Softw Syst Model (2007) 6:59–82
DOI 10.1007/s10270-006-0009-9

SPECIAL SECTION PAPER

Models and temporal logical specifications for timed component connectors

Farhad Arbab · Christel Baier · Frank de Boer · Jan Rutten

Received: 11 February 2005 / Accepted: 2 February 2006 / Published online: 22 August 2006

Abstract Component-based software engineering advocates construction of software systems through composition of coordinated autonomous components. Significant benefits of this approach include software reuse, simpler and faster construction, enhanced reliability, and dramatic reductions in the complexity of construction of provably correct critical systems, many of which involve real-time concerns. Effective, flexible component composition by itself still poses a challenge today and yet the special nature of real-time constraints makes component-based construction of real-time systems even more demanding. The coordination language Reo supports compositional system construction through connectors that exogenously coordinate the interactions among the constituent components which unawarely comprise a complex system, into a coherent collaboration. The simple, yet surprisingly rich, calculus of channel composition that underlies Reo offers a flexible framework for compositional construction of coordinating component connectors with real-time properties. In this paper, we present an operational semantics for the channel-based component connectors of Reo in terms of Timed Constraint Automata and introduce a temporal-logic for specification and verification of their real-time properties.

Communicated by Dr. Jorge Cuellar.

F. Arbab · F. de Boer (✉) · J. Rutten
Department of Software Engineering,
Centrum voor Wiskunde en Informatica,
Amsterdam, The Netherlands
e-mail: F.s.de.Boer@cwi.nl

C. Baier
Institut für Informatik I, Universität Bonn,
Bonn, Germany

F. Arbab · F. de Boer
Universiteit Leiden,
Leiden, The Netherlands

J. Rutten
Vrije Universiteit Amsterdam,
Amsterdam, The Netherlands

Keywords Coordination · Real-time · Composition · Reo · Constraint automata · Timed automata · Linear temporal logic · Timed data streams

1998 ACM Computing Classsification C.2.4, D.1.3, D.2.4, D.2.6, D.2.11, D.2.13, D.3.2, D.3.3, F.1.2, F.3.1, F.3.2, F.3.3

1 Introduction

The task of designing a complex concurrent system with several components requires a *coordination model* that formalizes their mutual interactions. The internals of black-box components cannot be modified to implement such coordinated interactions. Coordination, therefore, becomes the responsibility of the "glue-code" that interconnects the constituent components of a composite system, and of its underlying run-time middle-ware. Reo [6] offers a powerful glue language for implementation of coordinating component connectors based on a calculus of mobile channels.

In this paper, we consider the real-time aspects of Reo when the behavior specification of channels and component interfaces can involve *timing constraints*. Because connectors, not components, are the primary concern in Reo, our primary interest here is with channels whose behavior involves temporal constraints;

and with their composition. For instance, a deadline *t* for the availability of some data can be formalized as the behavior of a FIFO channel that associates an *expiration date*, *t*, with every data item that enters its buffer: the channel loses a data item in its buffer *t* units of time after it enters through its source (unless, of course, it is dispensed through its sink in the meanwhile). Another example is a timer channel that becomes activated by a data item through its source, after which it returns a timeout signal through its sink, after a specified delay of exactly *t* units of time.

As the operational model for Reo connector circuits, we use *timed constraint automata* (TCA) which extend their untimed version [9] with the concepts borrowed from classical timed automata with location invariants [2,16]. TCA have two kinds of transitions: (1) internal changes of the locations caused by some time constraints and (2) transitions that represent the synchronized execution of I/O-operations at some of the ports. Using ideas similar to [9], the construction of a timed constraint automaton from a given timed Reo circuit can be performed in a *compositional* manner, using composition operators on TCA that model Reo's operators *join* and *hide* to build complex connectors out of instances of basic channel types.

One conceptual difference between TCA and classical timed automata is the treatment of immediate actions or urgent synchronous channels, as they are used, e.g., in the tools [20,17,37]. The assumption that synchronous I/O-operations must be executed as soon as they become enabled makes no sense in our framework. For instance, assume that we have a FIFO channel carrying data from node *A* to node *B* and a synchronous channel from *B* to another node *C*. As soon as *A* places a value in the FIFO buffer it becomes available for consumption through node *B*, and thus, the synchronous communication between *B* and *C* becomes enabled. On the other hand, the input and output of the same data item must not occur simultaneously through a FIFO channel, by its definition. Thus, we need a delay for the synchronization between *B* and *C*. Moreover, Reo allows to explicitly specify deadlines of "shortly delayed" activities or other time constraints (e.g., lower bounds for the delay) using an appropriate combination of timed channels.

