Compositional Construction of Real-Time Dataflow Networks

Stephanie Kemper*

CWI, Amsterdam, The Netherlands
S.Kemper@cwi.nl

Abstract. Increasing sizes of present-day distributed software systems call for coordination models which are both *modular* and *scalable*. Precise modelling of real-life applications further requires the notion of *real-time*.

In this paper, we present a modular formal development of a compositional model for real-time coordination in dataflow networks. While real-time dataflow networks are typically asynchronous, our approach includes coordination patterns which combine, but are not limited to, synchrony and asynchrony. We define a constraint- and SAT-based encoding, which allows us to benefit from high-end constraint solving techniques when inspecting valid interactions of the system.

Keywords: Real-Time Dataflow Networks, Component-Based Software Construction, Coordination, Constraint Solving, SAT.

1 Introduction

The size of present-day distributed software systems increases constantly, demanding for adequate scalable methods, which assist developers in constructing large systems, by composing individual components. This in turn creates the need for well-defined formal coordination languages to integrate and, more importantly, orchestrate the distributed system components. Such coordination languages must handle temporal and other nonfunctional interactive properties that cannot be expressed algorithmically [18]. In this paper, we take the view that concurrent interacting software systems are variants of real-time dataflow networks. Distributed software components are components of the network, their timed behaviour is orchestrated by component connectors, such that coordination amounts to composing the behavioural constraints of individual components and their connectors. Traditionally, communication between components and connectors is asynchronous, resembling the idea that connections are realised by unbounded FIFO buffers.

Separation of concerns allows us to consider computation (i.e., concrete data values) and coordination (i.e., presence and absence of dataflow) in real-time dataflow networks separately. In this paper, we focus on the latter; in particular, we do not handle concrete data values. Yet, the integration of these data values is

* Part of this research has been funded by the Dutch BSIK/BRICKS project AFM3.

D. Clarke and G. Agha (Eds.): COORDINATION 2010, LNCS 6116, pp. 92–106, 2010.

straightforward. We extend the untimed asynchronous communication model by adding *timed connectors*, which orchestrate the network behaviour. Connectors consist of a number of distinctly named ports, through which they communicate with the environment (i.e., components), by transmitting data items. The ports are of three different types: *read ports* read data items from the environment, *write ports* write data items to the environment, and *internal ports* (not visible to the environment) transmit data items within the connector. Components are no longer connected to each other, but to a connector, such that communication between components becomes anonymous. The connector imposes a certain coordination pattern on the network, for example by delaying or reordering.

As a second extension of real-time dataflow networks, we take into account environmental constraints, cf. [8]: in traditional approaches, the possible states of ports are *active* (i.e. communicating) and *inactive* (i.e. idle). In this paper, we assume that whenever the internal state of a connector permits communication, it must not refuse communication requests from the environment (e.g., a simple empty buffer can always accept data). Thus, a port in a connector may only be inactive if there is a reason to delay the communication, coming from either the connector or the environment (or both). We therefore split the inactive state of ports into two, based on where the reason for the delay comes from.

In the end, we represent a network as a number of constraints, which can be solved using well-studied constraint solving techniques [2]. A single solution to these constraints corresponds to a valid interaction between a connector and its environment, and the set of all solutions precisely describes the imposed coordination pattern.

1.1 Contributions

The main contributions of this paper can be summarised as follows: we provide a modular framework for compositional construction of a real-time generalisation of dataflow networks. We use a new type of transition systems to describe the behaviour of connectors. We provide a *direct* definition of the behavioural constraints of the connectors in terms of these transition systems, which makes it easy to understand the incorporated coordination pattern. The approach is suitable to model both the "algorithmic" behaviour of connectors (i.e. the internal implementation of the coordination pattern) and the inter-component coordination behaviour. In addition, it can also be used to model the behaviour of the components, such that complete networks can be described with our formalism.

We define a constraint-based encoding of real-time networks, using propositional logic with linear arithmetic. These constraints capture the current state of connectors, how connectors are plugged together, and possible synchronisations with the environment. In this way, we can benefit from existing constraint solving techniques [2] to determine the possible coordination patterns of a connector, or the valid interactions within a network, since these correspond to solutions of the constraints. Starting from the transition systems describing the behaviour, the translation from networks to constraints is fully automatic, such

that new connectors (incorporating new coordination patterns) can be easily introduced simply by defining the underlying transition system.

1.2 Related Work

Dataflow networks have first been defined by Kahn [13], as a set of nodes—containing arbitrary sequential processes—which communicate via unbounded FIFO buffers. All processes need to be deterministic; the straightforward extension of the framework to nondeterministic processes is not compositional anymore [7]. Jonsson [12] showed that to regain compositionality, the model needs to contain information about *all* possible interleavings of *all* communication events in the system (Kahn considers the sequential order on each port in isolation). For this reason, we define the semantics of networks based on traces of LTSs (similar to Jonsson), which contain the necessary information about the possible interleavings. Beyond that, our communication model is more refined, in that we take into account not only presence and absence of dataflow, but in addition require a reason for the absence of dataflow, coming from either the network or the environment. Moreover, all the abovementioned approaches assume *asynchronous* communication over *unbounded FIFOs*. In contrast, we allow both asynchronous and synchronous communication—the latter being needed for global coordination (through synchronisation)—and we allow the nodes to be connected by arbitrary channels, including for example reordering and delay.

Clarke *et al.* [9] and Bonsangue *et al.* [6] present approaches for modelling and verifying connectors in the channel-based coordination language $\mathcal{R}$eo. The basic ideas are similar to ours: [9] represents connectors as constraints, and [6] uses an automata-based formal model, which takes into account environmental constraints by modelling presence and absence of requests. Yet, our framework is more general, in that neither of them considers timing constraints, or distinguishes between input and output ports. In fact, $\mathcal{R}$eo networks and connectors are an untimed subclass of networks in our framework.

The work of Ren and Agha [17] describes real-time systems as compositions of actors (the components), together with timing relations on the occurrence of events between them. The behaviour of the actors is orchestrated by so-called *RTsynchronizers*. These are collections of declarative real-time constraints, which restrict the temporal behaviour of events over groups of actors. The major drawbacks of the approach are that the declarative constraints do not allow to reason about sequential occurrences of events, and therefore are unable to express coordination patterns like for example buffering (for buffer sizes >1) or reordering (while this is possible in our approach). Moreover, RTsynchronizers describe a high-level programming language construct rather than a concrete implementation, so ordinary programming languages first need to (be able to) implement the construct in order to use it.

Kemper and Platzer [14] have presented an encoding of timed automata [1] (TA) in propositional logic with linear arithmetic, the nature of which is close to the constraints defined in this paper. Yet, due to the liberal notion of networks and connectors presented here, and since we take into account environmental

constraints, we may consider TA to be a subclass of those systems covered here. The same is true for earlier work on timed constraint automata [15] (modulo data values, but this is a straightforward extension of the work presented here).

