1. For a variable x, there are two possibilities:

1. Nothing happens at x (i.e., $x\dagger_{d(x)}$). For a port variable ($d(x) = 0$) this means that we do not observe any data ($x \doteq *$). For a memory variable ($d(x) = 1$) this means that the data does not change in ($x' \doteq x$).

2. Something happens at x (i.e., $x\dagger_{d(x)}$ does not hold). Then, the rule-based form states that (at least) one of the rules with x as a free variable must hold (and this rule explains what happens at x).

Both possibilities are captured by Equation (5.4).

We apply the rule-based form to the invariant of regular constraints, via $\psi_0 \wedge \Box\,\mathrm{rbf}(R)$, for some degree zero stream constraint ψ_0 and set of rules R. Intuitively, R remains smaller than the DNF of $\mathrm{rbf}(R)$ under composition.

Example 5.5.1. Let ψ be a reflexive stream constraint, with $\deg(\psi) \leq 1$ and $\Box \notin \psi$. By Definition 5.5.1 and Lemma 5.4.2 and the distributive law, we have

$$\mathrm{rbf}(\{\psi\}) \;=\; \bigwedge_{x \in \mathrm{free}(\{\psi\})} \left(x\dagger_{\deg_x(\psi)} \vee \psi \right) \;\equiv\; \left(\bigwedge_{x \in \mathrm{free}(\{\psi\})} x\dagger_{\deg_x(\psi)} \right) \vee \psi \;\equiv\; \psi$$

Now, for $\psi = (a \doteq b)$, we get from Example 5.4.1 that $\mathsf{Sync}(a,b) = \Box(a \doteq b) \equiv \Box\,\mathrm{rbf}(\{a \doteq b\})$, which shows the Sync from Example 5.2.1 can be expressed in rule-based form. ◊

Example 5.5.2. The stream constraint $\mathsf{LossySync}(a,b) := \Box\,\mathrm{rbf}(\{a \doteq a, a \doteq b\})$ is equivalent to $\Box(b \doteq * \vee a \doteq b)$. Note that $\Box\,\mathrm{rbf}(\{\top, a \doteq b\}) \simeq \Box\,\mathrm{rbf}(\{a \doteq b\}) \simeq \mathsf{Sync}(a,b)$. Hence, rules $a \doteq a$ and $\top$ are different, because they have different sets of free variables. ◊

Example 5.5.3. The set of rules that define a stream constraint is not unique. Consider the stream constraint FIFO from Example 5.2.3. On the one hand, we have $\mathsf{FIFO}(a,b,m) \simeq m \doteq * \wedge \Box\,\mathrm{rbf}(\{\varphi,\psi\})$, where $\varphi \simeq a \doteq m' \doteq \mathbf{0} \wedge m \doteq *$ models the action that puts data in the buffer and $\psi \simeq m' \doteq * \wedge b \doteq m \doteq \mathbf{0}$ models the action that takes data out of the buffer. On the other hand, we have $\mathsf{FIFO}(a,b,m) \simeq m \doteq * \wedge \Box\,\mathrm{rbf}(\{a \doteq m' \doteq \mathbf{0} \wedge b \doteq m \doteq *, a \doteq m' \doteq * \wedge b \doteq m \doteq \mathbf{0}\})$. ◊

Example 5.5.4. Rule-based forms are an alternative to disjunctive normal forms. Consider the reflexive constraint $\phi := \bigvee_{i=1}^{n} \rho_i$ in DNF for which the first conjunctive clause ρ_1 is equivalent to $\bigwedge_{x \in \mathrm{free}(\phi)} x\dagger_{d(x)}$, with $d(x) = \deg_x(\phi)$. By adding equalities of the form $x \doteq x$, we assume without loss of generality that $\mathrm{free}(\rho_i) = \mathrm{free}(\phi)$, for all $2 \leq i \leq n$. For $R = \{\rho_i \mid 2 \leq i \leq n\}$, it follows from

$$\mathrm{rbf}(R) \;\equiv\; \bigwedge_{x \in \mathrm{free}(R)} \left(x\dagger_{d(x)} \vee \bigvee_{\rho \in R} \rho \right) \;\equiv\; \left(\bigwedge_{x \in \mathrm{free}(\phi)} x\dagger_{d(x)} \right) \vee \bigvee_{\rho \in R} \rho \;\equiv\; \phi \quad (5.5)$$

that ϕ is defined by the set R. We therefore conclude that every reflexive constraint can be written in rule-based form. ◊

Definition 5.5.1 presents the rule-based form as a conjunctive normal form. The following result computes the disjunctive normal form of $\mathrm{rbf}(R)$.

Lemma 5.5.1. *For every set of rules R, we have*

$$\mathrm{rbf}(R) \;\simeq\; \mathrm{dnf}(R) \;:=\; \bigvee_{T \subseteq R} \bigwedge_{\rho \in T} \rho \wedge \bigwedge_{x \in \mathrm{free}(R) \setminus \mathrm{free}(T)} x\dagger_{d(x)}.$$

Proof. Let $x \in X$ be arbitrary. By construction, $\deg_x(\mathrm{dnf}(R)) \leq \max_{\rho \in R} \deg_x(\rho)$. Since $d(x) = \max_{\rho \in R} \deg_x(\rho)$, the clause for $T = \emptyset$ shows that $\deg_x(\mathrm{dnf}(R)) \geq d(x)$. By Lemma 5.6.2, $\deg_x(\mathrm{rbf}(R)) = \deg_x(\mathrm{dnf}(R))$, for all $x \in X$. Hence, $\mathrm{free}^k(\mathrm{rbf}(R)) = \mathrm{free}^k(\mathrm{dnf}(R))$, for all $k \geq 0$.