The semantics of the TCA and timed Reo circuits relies on *timed data streams* as in [7,9], comprising a formalization of the possible data-flow at each node over time. To specify a desired coordination mechanism, we use a variant of linear temporal logic (LTL) [22,27] with real-time constraints, which we call *timed scheduled-data-stream logic* (TSDSL) and has a semantics based on timed data streams. TSDSL essentially relies on a combination of the time-abstract temporal modalities in LTL and timed regular expressions [10]. We show through a series of examples how TSDSL can serve as a specification formalism for (timed) Reo circuits, sketch the ideas of a model checking algorithm, and explain the relation of TSDSL with refinement relations.

Related models. There are several other related real-time models that also focus on aspects of coordination. Timed interface automata (TIA) [1] or real-time variants of I/O-automata, e.g., [13,19,24], are related to TCA in the same way as their untimed versions. I/O-automata rely on the assumption of input-enabledness which is not required (and would not make sense) in constraint automata.

The purpose of TIA is orthogonal to our approach involving timed Reo connectors and TCA. (There are some conceptional differences, e.g., TIA use action labels rather than port names, but these are not important as the formal definition of TIA and TCA can be adapted to eliminate these differences.) The major goal of TIA is to provide a formalism to specify and to check the compatibility of real-time components by means of their interfaces. Our focus is on compositional reasoning about (design and analysis of) channel-based coordination mechanisms, based on their data-flow. Thus, our framework allows to design and analyze a coordination context in which certain components are used and to construct their interfaces, while the approach of interface automata allows to check a-posteriori whether a design makes the components work together in the desired way.

Although compositionality in timed Reo and TCA is in the spirit of real-time process algebras, e.g., [21, 29,36], Reo's philosophy of composing connectors out of a variety of basic channel types via join and hiding and supporting any kind of synchronous or asynchronous communication differs from classical process algebra approaches which provide operators for modeling choice, parallelism, and recursion (all of which are implicit in Reo).

In some respects, Reo circuits superficially resemble Petri nets. However, there are major differences between the two. Petri nets are constructed out of a fixed set of building blocks (i.e., *places*, *transitions*, and *arcs*), each with a fixed behavior, that can be composed in a prescribed fashion. In contrast, Reo defines a fixed set of composition rules and allows an arbitrary set of user defined channels as primitives with arbitrary behavior, on which its composition rules can be applied to construct connector circuits. This allows a harmonious combinations of synchrony and asynchrony in the same model which is not possible in Petri nets. It also allows incorporation of arbitrary computational entities into a composed Reo circuit. Specifically, as we show in this paper, real-time constraints can be easily incorporated into Reo simply by adding a few channels with time-sensitive behavior to the user-defined repertoire of primitives used in the construction of circuits, without any extension or revision of the Reo model or its composition rules. On the other hand, to express temporal constraints in Petri nets, various extended models have been proposed and studied, each revising the semantics of a basic Petri net model by associating temporal constraints with (1) the availability of tokens in places, (2) transitions, or (3) arcs [18,23,28,31].

Using proper time-sensitive channles, Reo can cover the temporal constraints modeled in various timed Petri nets. Timed or un-tiemd Petri nets differ from Reo in that synchrony and exclusion constraints propagate through (the synchronous sub-sections of) Reo circuits. This is generally not the case in Petri nets, because their transitions are local. Petri net transition nodes enable them to directly synchronize otherwise unrelated events, thus enforcing a synchronous *and* of several arcs/events. However, Petri nets have no primitive for the dual synchronous *or* of several arcs. The *or* of several arcs is possible only if they end in the same place, which implies the commitment of moving a token into that place. This means that events/arcs can be directly *and*-synchronized to compose more complex synchronous transitions (i.e., one-step atomic transactions), but a synchronous *or* of events/arcs is not possible, i.e., two transitions cannot be connected together without an intervening place/commitment. This disallows a direct modeling of composite atomic transactions in Petri nets and prevents arbitrary combinations of synchrony and asynchrony.