Example (Introduction). Fig. 1 (left side) shows our graphical representation of the external interface of a connector C, with n read ports $r_1, \ldots, r_n$, and m write ports $w_1, \ldots, w_m$. Our running example in the paper is a connector S_q of a timed sequencer, cf. Fig. 1 (right side). The idea of S_q is to cyclically communicate through ports w_1, r_1, w_2, r_2. For components C_i, i=1, 2, connected to S_q via the pair (w_i, r_i), S_q in fact works as a token ring: it offers the token to C_i through w_i, accepts it back through r_i, then offers it to the next component. We assume a timeout for each component, i.e., if a component fails to accept the token in time, S_q skips that component in the current round, and offers the token to the next component. We formalise this internal behaviour in Sect. 2.1, and we compose several instances of S_q to build a larger token ring in Sect. 2.2 (cf. also Fig. 3).

Fig. 1. Graphical Representation of Connector Interfaces

Structure of the Paper. In the next section, we provide the formal definitions for syntax and semantics of real-time components and networks. In Sect. 3, we present our encoding of real-time networks in propositional logic with linear arithmetic, show how to use constraint solving techniques to determine possible coordination patterns, and depict some preliminary experimental results. Finally, Sect. 4 concludes the paper and discusses some directions of future work.

2 Compositional Real-Time Networks

In this section, we present the formal definitions underlying our real-time networks, and illustrate them by means of our running example. In Sect. 2.1, we define simple real-time connectors and how they capture environmental constraints, then we show how to compose these to generate networks and how to propagate the environmental constraints (Sect. 2.2). In Sect. 2.3, we define the semantics of real-time networks by means of transition systems.

2.1 Primitive Connectors

The behaviour of a port, i.e., whether data flows or not, not only depends on the internal state of the connector, but also on the environment in which the connector occurs. In particular, "no dataflow" requires a reason from either the connector or the environment. This in turn means that if both connector

and environment are ready to communicate, then dataflow cannot be delayed. Following the three-colouring idea from [8], we define three different states of ports—called colours—which in the case of no dataflow capture where the reason for delaying the communication comes from.

The *set of possible colours* of a port a is $\{ {}^{a:}\!\!\longrightarrow, {}^{a:}\!\!-!-, {}^{a:}\!\!-?- \}$. A *colouring over a set of ports* P is a function c assigning a colour to each port in P. The set of *all possible colourings of* P is $\mathbb{C}(P)$. The colourings denote *dataflow through* a (${}^{a:}\!\!\longrightarrow$) and *delay on* a, with the underlying connector either providing (${}^{a:}\!\!-!-$) or getting (${}^{a:}\!\!-?-$) a reason for the delay on a. Intuitively, ${}^{a:}\!\!-?-$ means that the connector cannot actively delay dataflow through a, instead, delay requires a reason from the *outside*. On the other hand, ${}^{a:}\!\!-!-$ denotes that the connector *itself* delays the communication. We write colourings in either orientation (e.g. ${}^{a:}\!\!\longrightarrow$ or $\longrightarrow\!\!{}^{:a}$), and we omit port name a if it is clear from the context.

We describe the internal behaviour of connectors by means of labelled transition systems (LTS), which we call *Network Transition Automata* (NTA). We use clock constraints to describe enabling conditions of transitions: clock constraints $\varphi \in \Phi(X)$ over a finite set of real-valued clocks X are conjunctions of `true` and atoms $x \sim c$, with $x \in X$, $c \in \mathbb{Q}$, and $\sim \in \{<, \leq, =, \geq, >\}$. For simplicity, we assume dataflow to be instantaneous. Time may only elapse while the NTA remains in one of its states. Yet, it is straightforward to model duration of data flow, by for example adding a fresh clock and appropriate clock guards >0 on transitions.

Definition 1 (Network Transition Automaton). *An* NTA T over a finite set of ports P *is a tuple* $T=(S, s_0, X, E)$*, with* S *a finite set of states,* s_0 *the initial state,* X *a finite set of real-valued clocks, and* $E \subseteq S \times \Phi(X) \times \mathbb{C}(P) \times 2^X \times S$ *the finite transition relation. An element* $(s, \varphi, c, Y, s') \in E$ *describes a transition from state* s *to state* s'*, enabled under* guard φ*, with dataflow/delay according to colouring* c*, and resetting all clocks in the set* Y*; it is called* delay *iff* $s'=s$*,* $Y=\emptyset$ *and* $c(a) \neq \longrightarrow$ *for all* $a \in P$*, and* communication *otherwise. Two NTAs are called* disjoint *if the respective subsets (i.e, ports, states and clocks) are disjoint.*

A communication (s, φ, Y, c, s') describes conditions on presence/absence of dataflow and on clocks which trigger a state change, while a delay $(s, \varphi, \emptyset, c, s)$ describes the conditions under which T may delay in s, namely, as long as guard φ is satisfied, and a reason for delay exists which satisfies colouring c.

Definition 2 (Connector). *A* connector C *is a tuple* $C=(P^r, P^w, T)$*, with* T *an NTA over a set of ports* P*,* $P^r \subseteq P$ *and* $P^w \subseteq P$ *finite disjoint sets of read respectively write ports. The set* $\mathcal{I}_C \stackrel{\text{def}}{=} P^r \cup P^w \neq \emptyset$*, with* $\mathcal{I}_C \subseteq P$*, called* external interface of C*, contains all externally visible ports, while* P *may contain additional internal ports. Two connectors are called* disjoint *if the respective subsets (i.e., ports and NTAs) are disjoint.*

Example (Primitive Connector). Using Def. 2, the S_q connector from Sect. 1 is $S_q=(\{r_1, r_2\}, \{w_1, w_2\}, T)$, with NTA $T=(\{ot_1, wf_1, ot_2, wf_2\}, ot_1, \{x\}, E)$. The details of E—with a deadline of 3 time units on the availability of the token—are shown in Fig. 2. Communications are denoted by solid lines, delays by dashed

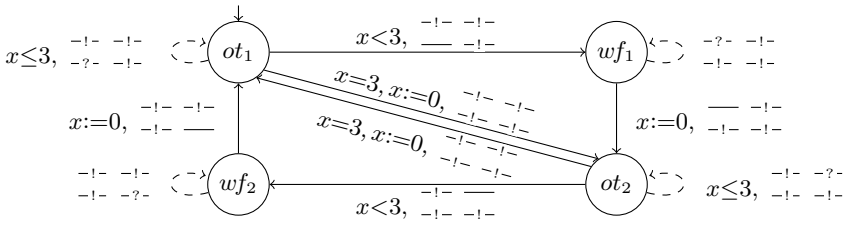


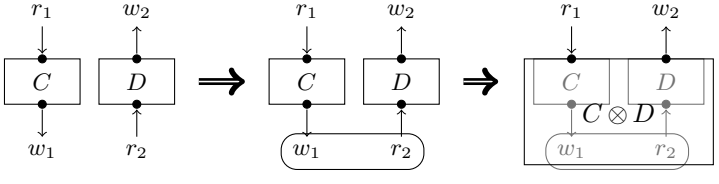
Fig. 2. NTA of the Two-Token Ring Connector

lines. We omit empty clock sets and clock constraints equal to `true`, and we use assignment rather than set notation for clock sets. We arrange the colourings so that they reflect the layout in Fig. 1 (r_1 and w_1 on the left, w_2 and r_2 on the right), and then omit the port names. For explanatory purposes, we assume component C_i is connected to S_q through w_i and r_i, $i = 1, 2$.