Next, we show that $\mathrm{rbf}(R) \models \mathrm{dnf}(R)$. Let $\theta \in \mathcal{L}(\mathrm{rbf}(R))$. We find, for every $x \in \mathrm{free}(R)$, some rule $\rho_x \in R$, such that $\theta \models \rho$ and $x \in \mathrm{free}(\rho)$. Now, define $T_\theta := \{\rho_x \mid x \in \mathrm{free}(R) \text{ and } \theta \notin \mathcal{L}(x\dagger_{d(x)})\}$. By construction, $\theta \models \rho_x$, for every $\rho_x \in T_\theta$. If $x \in \mathrm{free}(R)$ and $\theta \notin \mathcal{L}(x\dagger_{d(x)})$, then $\rho_x \in T_\theta$ and $x \in \mathrm{free}(\rho_x) \subseteq \mathrm{free}(T_\theta)$. By contraposition, we conclude that $\theta \models x\dagger_{d(x)}$, for all $x \in \mathrm{free}(R) \setminus \mathrm{free}(T_\theta)$. Hence, $\theta \models \mathrm{dnf}(R)$, and $\mathcal{L}(\mathrm{rbf}(R)) \subseteq \mathcal{L}(\mathrm{dnf}(R))$.

Finally, we show that $\mathrm{dnf}(R) \models \mathrm{rbf}(R)$. Let $\theta \in \mathcal{L}(\mathrm{dnf}(R))$. By definition of $\mathrm{dnf}(R)$, we find some $T \subseteq R$ with $\theta \models \rho$, for all $\rho \in T$, and $\theta \models x\dagger_{d(x)}$, for all

$x \in \text{free}(R) \setminus \text{free}(T)$. Suppose that $x \in \text{free}(R)$ and $\theta \not\models x\dagger_{d(x)}$. Since $\theta \models x\dagger_{d(x)}$, for all $x \in \text{free}(R) \setminus \text{free}(T)$, we find by contraposition that $x \in \text{free}(T)$. Hence, we find some $\psi \in T$ with $x \in \text{free}(\psi)$. Since $\theta \models \rho$, for all $\rho \in T$, we find that $\theta \models \psi$. Hence, $\theta \models \text{rbf}(R)$ and we conclude that $\text{rbf}(R) \simeq \text{dnf}(R)$. □

5.6 Composition

We express conjunction of stream constraints in terms of their defining sets of rules. That is, for two sets of rules R_0 and R_1, we define the composition $R_0 \wedge R_1$ of R_0 and R_1, such that $\text{rbf}(R_0 \wedge R_1) \simeq \text{rbf}(R_0) \wedge \text{rbf}(R_1)$. If R_0 and R_1 do not share any variable (i.e., $\text{free}(R_0) \cap \text{free}(R_1) = \emptyset$), composition $R_0 \wedge R_1$ is given by the union $R_0 \cup R_1$. It is not hard to verify that $\text{dnf}(R_0 \cup R_1) \equiv \text{dnf}(R_0) \wedge \text{dnf}(R_1)$, whenever R_0 and R_1 do not share any variable. This result already demonstrates the power of the rule-based form, because the number of rules grows linearly, while the number of clauses is the disjunctive normal form grows exponentially. Recall that we compare the rule-based form with the disjunctive normal form, because Jongmans' commandification algorithm requires conjunctions of literals as input.

Of course, the assumption that R_0 and R_1 do not share any variable is very strong. In this section, we define the composition $R_0 \wedge R_1$ of R_0 and R_1 for $\text{free}(R_0) \cap \text{free}(R_1) \neq \emptyset$. Intuitively, we must find 'small' subsets $S \subseteq R_0 \cup R_1$ of rules that must synchronize (i.e., fire together) as a result of a shared variable. The conjunction of all rules in such a subset S yields a rule in the composition $R_0 \wedge R_1$.

In view of Example 5.5.4, consider the normal form $\text{dnf}(R_0 \wedge R_1)$. Since $\text{dnf}(R_0 \wedge R_1)$ equals $\text{dnf}(R_0) \wedge \text{dnf}(R_1)$, it suffices to characterize the set of clauses of $\text{dnf}(R_0) \wedge \text{dnf}(R_1)$. Every such clause is a conjunction of a clause in $\text{dnf}(R_0)$ and a clause in $\text{dnf}(R_1)$. Lemma 5.5.1 shows that the clauses of $\text{dnf}(R_i)$ correspond to subsets T_i of R_i, for all $i \in \{0, 1\}$. Not every pair of subsets $T_0 \subseteq R_0$ and $T_1 \subseteq R_1$ yields a clause of $\text{dnf}(R_0) \wedge \text{dnf}(R_1)$, but only if $S = T_0 \cup T_1$ is *synchronous*:

Definition 5.6.1 (Synchronous)**.** A synchronous set over sets of rules R_0 and R_1 is a subset $S \subseteq R_0 \cup R_1$, with $\text{free}(S) \cap \text{free}(R_i) \subseteq \text{free}(S \cap R_i)$, for all $i \in \{0, 1\}$.

Example 5.6.1. For any integer $i \geq 1$, let $\varphi_i := a_i \doteq m_i' \doteq \mathbf{0} \wedge m_i \doteq *$ and $\psi_i := m_i' \doteq * \wedge a_{i+1} \doteq m_i \doteq \mathbf{0}$ be the two rules that define $\mathsf{FIFO}(a_i, a_{i+1}, m_i)$, from Example 5.5.3. The synchronous sets consist of exactly those sets $S \subseteq \{\varphi_1, \psi_1\} \cup \{\varphi_2, \psi_2\}$ that satisfy $\psi_1 \in S$ iff $\varphi_2 \in S$. That is, the synchronous sets are given by $\emptyset$, $\{\varphi_1\}$, $\{\psi_2\}$, $\{\psi_1, \varphi_2\}$, $\{\varphi_1, \psi_1, \varphi_2\}$, $\{\psi_1, \varphi_2, \psi_2\}$, $\{\varphi_1, \psi_1, \varphi_2, \psi_2\}$. ◊

Next, we recognize symmetries in the collection of synchronous sets. We can construct every synchronous set as a union of *irreducible* synchronous subsets:

Definition 5.6.2 (Irreducibility)**.** A non-empty synchronous set $\emptyset \neq S \subseteq R_0 \cup R_1$ is irreducible if and only if $S = S_0 \cup S_1$ implies $S = S_0$ or $S = S_1$, for all synchronous subsets $S_0, S_1 \subseteq R_0 \cup R_1$.