Organization of the paper. Timed constraint automata are introduced in Sect. 2. Section 3 contains a brief overview of Reo. In Sect. 4 we introduce timed primitive channels for constructing Reo circuits, provide some examples for Reo circuits with timing constraints, and explain how timed constraint automata can serve as their operational model. Timed scheduled-data-stream logic (TSDSL) is introduced in Sect. 5. In addition, we provide some examples to illustrate how TSDSL can serve to specify timed component connectors and sketch the main steps of a TSDSL model checking algorithm. Sect. 6 concludes the paper.

2 Timed constraint automata

The formal definition of timed constraint automata arises by combining the concepts of constraint automata [9] and timed automata [2,16]. We first introduce the syntax of TCA and provide some intuitive examples (Section 2.1) and then provide their semantics by means of an infinite state-transition graph and the induced language of timed scheduled data streams (Sect. 2.2).

2.1 Syntax of TCA

Edges in timed constraint automata are labeled with tuples (N, dc, cc, C) where N is a set of ports/nodes that synchronously perform certain I/O-operations, dc is a data constraint that specifies the concrete values that are transferred through those I/O-operations, cc is a clock constraint, and C is a set of clocks that are reset to 0. If $N = \emptyset$ then the edge represents an internal move (in which case $dc = \mathsf{true}$).

Before presenting the formal definition, we give a simple example. Figure 1 shows on its left a Reo circuit with a 1-bounded FIFO-channel with expiration, connecting nodes A and B, and a synchronous channel connecting nodes B and C. A FIFO channel "with expiration" is a lossy channel that loses any data item that remains in its buffer longer than its "expiration date" which in this case is 3 time units after it enters the buffer of the channel. Thus, in this example, there is an implicit deadline for the data transfer operation at node B. The graph on the right shows the TCA corresponding to this Reo circuit. In the TCA on the right-hand-side in Fig. 1, location s stands for the initial configuration where the buffer is empty, while location $\bar{s}(d)$ represents the configuration where the buffer is filled with data element d. If nodes B and C are ready for I/O-operations within 3 time units, in location $\bar{s}(d)$, then we assume that B takes an element d from the buffer and immediately forwards it to C. This corresponds to the transition labeled with the set $\{B, C\}$ and the data constraint $d_B = d_C = d$. Although there is no explicit lower time bound for the delay of the $\{B, C\}$-transition, our semantics forces some time elapse in location $\bar{s}(d)$ before the $\{B, C\}$-transition can fire, even if B and C are waiting for an input value. This is different in ordinary timed automata, but is needed here because a FIFO channel (by its definition) does not allow for the synchronous trans-

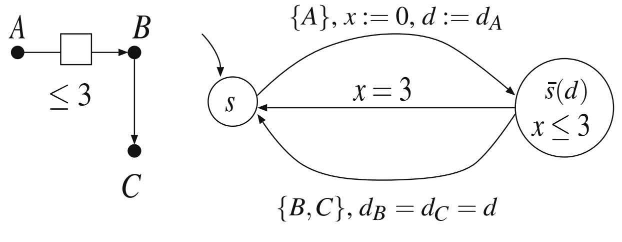


Fig. 1 Reo circuit and timed constraint automation

fer of data from its source to its sink end. If B cannot transfer the element out of the FIFO buffer (because no I/O operation is available on C to synchronize with B), the message is lost 3 time units after entering $\bar{s}(d)$. This is modeled by the invariance condition $x \leq 3$ at location $\bar{s}(d)$ which forces the automaton to leave $\bar{s}(d)$ if the current value of x is 3.