S_q starts in ot_1, where it **o**ffers the **t**oken to C_1. It may delay in this state (delay loop) as long as the deadline of 3 time units is not violated and C_1 is not ready to accept the token (w_1 requires a reason). If C_1 accepts the token in time (x<3), S_q moves to wf_1 and **w**aits **f**or C_1 to return the token. Otherwise (x=3), S_q moves directly to ot_2. In wf_1, S_q delays until C_1 returns the token (r_1 requires a reason); then (with dataflow through r_1) moves to ot_2 to offer the token to C_2, thereby resetting clock x to start a new deadline for C_2. The behaviour in ot_2 and wf_2 is symmetric.

2.2 Compositional Construction of Networks

We define composition of connectors by joining sets of (read and write) ports, which yields "invisible" *internal ports*. This is depicted as



Joining write port w_1 (from C) with read port r_2 (from D) yields a new connector $C \otimes D$ with a read port r_1, a write port w_2, and an invisible internal port. For a set of ports P, we call a set $P' \subseteq P$ of ports intended to be joined a *merge set (over P)*, the resulting internal port is denoted as $p_{\prec P'}$. Ports in P' become invisible to the environment (i.e., are removed from the external interface).

The intended behaviour of internal ports is to act as self-contained, stateless "pumping stations" [4], *merging* data from write ports, and *replicating* data to read ports. If data flows, then it flows through *exactly one* write port and through *all* read ports. Absence of dataflow is subject to environmental constraints on the involved ports: if there is a reason for delay (**−!−**) on at least one read port

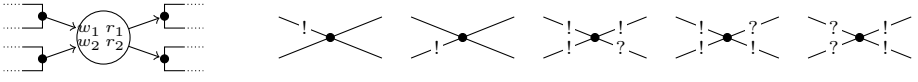
or on all write ports, data cannot flow. Stated differently, a valid colouring of an internal port must not involve the colour $-?-$ only. While other approaches restrict composition to one-to-one relations [3,8,9], we do not impose any restriction on the number, type (read/write) or origin (which connector) of ports in a merge set; the only condition is that merge sets are pairwise disjoint. Though the composition of colourings would be slightly simpler, our many-to-many composition provides a direct and more intuitive way of specifying compositions for for example mergers, replicators or multi-synchronisations.

A colouring $c \in \mathbb{C}(P)$ is *valid over a merge set P'/an internal port $p_{\prec P'}$* if it satisfies the following conditions for all ports $p, p', q \in P'$ (where P^r and P^w are the subsets of P of read respectively write ports):

1. If $\exists p \in P^w : c(p) = ——$, then $\forall q \in P^r : c(q) = ——$, and $\forall p' \in P^w, p' \neq p : c(p') \neq ——$
2. If $\exists q \in P^r : c(q) = ——$, then $\exists p \in P^w : c(p) = ——$
3. If $\nexists p \in P' : c(p) = ——$, then $((\forall p' \in P^w : c(p') = -!-)$ or $(\exists q \in P^r : c(q) = -!-))$

Only valid colourings correctly model the aforementioned behaviour: conditions 1 and 2 describe simultaneous dataflow through exactly one write port and all read ports of $p_{\prec P'}$. Condition 3 describes the propagation of environmental constraints (delays): no dataflow is possible only if either all write ports or at least one read port in P' provide a reason to delay.

The *flip rule* [8] is used to reduce the size of NTAs by identifying redundant (with respect to compositionality) colourings. If for some set of ports P and a port $p \in P$, two colourings $c_1 \in \mathbb{C}(P)$ and $c_2 \in \mathbb{C}(P)$ are identical except for $c_1(p) = -!-$ and $c_2(p) = -?-$, then c_2 is redundant and can be removed: the set of colourings with which c_2 can compose over p is a strict subset of the set of colourings with which c_1 can compose over p. The valid colourings (after applying the flip rule) of an internal port $p_{\prec\{r_1,r_2,w_1,w_2\}}$ are given by



(for clarity, the internal port is conceptually depicted on the left). The colouring on the right, for example, can be read as follows: if all write ports get a reason $(-?-)$, the reason propagates to the read ports, which then provide a reason $(-!-)$, and dataflow is not possible.

The behaviour of a composed connector $C \otimes D$ is described by the composition $T_C \bowtie T_D$ of the underlying NTAs. The basic idea is along the same lines as the standard cross product in other automata models. Yet, to ensure the composed NTA correctly models the propagation of reasons for delay on internal ports, we need to ensure that colourings satisfy the above conditions. Colourings in $T_C \bowtie T_D$ are compositions of colourings from T_C and T_D. The *composition $c_1 \cup c_2$ of colourings $c_1 \in \mathbb{C}(P_1)$ and $c_2 \in \mathbb{C}(P_2)$*, with P_1 and P_2 disjoint, is a new colouring $c = c_1 \cup c_2 \in \mathbb{C}(P_1 \dot{\cup} P_2)$, with $c(p) = c_1(p)$ iff $p \in P_1$, and $c(p) = c_2(p)$ iff $p \in P_2$ for all ports $p \in P_1 \cup P_2$.

Definition 3 (NTA Composition). *Let $\mathcal{T} = \{T_1, \ldots, T_k\}$, $k \geq 1$, be a set of disjoint NTA, $T_i = (S_i, s_{0,i}, X_i, E_i)$, $i \leq k$, an NTA over port set P_i, $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$,*

$n \geq 1$, a set of disjoint merge sets over $\bigcup P_i$. The composition of the T_i over Q, *denoted $T_1 \bowtie_Q \ldots \bowtie_Q T_k$ (or simply $\mathcal{T} \bowtie_Q$), is a new NTA $\mathcal{T} \bowtie_Q = (S, s_0, X, E)$ over $P \stackrel{\text{def}}{=} \bigcup P_i$, with $S = \prod S_i$ (Cartesian product), $s_0 = (s_{0,1}, \ldots, s_{0,k})$, $X = \bigcup X_i$, and transitions in E are given by*

$$\frac{(s_1, \varphi_1, c_1, Y_1, s_1') \in E_1, \ldots, (s_k, \varphi_k, c_k, Y_k, s_k') \in E_k, \quad c = c_1 \cup \ldots \cup c_k \text{ valid over } P}{((s_1, \ldots, s_k), \varphi_1 \wedge \ldots \wedge \varphi_k, c, Y_1 \cup \ldots \cup Y_k, (s_1', \ldots, s_k')) \in E}$$

The definition of connector composition is now straightforward. The basic idea is to compose the NTAs, join the sets of read and write ports, and remove the ports in the merge sets from the external interface.