Example 5.6.2. Let R_0 and R_1 be sets of rules, and let $\rho \in R_0$ be a rule, such that $\text{free}(\rho) \cap \text{free}(R_1) = \emptyset$. We show that $\{\rho\}$ is irreducible synchronous. Since $\text{free}(\{\rho\}) \cap \text{free}(R_0) = \text{free}(\rho) = \text{free}(\{\rho\} \cap R_0)$ and $\text{free}(\{\rho\}) \cap \text{free}(R_1) = \emptyset \subseteq \text{free}(\{\rho\} \cap R_1)$, we conclude that $\{\rho\}$ is synchronous. Suppose $\{\rho\} = S_0 \cup S_1$.

Then, $\rho \in S_i$, for some $i \in \{0,1\}$. Hence, $\{\rho\} \subseteq S_i \subseteq \{\rho\}$, which shows that $S_i = \{\rho\}$. We conclude that $\{\rho\}$ is irreducible synchronous in $R_0 \cup R_1$. ◊

Example 5.6.3. Consider φ_i and ψ_i, for $i \in \{1,2\}$, from Example 5.6.1. The irreducible synchronous sets of $\{\varphi_1, \psi_1\} \cup \{\varphi_2, \psi_2\}$ are $\{\varphi_1\}$, $\{\psi_2\}$, and $\{\psi_1, \varphi_2\}$. ◊

Definition 5.6.3 (Composition)**.** The composition of sets of rules R_0 and R_1 is $R_0 \wedge R_1 := \{\bigwedge_{\rho \in S} \rho \mid S \subseteq R_0 \cup R_1 \text{ irreducible synchronous}\}$.

Example 5.6.4. Let R_0 and R_1 be sets of rules, with $\text{free}(R_0) \cap \text{free}(R_1) = \emptyset$. By Example 5.6.2, we find that $\{\rho\} \subseteq R_0 \cup R_1$, for all $\rho \in R_0 \cup R_1$, is irreducible synchronous. Hence, every synchronous set $S \subseteq R_0 \cup R_1$, with $|S| \geq 2$, is reducible. Therefore, $S \subseteq R_0 \cup R_1$ is irreducible synchronous if and only if $S = \{\rho\}$, for some $\rho \in R_0 \cup R_1$. We conclude that $R_0 \wedge R_1 = R_0 \cup R_1$. Consequently, $\emptyset$ is a (unique) identity element with respect to composition $\wedge$ of sets of rules. ◊

To show that the composition of sets of rules coincides with conjunction of stream constraints, we need the following result that shows that every non-empty synchronous set can be covered by irreducible synchronous sets.

Lemma 5.6.1. *Let R_0 and R_1 be sets of rules, and let $S \subseteq R_0 \cup R_1$ be a non-empty synchronous set. Then, $S = \bigcup_{i=1}^{n} S_i$, where $S_i \subseteq R_0 \cup R_1$, for $1 \leq i \leq n$, is irreducible synchronous.*

Proof. We prove the lemma by induction on the size $|S|$ of S. For the base case, suppose that $|S| = 1$. We show that S is irreducible synchronous, which provides a trivial covering. Suppose that $S = S_0 \cup S_1$, for some synchronous sets $S_0, S_1 \subseteq R_0 \cup R_1$. Since, $|S| = 1$, we have $S \subseteq S_i \subseteq S$, for some $i \in \{0,1\}$. Hence, $S = S_i$, and S is irreducible. We conclude that the lemma holds, for $|S| = 1$.

For the induction step, suppose that $|S| = k > 1$, and suppose that the lemma holds, for $|S| < k$. If S is irreducible, we find a trivial covering of S. If S is reducible, we find $S = S_0 \cup S_1$, where $S_0 \neq S \neq S_1$ are synchronous sets in $R_0 \cup R_1$. Since $|S_i| < |S|$, for $i \in \{0,1\}$, we find by the hypothesis that $S_i = \bigcup_{j=1}^{n_i} S_{ij}$. Hence, $S = S_0 \cup S_1 = \bigcup_{i=0}^{1} \bigcup_{j=1}^{n_i} S_{ij}$. We conclude that the lemma holds, for $|S| = k$. By induction on $|S|$, we conclude the lemma. □

Lemma 5.6.2. $\deg_x(\text{rbf}(R)) = \max_{\rho \in R} \deg_x(\rho)$*, for all sets of rules R and $x \in X$.*

Proof. For any set of rules R and $y \in X$, we have

$$\deg_y(\text{rbf}(R)) = \max_{x \in \text{free}(R)} \max(\deg_y(x\dagger_{d(x)}), \max_{\rho \in R : x \in \text{free}(\rho)} \deg_y(\rho)).$$

Note that $\deg_y(x\dagger_{d(x)}) = d(y)$, if $y = x$, and $\deg_y(x\dagger_{d(x)}) = -1$, otherwise. Since $d(y) = \max_{\rho \in R} \deg_y(\rho)$, we have $\deg_y(\text{rbf}(R)) = \max_{\rho \in R} \deg_y(\rho)$. □

Theorem 5.6.3. $\text{rbf}(R_0 \wedge R_1) \simeq \text{rbf}(R_0) \wedge \text{rbf}(R_1)$*, for all sets of rules R_0 and R_1.*

Proof. By Lemma 5.6.2 and Definition 5.6.3, $\deg_x(\text{rbf}(R_0 \wedge R_1)) = \deg_x(\text{rbf}(R_0) \wedge \text{rbf}(R_1))$, for all $x \in X$. Hence, $\text{free}^k(\text{rbf}(R_0 \wedge R_1)) = \text{free}^k(\text{rbf}(R_0) \wedge \text{rbf}(R_1))$, for all $k \geq 0$.