Notation 2.1 (Data assignments, data constraints) In the sequel, we assume finite and non-empty sets *Data* consisting of data items that can be transferred through channels, and $\mathcal{N}$ consisting of node names. A data assignment denotes a function $\delta : N \to \mathit{Data}$ where $\emptyset \neq N \subseteq \mathcal{N}$. We use notations like $\delta = [A \mapsto \delta_A : A \in N]$ to describe the data-assignment that assigns the value $\delta_A \in \mathit{Data}$ to every node $A \in N$. Data constraints can be viewed as a symbolic representation of *sets* of data assignments. Formally, data constraints (denoted dc) are propositional formulas built from the atoms "$d_A \in P$" and "$d_A = d_B$" where $A, B \in \mathcal{N}$ and $P \subseteq \mathit{Data}$ (plus the standard boolean connectors $\wedge$, $\vee$, $\neg$, etc.). For $N \subseteq \mathcal{N}$, $DA(N)$ denotes the set of all data assignments for the node-set N and $DC(N)$ the set of data constraints that at most refer to the terms d_A for $A \in N$. We write DA for $\bigcup_{\emptyset \neq N \subseteq \mathcal{N}} DA(N)$ and DC for $DC(\mathcal{N})$. □

Notation 2.2 (Clock assignments, clock constraints) Let $\mathcal{C}$ be a finite set of clocks. A clock assignment means a function $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$. If $t \in \mathbb{R}_{\geq 0}$ then $\nu + t$ denotes the clock assignment that assigns the value $\nu(x) + t$ to every clock $x \in \mathcal{C}$. If $C \subseteq \mathcal{C}$ then $\nu[C := 0]$ stands for the clock assignment that returns the value 0 for every clock $x \in C$ and the value $\nu(x)$ for every clock $x \in \mathcal{C} \setminus C$. A clock constraint (denoted cc) for $\mathcal{C}$ is a conjunction of atoms of the form "$x \bowtie n$" where $x \in \mathcal{C}$, $\bowtie \in \{<, \leq, >, \geq, =\}$ and $n \in \mathbb{N}$. $CA(\mathcal{C})$ (or CA) denotes the set of all clock assignments and $CC(\mathcal{C})$ (or CC) the set of all clock constraints. □

The symbol $\models$ stands for the obvious satisfaction relation for data (or clock) constraints which results from interpreting data (clock) constraints over data (clock) assignments. Satisfiability, validity, logical equivalence $\equiv$ and logical implication $\leq$ of data (clock) constraints are defined as usual. For data constraints, we often use simplified notations such as "$d_A = d$" rather than "$d_A \in \{d\}$".

Definition 2.3 *(Timed constraint automata) A TCA is a tuple $\mathcal{T} = (S, \mathcal{C}, \mathcal{N}, \mathcal{E}, S_0, ic)$ where S is a finite set of control states (also called locations), $\mathcal{C}$ a finite set of clocks, $\mathcal{N}$ a finite set of node names, and $S_0 \subseteq S$ a set of initial locations. $ic : S \to CC$ is a function that assigns to any location s an invariance condition $ic(s)$. The edge relation $\mathcal{E}$ is a subset of $S \times 2^{\mathcal{N}} \times DC \times CC \times 2^{\mathcal{C}} \times S$ such that $dc \in DC(N)$ for any edge $e = (s, N, dc, cc, C, \bar{s}) \in \mathcal{E}$. Moreover, we assume that all data and clock guards on the edges and the invariance conditions are satisfiable. (For edges with the empty node-set, we require a data constraint dc with $dc \equiv$ true.)* □

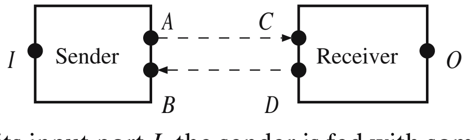
The automaton in Fig. 1 is a simplified picture for a TCA where d is used as a data parameter. The presented TCA has the location space $S = \{s\} \cup \{\bar{s}(d) : d \in \mathit{Data}\}$. For instance, if $\mathit{Data} = \{0, 1\}$ then the label "$\{A\}, x := 0, d := d_A$" in the parametric TCA for the edge leading from location s to $\bar{s}(d)$ stands for the two edges

$$s \xrightarrow{\{A\}, x:=0, d_A=0} \bar{s}(0) \text{ and } s \xrightarrow{\{A\}, x:=0, d_A=1} \bar{s}(1)$$

in the non-parametric TCA. That is, the assignment "$d := d_A$" in the parametric version stands for the data constraint $d_A = d$ in the TCA. These parametric TCA only serve to simplify the pictures for TCA with data-dependent edges.