Definition 4 (Connector Composition). *Let $\mathcal{C} = \{C_1, \ldots, C_k\}$, $k \geq 1$, be a set of disjoint connectors, with $C_i = (P_i^r, P_i^w, T_i)$, $i \leq k$. Let $R \stackrel{\text{def}}{=} \bigcup P_i^r$, $W \stackrel{\text{def}}{=} \bigcup P_i^w$, let $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, $n \geq 1$, a set of disjoint merge sets over $R \cup W$. The* composition of the C_i over $\mathcal{Q}$, *denoted $C_1 \otimes_\mathcal{Q} \ldots \otimes_\mathcal{Q} C_k$ (or simply $\mathcal{C} \otimes_\mathcal{Q}$), is a new connector $\mathcal{C} \otimes_\mathcal{Q} = (P^r, P^w, T)$, with $P^r = R \setminus \bigcup Q_i$, $P^w = W \setminus \bigcup Q_i$, and $T = \bigcup T_i \bowtie_\mathcal{Q}$. We call the C_i the* underlying connectors *of $\mathcal{C} \otimes_\mathcal{Q}$.*

Connector composition is commutative and—after applying the flip rule to remove redundant colourings—associative (modulo state names).

Though the ports contained in merge sets are removed from the external interface $\mathcal{I}_C$ during composition, they are still contained in the underlying NTA T. We define the *reduction of $C = (P^r, P^w, T)$ to $\mathcal{I}_C$* , denoted $C_\downarrow$, to be a new connector $C_\downarrow = (P^r, P^w, T_\downarrow)$, where transitions (s, φ, c', Y, s') in $T_\downarrow$ are obtained from transitions (s, φ, c, Y, s') in T by restricting the colourings to the external interface—i.e. $c' = c|_{\mathcal{I}_C}$—and removing duplicates if necessary

Example (Connector Composition). Consider a set $\mathcal{S} = \{S_{q,0}, S_{q,1}, S_{q,2}\}$ of three instances of the S_q connector from Sect. 2.1. The connectors are identical, except that we add an additional index $0, 1, 2$ to all names (ports, states, clocks) to make clear which connector they belong to, and we change the start state of the NTAs of $S_{q,1}$ and $S_{q,2}$ to be $wf_{2,1}$ and $wf_{2,2}$, respectively (to ensure initially only $S_{q,0}$ offers a token). Composing $\mathcal{S}$ over $\mathcal{Q} = \{\{w_{2,0}, r_{2,1}\}, \{w_{2,1}, r_{2,2}\}, \{w_{2,2}, r_{2,0}\}\}$, we create a token ring for mutual exclusion for three components, as depicted in Fig. 3 (left side). The reachable part of the NTA of the resulting connector $\mathcal{S} \otimes_\mathcal{Q}$ is shown in Fig. 4. Again, we omit port names in the colourings, and we

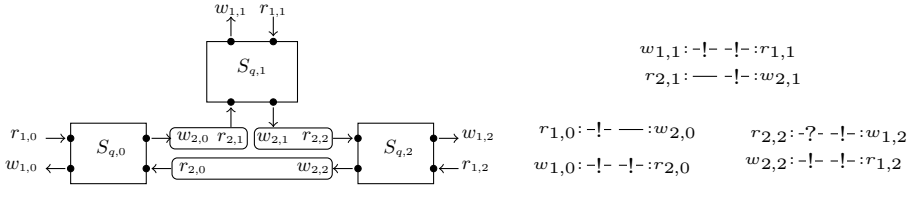


Fig. 3. Connector Composition: Three-Token Ring Connector

arrange the colourings so that they reflect the graphical layout. See the right side of Fig. 3 for an example.

In the initial state, $\mathcal{S}\otimes_{\mathcal{Q}}$ offers the token to the environment through $w_{1,0}$. If the token is taken in time, $\mathcal{S}\otimes_{\mathcal{Q}}$ moves to $(wf_{1,0}wf_{2,1}wf_{2,2})$, where it waits for the token to be returned through $r_{1,0}$. Otherwise (i.e., if the value of x_0 reaches the timeout), $\mathcal{S}\otimes_{\mathcal{Q}}$ moves to state $(ot_{2,0}wf_{2,1}wf_{2,2})$. Note that it is not possible to delay in this state. This is due to the fact that in state $ot_{2,0}$, $S_{q,0}$ offers the token to the environment through $w_{2,0}$, cf. Fig. 2. A delay in $ot_{2,0}$ is only possible if $S_{q,0}$ *gets* a reason to delay on $w_{2,0}$. But $S_{q,1}$, which is connected to $w_{2,0}$ via $r_{2,1}$, never provides a reason for delay on $r_{2,1}$ in state $wf_{2,1}$ (cf. Fig. 2 again). Therefore, the only possible transition from $(ot_{2,0}wf_{2,1}wf_{2,2})$ is to move to $(wf_{2,0}ot_{1,1}wf_{2,2})$, which correctly corresponds to passing the token from $S_{q,0}$ to $S_{q,1}$ without delay. This shows the importance of taking into account environmental constraints, since without these (i.e., when having only one unconstrained "no flow" colour) it would wrongly be possible to delay in $(ot_{2,0}wf_{2,1}wf_{2,2})$. The explanation for the rest of the connector is symmetric. Hiding the three internal ports, that means reducing $\mathcal{S}\otimes_{\mathcal{Q}}$ to $\mathcal{I}_{\mathcal{S}\otimes_{\mathcal{Q}}}$, yields a similar NTA, where blue colourings are removed.

Note that we have removed transitions with unsatisfiable guards. For example, there is a transition from $(ot_{2,0}wf_{2,1}wf_{2,2})$ back to $(ot_{1,0}wf_{2,1}wf_{2,2})$, with guard $x_0{=}3$. But since clock x_0 is reset on both incoming transitions of $(ot_{2,0}wf_{2,1}wf_{2,2})$, and $\mathcal{S}\otimes_{\mathcal{Q}}$ cannot delay in that state (see above), this guard can never be satisfied.