Next, we show $\mathrm{rbf}(R_0) \wedge \mathrm{rbf}(R_1)) \models \mathrm{rbf}(R_0 \wedge R_1)$. Let $\theta \in \mathcal{L}(\mathrm{rbf}(R_0) \wedge \mathrm{rbf}(R_1))$. By Definition 5.5.1, we must show that for every $x \in \mathrm{free}(R_0 \wedge R_1)$ there exists some $\rho_x \in R_0 \wedge R_1$ such that $x \in \mathrm{free}(\rho_x)$ and either $\theta \models x\dagger_{d(x)}$ or $\theta \models \rho_x$. Hence, suppose that $\theta \notin \mathcal{L}(x\dagger_{d(x)})$, for some variable $x \in \mathrm{free}(R_0 \wedge R_1)$. Since $\mathrm{free}(R_0 \wedge R_1) = \mathrm{free}(R_0) \cup \mathrm{free}(R_1)$ and $\theta \models \mathrm{free}(R_0) \wedge \mathrm{free}(R_1)$, we find from Definition 5.5.1 some $\psi \in R_0 \cup R_1$, with $\theta \models \psi$ and $x \in \mathrm{free}(\psi)$. We now show that there exists an irreducible synchronous set $S \subseteq R_0 \cup R_1$, such that, for $\rho_x := \bigwedge_{\rho \in S} \rho$, we have $\theta \models \rho_x$ and $x \in \mathrm{free}(\rho_x)$. By repeated application of Definition 5.6.1, we construct a finite sequence

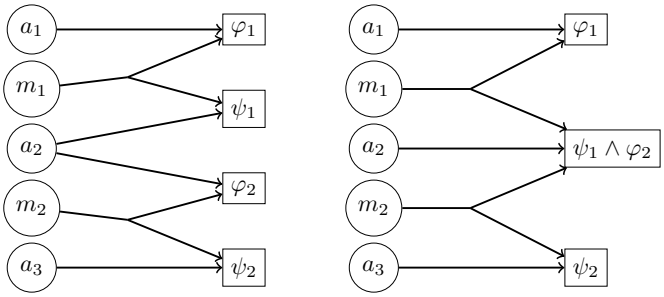
$$\{\psi\} = S_0 \subsetneq \cdots \subsetneq S_n,$$

such that $S_n \subseteq R_0 \cup R_1$ is synchronous, and $\theta \models \bigwedge_{\rho \in S_n} \rho$. Suppose $S_k \subseteq R_0 \cup R_1$, for $k \geq 1$, is not synchronous. By Definition 5.6.1, there exists some $i \in \{0, 1\}$ and a variable $x \in \mathrm{free}(S_k) \cap \mathrm{free}(R_i)$, such that $x \notin \mathrm{free}(S_k \cap R_i)$. Since $x \in \mathrm{free}(R_i)$, we have $R_i^x := \{\rho \in R_i \mid x \in \mathrm{free}(\rho)\} \neq \emptyset$. Since $\theta \models \mathrm{rbf}(R_i)$, there exists some $\psi_k \in R_i^x$ such that $\theta \models \psi_k$. Now define $S_{k+1} := S_k \cup \{\psi_k\}$. Since $x \notin \mathrm{free}(S_k \cap R_i)$ and $x \in \mathrm{free}(S_{k+1} \cap R_i)$, we have a strict inclusion $S_k \subsetneq S_{k+1}$. Due to these strict inclusions, we have, for $k \geq |R_0 \cup R_1|$, that $S_k = R_0 \cup R_1$, which is trivially synchronous in $R_0 \cup R_1$. Therefore, our sequence $S_0 \subsetneq \cdots$ of inclusions terminates, from which we conclude the existence of S_n. By Lemma 5.6.1, we find some irreducible synchronous set $S \subseteq S_n$, such that $\psi \in S$. We conclude that $\rho_x := \bigwedge_{\rho \in S} \rho \in R_0 \wedge R_1$ satisfies $\theta \models \rho_x$ and $x \in \mathrm{free}(\psi) \subseteq \mathrm{free}(S) = \mathrm{free}(\rho_x)$. By Definition 5.5.1, we have $\theta \models \mathrm{rbf}(R_0 \wedge R_1)$, and $\mathrm{rbf}(R_0) \wedge \mathrm{rbf}(R_1) \models \mathrm{rbf}(R_0 \wedge R_1)$.