An interface specification of a *timed sequencer* that coordinates the data-flow of two components via synchronous channels is shown in Fig. 2. Here and in the sequel, we skip the guards (data or clock constraint) of the edges when they are true. We assume the deadline $t = 3$ for the write-operations, that is, the sequencer in location s waits up to 3 time units to synchronize with component 1. If it fails then the sequencer moves via the edge labeled with the empty set to location $\bar{s}$ and tries to synchronize with component 2, and so on. Note that a single clock x suffices since clock x serves to measure the amount of time staying in one of the locations s or $\bar{s}$. After changing the location, clock x is "reused" to measure the sojourn time in the new location.

Example 2.4 (Alternating Bit Protocol) We consider a variant of the ABP where two components (the sender and the receiver) are connected via lossy synchronous channels. We follow here essentially the description in [25] but do not assume unreliable channels that may lose data in an unpredictable way. Instead, we assume lossy synchronous channels (as in Reo, see Sect. 4) where a data item written to the source end of such a channel is lost if the sink end of the channel cannot perform a matching I/O-operation to consume it.



Via its input port I, the sender is fed with some input which it delivers to the receiver via the channel connecting ports A and C. The receiver acknowledges the receipt of the message via the channel between D and B and outputs the message through its port O. The sender attaches a bit to the messages and expects the corresponding control bit as acknowledgment. If the expected control bit b arrives through port B then the sender switches its mode and sends the next message together with the bit $-b$. If a certain deadline (t_S in our example) expires then the sender resends the message with the same control bit b with a delay of at most ρ_S. The same upper bound ρ_S is assumed for the time interval between the receipt of a message d on input port I and the sending a message from output port A. Acknowledgments that contain a non-expected control bit are ignored as they belong to the previous message.

The behavior of the receiver is complementary to that of the sender. In mode b, the receiver waits for the arrival of an input (d, b) through its port C and acknowledges its receipt with the bit b, while messages of the form $(d, -b)$ are ignored. The receiver resends the acknowledgment if the next message with the expected control bit b does not arrive within t_R time units. (In particular, the receiver resends the control bit of the last message infinitely many times if data-flow at port I eventually terminates.) Moreover, we assume the upper time bound ρ_R for the success of the write-operation on output port O as well as the receiver's acknowledgment by sending the control bit.

Figure 3 shows the interface specifications for the sender and the receiver as (data parametrized) TCA. We assume here the data domain $Data = \{0,1\} \cup Msg \cup Msg \times \{0,1\}$ and write msg to denote the projection of the pairs (d, b) to the message-component (i.e., $msg(d, b) = d$). For the sender we use the locations in(b), try(d, b) and wait(d, b) where $d \in Msg$ and $b \in \{0,1\}$ is the control bit. In location in(b) the sender waits for the next input value d. In location try(d, b), the sender tries to deliver the obtained message d by sending (d, b) along channel AC. Location wait(d, b) serves to wait for the acknowledgement (control bit b) and to resend message d after t_S time units. The intuitive meaning of the locations of the receiver is analogous: in location out(d, b) the receiver delivers the obtained data item d via output port O, while locations wait(b) and ack(b) represent waiting for an input (d, b) via channel AC and acknowledging the receipt of a message, respectively.

Figure 4 shows the "combined" TCA $\mathcal{T}_{ABP}$ for the ABP. Essentially, this TCA is obtained by the join operator (see Definition 4.5), while taking care of the special semantics of lossy synchronous channels, which forces its sink and the source ends to synchronize if both can perform I/O-operations □

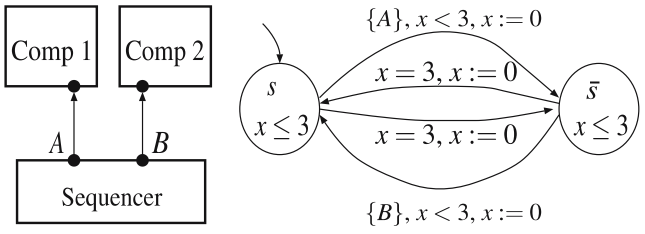


Fig. 2 Timed sequencer

2.2 The state-transition graph of a TCA

So far we described the syntax of TCA and gave some intuitive explanations for their meaning. The following definition formalizes this intuitive behavior by means of a state-transition graph. Following the standard semantics of timed automata [2,16], we use a dense time domain where the values of the clocks can be arbitrary real numbers. Dense time models are more appropriate than discrete time models when dealing with distributed, asynchronous systems where the components are not synchronized via a single global clock. The states consist of the current location of the TCA and the current values of the clocks. The transitions corresponding to a set of visible I/O-operations arise through passage of time, followed by the I/O-operations specified by some edge with a non-empty node-set. Invisible transitions (transitions with no observable data flow) are obtained by the edges with empty node-set. They can occur immediately, i.e., without any passage of time, in the current location.