Remark 5 (Size of the Composition). While the NTA of $\mathcal{S}\otimes_{\mathcal{Q}}$ still has a reasonable size, the number of states of the NTA of a composed connector can be

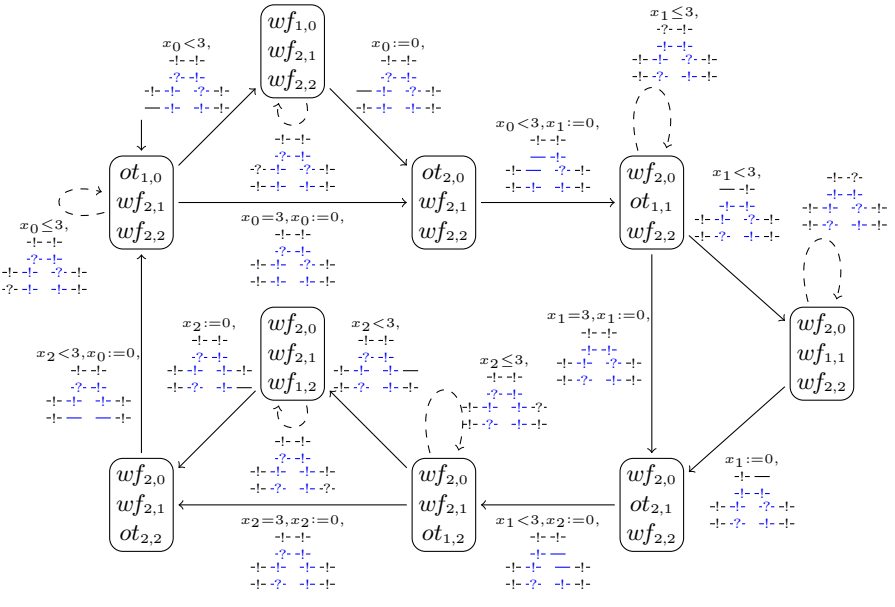


Fig. 4. Connector Composition: NTA of the Three-Token Ring Connector

exponential in the worst case. In Sect. 3.3, we define an encoding of the composition of connectors, which avoids the explicit construction of the composition, and is *linear* in the number of underlying connectors.

2.3 Semantics

We define the semantics of a connector $C{=}(P^r, P^w, T)$, with $T{=}(S, s_0, X, E)$, as the set of runs of the associated LTS $\mathcal{L}_C$, which describes transition sequences of T. A configuration q of $\mathcal{L}_C$ is a pair $\langle s, \nu\rangle$ of a state $s{\in}S$ and a clock valuation ν. A clock valuation is a mapping $\nu{:}X{\to}\mathbb{R}_{\geq 0}$ assigning a real value to each clock, its current value. $\mathcal{V}(X)$ denotes the set of all clock valuations over X, and we use $\models$ for the standard satisfaction relation.

The initial configuration of $\mathcal{L}_C$ is $\langle s_0, \mathbf{0}\rangle$, with $\mathbf{0}(x){=}0$ for all $x{\in}X$. Transitions of $\mathcal{L}_C$ directly correspond to the two types of transitions of NTA. An *action transition* $\langle s, \nu\rangle \xrightarrow{c} \langle s', \nu'\rangle$ describes the firing of an instantaneous communication $(s, \varphi, Y, c, s'){\in}E$, with $\nu{\models}\varphi$, and ν' resulting from ν by resetting all clocks in the set Y to zero. A *delayed action transition* $\langle s, \nu\rangle \xrightarrow{t,c} \langle s, \nu{+}t\rangle$, with delay $t{\geq}0$, describes the firing of a delay $(s, \varphi, \emptyset, c, s){\in}E$, where $\nu{+}t'{\models}\varphi$ for all $0{\leq}t'{\leq}t$ (that means the guard has to be satisfied at all times during the delay). In both cases, data flows according to colouring c.

An *execution of C of length k* is given by a *k-run of $\mathcal{L}_C$* , which is a sequence of k transitions, starting in the initial configuration: $\langle s, \mathbf{0}\rangle \xrightarrow{\gamma_1} q_1 \xrightarrow{\gamma_2} \ldots \xrightarrow{\gamma_k} q_k$, with $\gamma_i{\in}\mathbb{C}(P^r{\cup}P^w){\cup}(\mathbb{C}(P^r{\cup}P^w){\times}\mathbb{R}_{\geq 0})$, and $q_i{\in}(S{\times}\mathcal{V}(X))$, for all $1{\leq}i{\leq}k$. The *semantics of C* is given by the set Run_C of k-runs of $\mathcal{L}_C$, for all $k{\geq}0$.

Example (Semantics). Consider again the connector $\mathcal{S}{\otimes}\mathcal{Q}$ in Fig. 4, after hiding. A run of length 6 is given as[1]

$$\langle \begin{pmatrix} ot_{1,0} \\ wf_{2,1} \\ wf_{2,2} \end{pmatrix}, \mathbf{0}\rangle \xrightarrow{\substack{-!-!-\\-!-!-\\1,-?-!-}} \langle \begin{pmatrix} ot_{1,0} \\ wf_{2,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}1 \\ x_1{=}1 \\ x_2{=}1 \end{matrix}\rangle \xrightarrow{\substack{-!-!-\\-!-!-\\---!-}} \langle \begin{pmatrix} wf_{1,0} \\ wf_{2,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}1 \\ x_1{=}1 \\ x_2{=}1 \end{matrix}\rangle \xrightarrow{\substack{---!-\\-!-!-}} \langle \begin{pmatrix} ot_{2,0} \\ wf_{2,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}0 \\ x_1{=}1 \\ x_2{=}1 \end{matrix}\rangle$$

$$\xrightarrow{\substack{-!-!-\\-!-!-\\-!-!-}} \langle \begin{pmatrix} wf_{2,0} \\ ot_{1,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}0 \\ x_1{=}0 \\ x_2{=}1 \end{matrix}\rangle \xrightarrow{3,\substack{-?-!-\\-!-!-}} \langle \begin{pmatrix} wf_{2,0} \\ ot_{1,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}3 \\ x_1{=}3 \\ x_2{=}4 \end{matrix}\rangle \xrightarrow{\substack{-!-!-\\-!-!-\\-!-!-}} \langle \begin{pmatrix} wf_{2,0} \\ ot_{2,1} \\ wf_{2,2} \end{pmatrix}, \begin{matrix} x_0{=}3 \\ x_1{=}0 \\ x_2{=}4 \end{matrix}\rangle$$

After a delay of 1, the token is delivered to the environment through $w_{1,0}$, and accepted back immediately through $r_{1,0}$. The next transition corresponds to the token being transmitted from $S_{q,0}$ to $S_{q,1}$ (since internal ports are hidden, there is no visible dataflow). Next, $\mathcal{S}{\otimes}\mathcal{Q}$ delays in state $(wf_{2,0}ot_{1,1}wf_{2,2})$ for 3 time units, then, due to the timeout condition $x_1{=}3$, moves to $(wf_{2,0}ot_{2,1}wf_{2,2})$.

3 Encoding

In this section, we construct a formula $\varphi(C)$ in propositional logic with linear arithmetic, that encodes the transition relation of an NTA T of a connector C,

[1] The colourings—reflecting the layout in Fig. 4 after hiding—correspond to $w_{1,1}$, $r_{1,0}$, $w_{1,0}$ on the left (top to bottom) and $r_{1,1}$, $w_{1,2}$, $r_{1,2}$ on the right (top to bottom).

and we present an encoding of connector composition which is *linear* in the number of involved connectors. In the sequel, let $C{=}(P^r, P^w, T)$ be a connector, with $T{=}(S, s_0, X, E)$ over $P{\supseteq}\mathcal{I}_C$ (cf. Def. 2).