Finally, we prove that $\mathrm{rbf}(R_0 \wedge R_1) \models \mathrm{rbf}(R_0) \wedge \mathrm{rbf}(R_1)$. Let $\theta \in \mathcal{L}(\mathrm{rbf}(R_0 \wedge R_1))$. We show that $\theta \models \mathrm{rbf}(R_i)$, for all $i \in \{0, 1\}$. By Definition 5.5.1, we must show that for every $i \in \{0, 1\}$ and every $x \in \mathrm{free}(R_i)$ there exists some $\rho \in R_i$ such that $x \in \mathrm{free}(\rho)$ and either $\theta \models x\dagger_{d(x)}$ or $\theta \models \rho$. Hence, let $i \in \{0, 1\}$ and $x \in \mathrm{free}(R_i)$ be arbitrary, and suppose that $\theta \notin \mathcal{L}(x\dagger_{d(x)})$. Since $\mathrm{free}(R_i) \subseteq \mathrm{free}(R_0 \wedge R_1)$, it follows from our assumption $\theta \models \mathrm{rbf}(R_0 \wedge R_1)$ that $\theta \models \bigwedge_{\rho \in S} \rho$, for some irreducible synchronous set $S \subseteq R_0 \cup R_1$ satisfying $x \in \mathrm{free}(S)$. Since $S \subseteq R_0 \cup R_1$ synchronous, we find that $x \in \mathrm{free}(S) \cap \mathrm{free}(R_i) = \mathrm{free}(S \cap R_i)$. Hence, we find some $\rho \in S \cap R_i$, such that $\theta \models \rho$ and $x \in \mathrm{free}(\rho)$. By Definition 5.5.1, we conclude that $\theta \models \mathrm{rbf}(R_i)$, for all $i \in \{0, 1\}$. Therefore, $\mathrm{rbf}(R_0 \wedge R_1) \simeq \mathrm{rbf}(R_0) \wedge \mathrm{rbf}(R_1)$. □

Example 5.6.5. Let φ_i and ψ_i, for $i \geq 1$, be the rules from Example 5.6.1. By Example 5.6.3, the composition $\mathsf{FIFO}_2 := \bigwedge_{i=1}^{2} \mathsf{FIFO}(a_i, a_{i+1}, m_i)$ is defined by the set of rules $\{\varphi_1, \psi_1 \wedge \varphi_2, \psi_2\}$.[2] To compute a set of rules that defines the composition, it is not efficient to enumerate all (exponentially many) synchronous subsets of $R_0 \cup R_1$ and remove all reducible sets. Our tools use an algorithm based on hypergraph transformations to compute the irreducible synchronous sets. Although is would certainly be possible to offer the details of this algorithm here, we postpone the description of such an algorithm until Section 6.3.1. The reason

[2]The rules for the composition of two FIFO stream constraints has striking similarities with *synchronous region* decomposition developed by Proença et al. [PCdVA12]. Indeed, φ_1, $\psi_1 \wedge \varphi_2$, and ψ_2 correspond to the synchronous regions in the composition of two buffers. Therefore, rule-based composition generalizes synchronous region decomposition that has been used as a basis for generation of parallel code [JA18].



(a) $\bigwedge_{i=1}^{2} \mathrm{rbf}(\{\varphi_i, \psi_i\})$ (b) $\mathrm{rbf}(\bigwedge_{i=1}^{2} \{\varphi_i, \psi_i\})$

Figure 5.2: Hypergraph representations of $\bigwedge_{i=1}^{2} \mathsf{FIFO}(a_i, a_{i+1}, m_i)$.

is that the composition operator does not depend on the particular details of the syntax of stream constraints. Indeed, knowing which rules share a variable is the only relevant information for composition. This is precisely the information that is available in multilabeled Petri nets (introduced in Chapter 6), which can be viewed as a data-agnostic abstraction of stream constraints.

Figure 5.2 shows a graphical representation of composition FIFO_2, using hypergraphs. These hypergraphs consist of sets of hyperedges (x, F), where x is a variable and F is a set of rules. Each hyperedge (x, F) in a hypergraph corresponds to a disjunction $x\dagger_{d(x)} \vee \bigvee_{\rho \in F} \rho$ of the rule-based form in Definition 5.5.1. $\Diamond$

5.7 Complexity

In the worst case, composition $R_0 \wedge R_1$ of arbitrary sets of rules R_0 and R_1 may consists of $|R_0| \times |R_1|$ rules. However, if R_0 and R_1 are simple, the size of the composition is bounded by $|R_0| + |R_1|$.

Recall that $P(\phi) = \mathrm{free}^0(\phi)$ is the set of port variables of a stream constraint ϕ.

Definition 5.7.1 (Simple)**.** A set R of rules is simple if and only if $\mathrm{free}(\rho) \cap \mathrm{free}(\rho') \cap P(\mathrm{rbf}(R)) \neq \emptyset$ implies $\rho = \rho'$, for every $\rho, \rho' \in R$.

In other words, a set of rules R is simple if no two (distinct) rules share a port variable. This implies that the dataflow through each port variable is governed by exactly one rule.

Not every stream constraint can be represented by a simple set of rules. For example, a binary exclusive router (Figure 2.5(a) and Example 2.1.7) requires two rules that govern dataflow through its input port A: one rule that routes data from port A to port B and one rule that routes data from port A to port B'. Since both rules share port A, the set of rules is not simple.

Example 5.7.1. By Example 5.5.3, the invariant of $\mathsf{FIFO}(a, b, m)$ is defined by $R := \{a \doteq m' \doteq \mathbf{0} \wedge m \doteq *, m' \doteq * \wedge b \doteq m \doteq \mathbf{0}\}$ as well as $R' := \{a \doteq m' \doteq \mathbf{0} \wedge b \doteq m \doteq *, a \doteq m' \doteq * \wedge b \doteq m \doteq \mathbf{0}\}$. The set R is simple, while R' is not. $\Diamond$

Lemma 5.7.1. *Let R_0 and R_1 be sets of rules, such that* $\mathrm{free}(R_0) \cap \mathrm{free}(R_1) \subseteq P(\mathrm{rbf}(R_0 \cup R_1))$*, and let $S \subseteq R_0 \cup R_1$ be synchronous. Let G_S be a graph with vertices S and edges $E_S = \{(\rho, \rho') \in S^2 \mid \mathrm{free}(\rho) \cap \mathrm{free}(\rho') \cap P(\mathrm{rbf}(R_0 \cup R_1)) \neq \emptyset\}$. If S irreducible, then G_S is connected.*