Definition 2.5 *(State-transition graph of a TCA) Given a TCA $\mathcal{T}$ as above, $\mathcal{T}$ induces a state-transition graph $\mathcal{A}_{\mathcal{T}} = (Q, \longrightarrow, Q_0)$ as follows. The states are pairs $q = \langle s, \nu \rangle$ consisting of a location s and a clock assignment ν. Thus, the state space is $Q = S \times CC$. The set of initial states is $Q_0 = \{\langle s_0, \mathbf{0} \rangle : s_0 \in S_0, \mathbf{0} \models ic(s_0)\}$ where $\mathbf{0}$ stands for the clock assignment that returns the value 0 for all clocks. The transition relation $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DA \times \mathbb{R}_{\geq 0} \times Q$ is defined by the following rules:*

$$\frac{\begin{array}{l}(s, N, dc, cc, C, \bar{s}) \in \varepsilon, \\ t > 0\, s.t.\, \nu + \bar{t} \mid = ic(s)\, for\, all\, 0 < \bar{t} \leq t \\ (\nu + t)[C := 0] \mid = ic(\bar{s})\, and\, \nu + t \mid = cc \\ \quad \delta \in DA(N)\, s.t\, \delta \mid = dc\end{array}}{\langle s, \nu \rangle \xrightarrow{N, \delta, t} \langle \bar{s}, (\nu + t)[C := 0] \rangle}.$$

If $N = \emptyset$, we use in addition the same rule with $t = 0$:

$$\frac{(s, \emptyset, \mathsf{true}, cc, C, \bar{s}) \in \mathcal{E}, \nu[C := 0] \models ic(\bar{s}),\ \nu \models cc}{\langle s, \nu \rangle \xrightarrow{\emptyset, \emptyset, 0} \langle \bar{s}, \nu[C := 0] \rangle}.$$

A state $q = \langle s, \nu \rangle$ is called terminal iff it has no outgoing transitions, but allows the possibility for unbounded passage of time, i.e., $\nu + t \models ic(s)$ for all $t > 0$. A time-lock

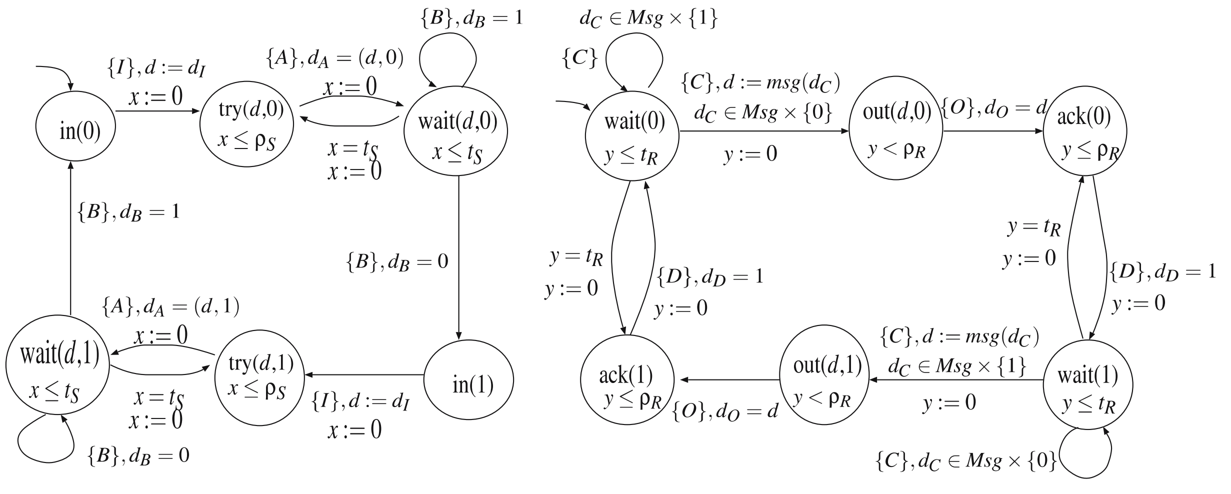


Fig. 3 TCA for the sender and the receiver of the ABP

refers to a state $q = \langle s, \nu\rangle$ that has no outgoing transitions and there exists some $t\langle 0$ with $\nu + t \not\models ic(s)$. $\mathcal{T}$ is called time-lock free iff $\mathcal{A}_\mathcal{T}$ does not contain a reachable time-lock. □