3.1 Basic Concepts

The presence (and absence) of dataflow through the ports of C depends on the state of T and the values of its clocks. We encode these concepts as follows.

For every port $p{\in}P$, we introduce two Boolean variables $\mathtt{p}^0$ and $\mathtt{p}^1$, to encode the three possible colours of p: the *encoding* $\mathtt{p}_{\langle c\rangle}$ *of* p *under colouring* c is $\neg\mathtt{p}^0{\wedge}\neg\mathtt{p}^1$ iff $c(p){=}\mathtt{-?-}$, $\neg\mathtt{p}^0{\wedge}\mathtt{p}^1$ iff $c(p){=}\mathtt{-!-}$, and $\mathtt{p}^0$ iff $c(p){=}\mathtt{---}$.

We use logarithmic encoding for states: for $|S|{=}n$, we introduce a vector $\boldsymbol{s}$ of $j{=}\lceil\log_2(n)\rceil$ Boolean variables, encoding a j-digit Boolean value. The intended meaning is that $\boldsymbol{s}$ encodes the value i, denoted by $\boldsymbol{s}_\mathtt{i}$, iff the connector is in state s_i; $\boldsymbol{s}_\mathtt{i}$ is called the *encoding of* s_i.

For every clock $x{\in}X$, we introduce a rational variable $\mathtt{x}$ (*clock reference*),[2] such that $\mathtt{x}$ holds the absolute point in time when x was last reset prior to the current step. An additional rational variable $\mathtt{z}$ (*absolute time reference*) denotes the absolute amount of time that has passed, such that the *clock value* of clock x is given by $\mathtt{z}{-}\mathtt{x}$, and the encoding φ of a clock constraint $\varphi{=}x{\sim}c$ is $\varphi{=}(\mathtt{z}{-}\mathtt{x}){\sim}c$. This temporal difference representation significantly reduces the number of arithmetic operations [14].

3.2 Transition Relation

The transition relation of T describes the possibilities to evolve to the next step, based on the configuration in the current step. In the sequel, the variables introduced in the previous section refer to the values *before* the firing of a transition, and primed variants refer to the values *after* the firing.

Definition 6 (Connector Encoding). *Let C and T be a connector and NTA as before, $f{=}(s_i, \varphi, Y, c, s_j)$ and $d{=}(s_k, \varphi, \emptyset, c, s_k)$ a communication respectively delay in E. The* encoding $\varphi(C)$ of C *is given in* (5) *(on the facing page).*

The encoding of a communication (1) ensures that the connector is in state s_i before firing, guard φ is satisfied, and data can flow according to colouring c. After firing, the connector is in state s_j, the values of the absolute time reference and clock references of $X{\setminus}Y$ have not changed, while all other clock references have been set to the actual point in time. The encoding of a delay (2) is similar, except that the value of the absolute time reference increases, while all other clocks keep their value. In addition, guard φ still needs to be satisfied after the time delay. The disjunction of these formulas expresses (nondeterministic) transition choice (3). The connector starts in its initial state, and initially all clocks start at zero (4).

[2] Linear arithmetic is equisatisfiable for rational and real variables [14], so rational variables are sufficient to encode real-valued clocks.

$$\varphi^f(f) = \boldsymbol{s}_{\mathtt{i}} \wedge \varphi \wedge (\mathtt{z}'{=}\mathtt{z}) \wedge \bigwedge_{x \in Y} (\mathtt{x}'{=}\mathtt{z}') \wedge \bigwedge_{x \in X \setminus Y} (\mathtt{x}'{=}\mathtt{x}) \wedge \bigwedge_{p \in \mathcal{I}_C} \mathtt{p}_{\langle \mathtt{c} \rangle} \wedge \boldsymbol{s}_{\mathtt{j}}' \quad (1)$$

$$\varphi^d(d) = \boldsymbol{s}_{\mathtt{k}} \wedge \varphi \wedge (\mathtt{z}' \geq \mathtt{z}) \wedge \bigwedge_{x \in X} (\mathtt{x}'{=}\mathtt{x}) \wedge \bigwedge_{p \in \mathcal{I}_C} \mathtt{p}_{\langle \mathtt{c} \rangle} \wedge \boldsymbol{s}_{\mathtt{k}}' \wedge \varphi' \quad (2)$$

$$\varphi^E(C) = \bigvee_{f \text{ comm.}} \varphi^f(f) \vee \bigvee_{d \text{ delay}} \varphi^d(d) \quad (3)$$

$$\varphi^i(C) = \boldsymbol{s}_0 \wedge (\mathtt{z}{=}0) \wedge \bigwedge_{x \in X} (\mathtt{x}{=}0) \quad (4)$$

$$\varphi(C) = \varphi^i(C) \wedge \varphi^E(C) \quad (5)$$

$$\varphi(C)^k = \varphi^i(C) \wedge \bigwedge_{0 \leq j \leq k} \varphi^E(C)^{j \cdot \prime} \quad (6)$$

$$\varphi^1(P) = \bigvee_{w \in P^w} \mathtt{w}^0 \rightarrow \Big(\bigwedge_{r \in P^r} \mathtt{r}^0 \wedge \bigwedge_{\substack{w_h, w_i \in P^w, \\ w_h \neq w_i}} \neg(\mathtt{w_h}^0 \wedge \mathtt{w_i}^0) \Big) \quad (7)$$

$$\varphi^2(P) = \bigvee_{r \in P^r} \mathtt{r}^0 \rightarrow \bigvee_{w \in P^w} \mathtt{w}^0 \quad (8)$$

$$\varphi^3(P) = \bigwedge_{p \in P} \neg \mathtt{p}^0 \rightarrow \Big(\bigwedge_{w \in P^w} \mathtt{w}^1 \vee \bigvee_{r \in P^r} \mathtt{r}^1 \Big) \quad (9)$$

$$\varphi(P) = \varphi^1(P) \wedge \varphi^2(P) \wedge \varphi^3(P) \quad (10)$$

$$\varphi(\mathcal{C} \otimes_{\mathcal{Q}}) = \bigwedge_{C \in \mathcal{C}} \varphi(C) \wedge \bigwedge_{Q \in \mathcal{Q}} \varphi(Q) \quad (11)$$

3.3 Connector Composition

Though the flip rule (Sect. 2.2) reduces the size of NTAs, the size of the NTA of a composed connector is still exponential in the worst case. Here, we define a *linear size* logical encoding of the composition of connectors. The basic idea is to define composition via conjunction of the encodings of the single connectors. In addition, we need to encode the constraints on internal ports, to ensure the encoding of the composition correctly models internal ports.