Proof. Suppose that G_S is disconnected. We find $\emptyset \neq S_0, S_1 \subseteq S$, with $S_0 \cup S_1 = S$, $S_0 \cap S_1 = \emptyset$ and $\mathrm{free}(S_0) \cap \mathrm{free}(S_1) \cap P(\mathrm{rbf}(R_0 \cup R_1)) = \emptyset$. We show that S_0 and S_1 are synchronous. Let $i, j \in \{0, 1\}$ and $x \in \mathrm{free}(S_i) \cap \mathrm{free}(R_j)$. We distinguish two cases:

Case 1 ($x \in \mathrm{free}(R_{1-j})$): Then, $x \in \mathrm{free}(R_0) \cap \mathrm{free}(R_1) \subseteq P(\mathrm{rbf}(R_0 \cup R_1))$. Since $\mathrm{free}(S_0) \cap \mathrm{free}(S_1) \cap P(\mathrm{rbf}(R_0 \cup R_1)) = \emptyset$, we have $x \notin \mathrm{free}(S_{1-i})$. Since S is synchronous, we have $x \in \mathrm{free}(S_i) \cap \mathrm{free}(R_j) \subseteq \mathrm{free}(S) \cap \mathrm{free}(R_j) \subseteq \mathrm{free}(S \cap R_j)$. Hence, we find some $\rho \in S \cap R_j$, with $x \in \mathrm{free}(\rho)$. Since $x \notin \mathrm{free}(S_{1-i})$, we conclude that $\rho \in S_i \cap R_j$. Thus, $x \in \mathrm{free}(S_i \cap R_j)$, if $x \in \mathrm{free}(R_{1-j})$.

Case 2 ($x \notin \mathrm{free}(R_{1-j})$): Since $x \in \mathrm{free}(S_i)$, we find some $\rho \in S_i$, with $x \in \mathrm{free}(\rho)$. Since $x \notin \mathrm{free}(R_{1-j})$, we conclude that $\rho \in R_j$. Hence, $x \in \mathrm{free}(\rho) \subseteq \mathrm{free}(S_i \cap R_j)$, if $x \notin \mathrm{free}(R_{1-j})$.

We conclude in both cases that $x \in \mathrm{free}(\rho) \subseteq \mathrm{free}(S_i \cap R_j)$. Hence, $\mathrm{free}(S_i) \cap \mathrm{free}(R_j) \subseteq \mathrm{free}(S_i \cap R_j)$, for all $i, j \in \{0, 1\}$, and we conclude that S_0 and S_1 are synchronous. Since $S_0 \neq S \neq S_1$, we conclude that S is reducible. By contraposition, we conclude that G_S is connected, whenever S is irreducible. □

Lemma 5.7.2. *Let R_0 and R_1 be simple sets of rules, with* $\mathrm{free}(R_0) \cap \mathrm{free}(R_1) \subseteq P(\mathrm{rbf}(R_0 \cup R_1))$*, and let $S_0, S_1 \subseteq R_0 \cup R_1$ be irreducible synchronous. If $S_0 \cap S_1 \neq \emptyset$, then $S_0 = S_1$.*

Proof. Suppose that $S_0 \cap S_1 \neq \emptyset$. Then, there exists some $\rho_0 \in S_0 \cap S_1$. We show that $S_i \subseteq S_{1-i}$, for all $i \in \{0, 1\}$. Let $i \in \{0, 1\}$, and $\rho \in S_i$. By Lemma 5.7.1, we find an undirected path in G_{S_i} from ρ_0 to ρ. That is, we find a sequence $\rho_0 \rho_1 \cdots \rho_n \in S^*$, such that $\rho_n = \rho$ and $(\rho_i, \rho_{i+1}) \in E_{S_i}$, for all $0 \leq i < n$. We show by induction on $n \geq 0$, that $\rho_n \in S_{1-i}$. For the base case ($n = 0$), observe that $\rho_n = \rho_0 \in S_0 \cap S_1 \subseteq S_{1-i}$. For the induction step, suppose that $\rho_n \in S_{1-i}$. By construction of G_{S_i}, we find that $\mathrm{free}(\rho_n) \cap \mathrm{free}(\rho_{n+1}) \cap P_{01} \neq \emptyset$, where $P_{01} = P(\mathrm{rbf}(R_0 \cup R_1))$. Let $j \in \{0, 1\}$, such that $\rho_{n+1} \in R_j$. Since $\rho_n \in S_{1-i}$ and S_{1-i} is synchronous, we have $\emptyset \neq \mathrm{free}(S_{1-i}) \cap \mathrm{free}(R_j) \cap P_{01} = \mathrm{free}(S_{1-i} \cap R_j) \cap P_{01}$. We find some $\rho' \in S_{1-j} \cap R_j$, with $\mathrm{free}(\rho_{n+1}) \cap \mathrm{free}(\rho') \cap P_{01} \neq \emptyset$. Since R_j is simple, we have $\rho_{n+1} = \rho' \in S_{1-i}$, which concludes the proof by induction. It follows from $\rho_n \in S_{1-i}$ that $S_i \subseteq S_{1-i}$, for all $i \in \{0, 1\}$, that is, $S_0 = S_1$. □

As seen in Lemma 5.5.1, the number of clauses in the disjunctive normal form $\mathrm{dnf}(R_0 \wedge R_1)$ can be exponential in the number of rules $|R_0 \wedge R_1|$ of the composition of R_0 and R_1. However, the following (main) theorem shows the number of rules required to define $\bigwedge_i \phi_i$ is only linear in k.

Theorem 5.7.3. *If R_0 and R_1 are simple sets of rules, and* $\mathrm{free}(R_0) \cap \mathrm{free}(R_1) \subseteq P(\mathrm{rbf}(R_0 \cup R_1))$*, then $R_0 \wedge R_1$ is simple and $|R_0 \wedge R_1| \leq |R_0| + |R_1|$.*

Proof. From Lemmas 5.6.1 and 5.7.2, we find that the irreducible synchronous subsets partition $R_0 \cup R_1$. We conclude that $|R_0 \wedge R_1| \leq |R_0| + |R_1|$. We now show that $R_0 \wedge R_1$ is simple. Let ρ_0 and ρ_1 be rules in $R_0 \wedge R_1$, with $\mathrm{free}(\rho_0) \cap$

$\mathrm{free}(\rho_1) \cap P_{01} \neq \emptyset$, where $P_{01} = P(\mathrm{rbf}(R_0 \cup R_1))$. By Definition 5.6.3, we find, for all $i \in \{0,1\}$, an irreducible synchronous set S_i, such that $\rho_i = \bigwedge_{\psi \in S_i} \psi$. Since $\mathrm{free}(\rho_0) \cap \mathrm{free}(\rho_1) \cap P_{01} \neq \emptyset$ and $\mathrm{free}(\rho_i) = \mathrm{free}(S_i)$, for all $i \in \{0,1\}$, we find some $x \in \mathrm{free}(S_0) \cap \mathrm{free}(S_1) \cap P_{01}$. Suppose that $x \in \mathrm{free}(R_j)$, for some $j \in \{0,1\}$. Since S_0 and S_1 are synchronous sets, we have $x \in \mathrm{free}(S_i) \cap \mathrm{free}(R_j) \subseteq \mathrm{free}(S_i \cap R_j)$, for all $i \in \{0,1\}$. We find, for all $i \in \{0,1\}$, some $\psi_i \in S_i \cap R_j$, such that $x \in \mathrm{free}(\psi_i)$. Hence, $\mathrm{free}(\psi_0) \cap \mathrm{free}(\psi_1) \cap P_{01} \neq \emptyset$, and since R_j is simple, we conclude that $\psi_0 = \psi_1$. Therefore, $S_0 \cap S_1 \neq \emptyset$, and Lemma 5.7.2 shows that $S_0 = S_1$ and $\rho_0 = \rho_1$. We conclude that $R_0 \wedge R_1$ is simple. □

The number of clauses in the disjunctive normal form of direct compositions of k fifo constraints grows exponentially in k. This typical pattern of a sequence of queues manifests itself in many other constructions, which causes serious scalability problems (cf., the benchmarks for 'Alternator$_k$' in [JKA17, Section 7.2]). However, Theorem 5.7.3 shows that rule-based composition of k fifo constraints does not suffer from scalability issues: by Example 5.7.1, the fifo constraint can be defined by a simple set of rules. The result in Theorem 5.7.3, therefore, promises (exponential) improvement over the classical constraint automaton representation.

Unfortunately, it seems impossible to define any arbitrary stream constraint by a simple set of rules. Therefore, the rule-based form may still blow up for certain stream constraints. It seems, however, possible to recognize even more symmetries (cf., the queue-optimization in [JHA14]) to avoid explosion and obtain comparable compilation and execution performance for these stream constraints.

5.8 Abstraction

We now study how existential quantification of stream constraints operates on its defining set of rules.

Definition 5.8.1 (Abstraction)**.** Hiding a variable x in a set of rules R yields $\exists x R := \{\exists x \rho \mid \rho \in R\}$.

Unfortunately, $\exists x R$ does not always define $\exists x \phi$, for a stream constraint ϕ defined by a set of rules R. The following result shows that $\exists x R$ defines $\exists x \phi$ if and only if $\mathrm{rbf}(\exists x R) \models \exists x\, \mathrm{rbf}(R)$. In this case, we call variable x *hidable* in R.

It is non-trivial to find a defining set of rules for $\exists x \phi$, if x is not hidable in R, and we leave this as future work.

Theorem 5.8.1. *Let R be a set of rules, and let $x \in X$ be a variable. Then, $\exists x\, \mathrm{rbf}(R) \simeq \mathrm{rbf}(\exists x R)$ if and only if $\mathrm{rbf}(\exists x R) \models \exists x\, \mathrm{rbf}(R)$.*

Proof. Trivially, $\exists x\, \mathrm{rbf}(R) \simeq \mathrm{rbf}(\exists x R)$ implies $\mathrm{rbf}(\exists x R) \models \exists x\, \mathrm{rbf}(R)$. Conversely, suppose that $\mathrm{rbf}(\exists x R) \models \exists x\, \mathrm{rbf}(R)$. From Lemma 5.5.1, it follows that $\exists x\, \mathrm{rbf}(R) \equiv \exists x\, \mathrm{dnf}(R)$. Since existential quantification distributes over disjunction and $\exists x \phi \wedge \psi \models \exists x \phi \wedge \exists x \psi$, for all stream constraints ϕ and ψ, we find

$$\exists x\, \mathrm{dnf}(R) \models \bigvee_{S \subseteq R} \bigwedge_{\rho \in S} \exists x \rho \wedge \bigwedge_{x \neq y \in \mathrm{free}(R) \setminus \mathrm{free}(S)} y\dagger_{d(y)} \equiv \mathrm{dnf}(\exists x R).$$

By Lemma 5.5.1, we have $\exists x\,\mathrm{rbf}(R) \models \mathrm{rbf}(\exists x R)$, and by assumption $\exists x\,\mathrm{rbf}(R) \equiv \mathrm{rbf}(\exists x R)$. Using Lemma 5.6.2, we have $\deg_y(\exists x\,\mathrm{rbf}(R)) = \max_{\rho \in R} \deg_y(\exists x \rho) = \deg_y(\mathrm{rbf}(\exists x R))$, for every variable y. We conclude $\exists x\,\mathrm{rbf}(R) \simeq \mathrm{rbf}(\exists x R)$. □