For a merge set P, the *encoding* $\varphi(P)$ *of internal port* $p_{\prec P}$ is given in (10). The constituents of $\varphi(P)$ directly correspond to the conditions in Sect. 2.2 (on Page 98). For example, (9) corresponds to condition 3: if there is no flow at all (left side of the implication), then (right side) either all write ports, or at least one read port provide a reason for delay (first respectively second disjunct). Using this, the *encoding of a composition* $\mathcal{C} \otimes_{\mathcal{Q}}$, for sets $\mathcal{C}$ and $\mathcal{Q}$ of disjoint connectors respectively merge sets (over the ports of connectors in $\mathcal{C}$), is defined in (11).

Hiding the internal ports amounts to existential quantification over the variables representing ports in $\mathcal{Q}$: the *reduction of* $\varphi(\mathcal{C} \otimes_Q)$ *to* $\mathcal{I}_{\mathcal{C}}$ is defined as $\exists \bigcup Q_i (\varphi(\mathcal{C} \otimes_{\mathcal{Q}}))$.

Example (Encoding). Consider again the connector $\mathcal{S} \otimes_{\mathcal{Q}}$ from Sect. 2.2, and the definition of its encoding (11). Due to space limitations, we do not show the complete encoding, but restrict this example to two instructive transitions of $\mathcal{S} \otimes_{\mathcal{Q}}$. For each of the underlying connectors $S_{q,i}$, $i{=}0, 1, 2$, we introduce a vector $\boldsymbol{si}$ of two Boolean variables, where $\boldsymbol{si}_0$, $\boldsymbol{si}_1$ and $\boldsymbol{si}_2$ are the encodings of states $ot_{1,i}$, $ot_{2,i}$ and $wf_{2,i}$, respectively. We show the encoding of the communication from $ot_{1,0}$ to $ot_{2,0}$ (12), and the delays in $ot_{1,0}$ (13), $wf_{2,1}$ (14) and $wf_{2,2}$ (15). The communication from $(ot_{1,0} wf_{2,1} wf_{2,2})$ to $(ot_{2,0} wf_{2,1} wf_{2,2})$ (in $\mathcal{S} \otimes_{\mathcal{Q}}$) is then given by the conjunction of (12), (14) and (15), and the delay in $(ot_{1,1} wf_{2,1} wf_{2,2})$ by the conjunction of (13), (14) and (15).

$$\boldsymbol{s0}_0 \wedge ((\mathtt{z}-\mathtt{x}_0)=3) \wedge (\mathtt{z}'=\mathtt{z}) \wedge (\mathtt{x}_0'=\mathtt{z}') \wedge \tag{12}$$
$$\neg \mathtt{r}_{1,0}{}^0 \wedge \mathtt{r}_{1,0}{}^1 \wedge \neg \mathtt{w}_{1,0}{}^0 \wedge \mathtt{w}_{1,0}{}^1 \wedge \neg \mathtt{r}_{2,0}{}^0 \wedge \mathtt{r}_{2,0}{}^1 \wedge \neg \mathtt{w}_{2,0}{}^0 \wedge \mathtt{w}_{2,0}{}^1 \wedge \boldsymbol{s0}_1{}'$$
$$\boldsymbol{s0}_0 \wedge ((\mathtt{z}-\mathtt{x}_0)\leq 3) \wedge (\mathtt{z}'\geq \mathtt{z}) \wedge (\mathtt{x}_0'=\mathtt{x}_0) \wedge \tag{13}$$
$$\neg \mathtt{r}_{1,0}{}^0 \wedge \mathtt{r}_{1,0}{}^1 \wedge \neg \mathtt{w}_{1,0}{}^0 \wedge \neg \mathtt{w}_{1,0}{}^1 \wedge \neg \mathtt{r}_{2,0}{}^0 \wedge \mathtt{r}_{2,0}{}^1 \wedge \neg \mathtt{w}_{2,0}{}^0 \wedge \mathtt{w}_{2,0}{}^1 \wedge \boldsymbol{s0}_0 \wedge ((\mathtt{z}'-\mathtt{x}_0')\leq 3)$$
$$\boldsymbol{s1}_2 \wedge (\mathtt{z}'\geq \mathtt{z}) \wedge (\mathtt{x}_1'=\mathtt{x}_1) \wedge \neg \mathtt{r}_{1,1}{}^0 \wedge \neg \mathtt{r}_{1,1}{}^1 \wedge \neg \mathtt{w}_{1,1}{}^0 \wedge \mathtt{w}_{1,1}{}^1 \wedge \neg \mathtt{r}_{2,1}{}^0 \wedge \mathtt{r}_{2,1}{}^1 \wedge \neg \mathtt{w}_{2,1}{}^0 \wedge \mathtt{w}_{2,1}{}^1 \wedge \boldsymbol{s1}_2 \tag{14}$$
$$\boldsymbol{s2}_2 \wedge (\mathtt{z}'\geq \mathtt{z}) \wedge (\mathtt{x}_2'=\mathtt{x}_2) \wedge \neg \mathtt{r}_{1,2}{}^0 \wedge \neg \mathtt{r}_{1,2}{}^1 \wedge \neg \mathtt{w}_{1,2}{}^0 \wedge \mathtt{w}_{1,2}{}^1 \wedge \neg \mathtt{r}_{2,2}{}^0 \wedge \mathtt{r}_{2,2}{}^1 \wedge \neg \mathtt{w}_{2,2}{}^0 \wedge \mathtt{w}_{2,2}{}^1 \wedge \boldsymbol{s2}_2 \tag{15}$$

3.4 Coordination as Constraint Satisfaction

With the encoding $\varphi(C)$ of a connector C (5), traditional constraint solving techniques [2] and tools (e.g. MATHSAT [16] or HySAT [11]) are used to model check the behaviour and verify properties of C. To inspect executions of C of length k (cf. Sect. 2.3), the encoding $\varphi(C)$ of C is *unfolded* k times, i.e., instantiated for all steps up to k. The resulting formula is shown in (6). The formula $\varphi^E(C)^{j\cdot\prime}$ denotes the variant of $\varphi^E(C)$, where j primes have been added to all variable symbols (e.g., $\varphi^E(C)^{3\cdot\prime}$ contains $\mathtt{z}^{3\cdot\prime}=\mathtt{z}'''$). Intuitively, a satisfying valuation (*model*, i.e., a single solution to the satisfiability check) of $\varphi(C)^k$ corresponds to an execution of length k, and the set of all models precisely describes the coordination pattern of C (for executions up to length k).

Other properties used to analyse the behaviour of C include for example whether a certain error state s is reachable within k steps. This amounts to conjoining $\varphi(C)^k$ with $\rho \stackrel{\text{def}}{=} \boldsymbol{s}^{0\cdot\prime} \vee \ldots \vee \boldsymbol{s}^{k\cdot\prime}$, the error state is (un)reachable iff the conjunction is (un)satisfiable. Lifting ρ to reason about configurations (i.e., include timing information) is straightforward. Other bounded LTL properties can be specified using the encoding in [5], for example. The next section shows how to use constraint satisfaction to check the correctness of the $\mathcal{S}\otimes_{\mathcal{Q}}$ connector.

3.5 Preliminary Experimental Results

Some preliminary experimental results (runtime and memory consumption) are shown below. All experiments have been carried out with MATHSAT, on an Intel Core 2 Quad with 2.83GHz, 8GB RAM and Fedora 10. The input file, containing the system description and the representation of the properties, can be found at `http://www.cwi.nl/~kemper/ThreeTokenRing/`. For the interested reader, we have added some comments to the file, to ease understanding the encoding.

We checked three correctness properties for the encoding $\varphi(\mathcal{S}\otimes_{\mathcal{Q}})$ of the $\mathcal{S}\otimes_{\mathcal{Q}}$ connector, for executions of length 20 and 50, respectively. Property MORETOKENS is satisfiable iff $\mathcal{S}\otimes_{\mathcal{Q}}$ can reach a state where more than one of the underlying connectors is in an "*ot*" state, i.e., where more than one token is offered at the same time: though $\mathcal{S}\otimes_{\mathcal{Q}}$ is composed from three two-token rings, there must be only one token in the composition (remember that we changed the initial states of $S_{q,1}$ and $S_{q,2}$ for this purpose). The property SHORTCUT is satisfiable iff it is possible to fire the communication from $ot_{2,i}$ to $ot_{1,i}$ in any of the underlying connectors: firing it in $S_{q,0}$, for example, would wrongly skip the connectors

$S_{q,1}$ and $S_{q,2}$, and immediately offer the token through port $w_{1,0}$ again. Property NoSeqFlow is used to check that every dataflow through a $w_{1,i}$ port is followed by dataflow through the corresponding $r_{1,i}$ port, without dataflow through any other port of the external interface in between; it is satisfiable iff the sequential order is violated. As expected, the result of checking the conjunction of $\varphi(\mathcal{S} \otimes_{\mathcal{Q}})$ with any of the properties is "unsatisfiable".

length	MoreTokens		Shortcut		NoSeqFlow	
20	1.493s	23.465MB	1.481s	23.227MB	4.526s	33.215MB
50	3.300s	29.598MB	2.242s	24.867MB	9.431s	39.484MB

The results clearly show that our approach is tailored towards and profits from using well-optimised, high-end constraint solving techniques, as it scales very well for long executions: the increase in runtime is roughly linear in the increase of the execution length, while at the same time, the number of possible executions that need to be checked increases exponentially.

4 Conclusion and Future Work

In this paper, we have presented a modular framework for *compositional construction of* and *coordination in* real-time dataflow networks, which takes into account environmental constraints from outside the network. We have defined a new type of transition systems (NTA), used to describe the behaviour of components and, since our approach is compositional, whole networks. This *direct* definition of the behaviour makes it easy to understand (and thus, use) our framework. In addition, it also facilitates the introduction of new, user-defined primitives, by just giving the underlying NTA. Liberal notions of components and networks allow to encode many common (coordination) models, like e.g. TA [1] or timed constraint automata (TCA) [15], in our framework.

We have defined a constraint-based encoding of connectors, using propositional logic with linear arithmetic. This enables us to benefit from well-studied, high-end constraint solving techniques, when checking for valid interactions within a network, or inspecting the incorporated coordination pattern. The logical encoding of networks —i.e. composition of connectors—is *linear* in the size of the NTAs of the underlying connectors. In this way, we overcome the omnipresent state explosion problem, and are able to deal with larger systems.

We do not consider concrete data values which are transmitted, but rather focus on the presence and absence of dataflow. However, the integration of handling these data values is straightforward: transitions of NTA are augmented with data constraints in a "TCA-like" style, the basis for encoding these constraints has already been established in [9].

Besides this, future work includes comparisons of our approach on different real-world case studies, possibly using different constraint solvers. So far, no tool support for the full theory presented in this paper exists. Yet, for the trivial case of untimed connectors, our approach is essentially equivalent to the animation of $\mathcal{R}$eo [3] in the ECT framework [10]. We plan to further integrate our work

into this framework. In particular, implement an editor for NTA, which provides composition and hiding, such that connectors and networks can be easily defined by users. Adding the translation from NTA to constraints will then offer support for the full theory. We expect some performance gains when using our constraint-based approach as underlying theory for computing the $\mathcal{R}$eo animations in the ECT, which will increase the manageable system size.

References

1. Alur, R.: Timed automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
2. Apt, K.R.: Principles of Constraint Programming. Cambridge Univ. Press, Cambridge (2003)
3. Arbab, F.: $\mathcal{R}$eo: a channel-based coordination model for component composition. Mathematical Structures in Comp. Sci. 14(3), 329–366 (2004)
4. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in $\mathcal{R}$eo by constraint automata. Sci. Comp. Prog. 61(2), 75–113 (2006)
5. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
6. Bonsangue, M., Clarke, D., Silva, A.: Automata for Context-Dependent Connectors. In: Field, J., Vasconcelos, V.T. (eds.) COORDINATION 2009. LNCS, vol. 5521, pp. 184–203. Springer, Heidelberg (2009)
7. Brock, J.D., Ackerman, W.B.: Scenarios: A model of non-determinate computation. In: Díaz, J., Ramos, I. (eds.) Formalization of Programming Concepts. LNCS, vol. 107, pp. 252–259. Springer, Heidelberg (1981)
8. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Sci. Comp. Prog. 66(3), 205–225 (2007)
9. Clarke, D., Proença, J., Lazovik, A., Arbab, F.: Deconstructing $\mathcal{R}$eo. Electr. Notes Theor. Comput. Sci. 229(2), 43–58 (2009)
10. Eclipse Coordination Tools, `http://reo.project.cwi.nl/`
11. HySAT Bounded Model Checker, `http://hysat.informatik.uni-oldenburg.de`
12. Jonsson, B.: A fully abstract trace model for dataflow networks. In: POPL, pp. 155–165 (1989)
13. Kahn, G.: The semantics of a simple language for parallel programming. In: IFIP Congress, pp. 471–475 (1974)
14. Kemper, S., Platzer, A.: SAT-based abstraction refinement for real-time systems. Electr. Notes Theor. Comput. Sci. 182, 107–122 (2007)
15. Kemper, S.: SAT-based verification for timed component connectors. Electr. Notes Theor. Comput. Sci. 255, 103–118 (2009)
16. The MathSAT 4 SMT solver, `http://mathsat4.disi.unitn.it`
17. Ren, S., Agha, G.: RTsynchronizer: Language support for real-time specifications in distributed systems. In: LCT-RTS, pp. 50–59 (1995)
18. Wegner, P.: Coordination as constrainted interaction (extended abstract). In: Hankin, C., Ciancarini, P. (eds.) COORDINATION 1996. LNCS, vol. 1061, pp. 28–33. Springer, Heidelberg (1996)