Example 5.8.1. Suppose $Data = \{0, 1\}$, which means that the data domain equals $D = \{0, 1, *\}$. Let $\mathbf{1}$ be the constant stream defined as $\mathbf{1}(n) = 1$, for all $n \in \mathbb{N}$. For $i \in \{0, 1\}$, consider the set of rules $R_i = \{x = x, x = y_i = \mathbf{i}\}$. Observe that $\{x = x, x = y_i = \mathbf{i}\} \subseteq R_0 \cup R_1$ is synchronous, for all $i \in \{0, 1\}$. Hence, $x = y_i = \mathbf{i} \in R_0 \wedge R_1$, for all $i \in \{0, 1\}$. However, for $\theta = [y_0 \mapsto \mathbf{0}, y_1 \mapsto \mathbf{1}]$, we have $\theta \models \bigwedge_{i \in \{0,1\}} \exists x (x = y_i = i)$, while $\exists x \bigwedge_{i \in \{0,1\}} x = y_i = i \equiv \bot$. Thus, variable x is not hidable from $R_0 \wedge R_1$. ◊

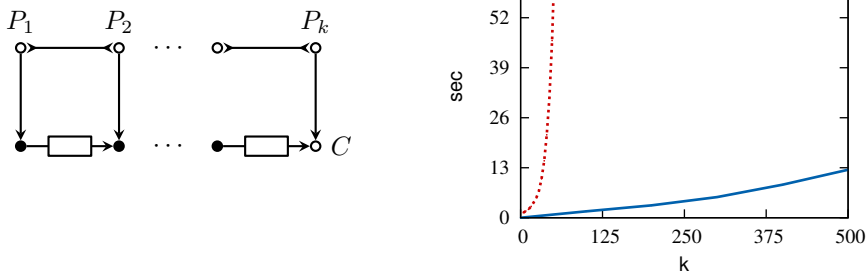
5.9 Application

In on-going work, we applied the rule-based form to compile protocols (in the form of Reo connectors) into executable code. Reo is an exogenous coordination language that models protocols as graph-like structures [Arb04, Arb11]. We recently developed a textual version of Reo, which we use to design non-trivial protocols [DA18b]. An example of such non-trivial protocol is the $\mathsf{Alternator}_k$, where $k \geq 2$ is an integer. Figure 5.3(a) shows a graphical representation of the $\mathsf{Alternator}_k$ protocol.

Intuitively, the behavior of the alternator protocol is as follows: The *nodes* $P_1, \ldots, P_k$ accept data from the environment. Node C offer data to the environment. All other nodes are internal and do not interact with the environment. In the first step of the protocol, the $\mathsf{Alternator}_k$ waits until the environment is ready to offer data at all nodes $P_1, \ldots, P_k$ and is ready to accept data from node C. Only then, the $\mathsf{Alternator}_k$ transfers the data from P_k to C via a synchronous channel, and puts the data from P_i in the i-th fifo channel, for all $i < k$. The behavior of a synchronous channel is defined by the sync stream constraint in Example 5.2.1. Each fifo channel has buffer capacity of one, and its behavior is defined by the fifo stream constraint from Example 5.2.3. In subsequent steps, the environment can one-by-one retrieve the data from the fifo channel buffers, until they are all empty. Then, the protocol cycles back to its initial configuration, and repeats its behavior. For more details on the Reo language and its semantics, we refer to [Arb04, Arb11].

As mentioned in the introduction, Jongmans developed a compiler based on constraint automata [JKA17]. The otherwise stimulating benchmarks presented in [JKA17] show that Jongmans' compiler still suffers from state-space explosion. Figure 5.3(b) shows the compilation time of the $\mathsf{Alternator}_k$ protocol for Jongmans' compiler and ours. Clearly, the compilation time improved drastically and went from exponential in k to almost linear in k.

Every fifo channel in the $\mathsf{Alternator}_k$, except the first, either accepts data from the environment or accepts data from the previous fifo channel. This choice is made by the internal node at the input of each fifo channel. Unfortunately, the behavior of such nodes is not defined in terms of a simple set of rules. Consequently, we cannot readily apply Theorem 5.7.3 to conclude that the number of rules depends only linearly on k. However, it turns out that $\mathsf{Alternator}_k$ can be defined using only k rules: one rule for filling the buffers of all fifo channels, plus $k - 1$ rules, one for



(a) $\mathsf{Alternator}_k$ (b) Compilation times

Figure 5.3: Graphical representation (a) of the $\mathsf{Alternator}_k$ protocol in [JKA17], for $2 \leq k \leq 500$, and its compilation time (b). The dotted red line is produced by the Jongmans' compiler (and corresponds to [JKA17, Fig 11(a)]), and the solid blue line is our compiler.

taking data out of the buffer of each of the $k-1$ fifo channels. This observation explains why our compiler drastically improves upon Jongmans' compiler.

5.10 Discussion

We introduce (regular) stream constraints as an alternative to constraint automata that does not suffer from state space explosions. We define the rule-based form for stream constraints, and we express composition and abstraction of constraints in terms of their rule-based forms. For simple sets of rules, composition of rule-based forms does not suffer from 'transition space explosions' either.

We have experimented with a new compiler for protocols using our rule-based form, which avoids the scalability problems of state- and transition-space explosions of previous automata-based tools. Our approach still leaves the possibility for transition space explosion for non-simple sets of rules. In the future, we intend to study symmetries in stream constraints that are not defined by simple sets of rules. The queue-optimization of Jongmans serves as a good source of inspiration for exploiting symmetries [JHA14].

The results in this chapter are purely theoretical. In on-going work, we show practical implications of our results by developing a compiler based on stream constraints. Such a compiler requires an extension to the current theory on stream constraints: we did not compute the abstraction $\exists x R$ on sets of rules R wherein variable x is not hidable. Example 5.5.4 indicates the existence of situations where we can compute $\exists x R$ even if x is not hidable, a topic which we leave as future work.