For instance, the reachable part of the state-transition graph of the timed sequencer in Fig. 2 consists of the states
$\langle s, x = \vartheta\rangle$ and $\langle \bar{s}, x = \vartheta\rangle$ where $\vartheta \in [0, 3]$. The outgoing transitions of the states $\langle s, x = \vartheta\rangle$ with $0 \le \vartheta < 3$ are:

$$\langle s, x = \vartheta\rangle \xrightarrow{\{A\},\text{true},t} \langle \bar{s}, x = 0\rangle, \quad \text{where } t > 0 \text{ and } \vartheta + t < 3$$

and $\langle s, x = \vartheta\rangle \xrightarrow{\emptyset,\text{true},3-\vartheta} \langle \bar{s}, x = 0\rangle$. For state $\langle s, x = 3\rangle$ there is only a single outgoing transition, namely

$$\langle s, x = 3\rangle \xrightarrow{\emptyset,\text{true},0} \langle \bar{s}, x = 0\rangle,$$

which is taken without any further passage of time in location s. The outgoing transitions of the states $\langle \bar{s}, x{=}\vartheta\rangle$ are analogous. Thus, the timed sequencer in Fig. 2 is time-lock free. However, if we remove the clock reset "$x := 0$" from the two edges with empty node-sets then the resulting TCA has a time-lock in states $\langle s, x = 3\rangle$ and $\langle \bar{s}, x = 3\rangle$ since the invariance conditions in s and $\bar{s}$ do not allow for any passage of time.

Edges with non-empty node-sets can fire only after some positive delay. This reflects the general idea of constraint automata where all observable activities that occur at the same time instant (i.e., atomically) are collapsed into a single transition.

Notation 2.6 (Runs, time divergence) Let $\mathcal{T}$ be a TCA as before and $q = \langle s, \nu\rangle$ a state in $\mathcal{A}_\mathcal{T}$. A q-run (or briefly run) in $\mathcal{T}$ denotes any (finite or infinite) sequence of successive transitions in $\mathcal{A}_\mathcal{T}$ starting in state q. Formally, a q-run has the form

$$\mathbf{q} = q_0 \xrightarrow{N_0,\delta_0,t_0} q_1 \xrightarrow{N_1,\delta_1,t_1} \cdots,$$

where $q_0 = q$. The q-run $\mathbf{q}$ is called initial if $q_0 \in Q_0$. The q-run $\mathbf{q}$ is called time divergent if $\mathbf{q}$ is infinite and $t_0 + t_1 + \ldots = \omega$. Maximality of a run means that it is either time divergent or finite and ends in a terminal state. □

Intuitively, N_i is the set of nodes in state q_i that are scheduled to synchronously perform the next set of I/O-operations, while δ_i represents the concrete values that are exchanged through those operations at the nodes $A \in N_i$. The value t_i stands for the delay.

We next define the notion of a TSD stream which will serve to formalize the observable data flow of the runs in a TCA. TSD streams are sequences of triples $(N, \delta, \bar{t})$ where N is a nonempty node-set, δ a data assignment for the nodes in N and $\bar{t}$ a point in time. The intuitive meaning of $(N, \delta, \bar{t})$ is that at time $\bar{t}$ the nodes $A \in N$ simultaneously perform the I/O-operations specified by the pair (N, δ).

Notation 2.7 (TSD stream) A timed scheduled data stream for a node-set $\mathcal{N}$ denotes any (finite or infinite) sequence $\Theta = (N_0, \delta_0, \bar{t}_0), (N_1, \delta_1, \bar{t}_1), \ldots \in (2^\mathcal{N} \times DA \times \mathbb{R}_{>0})^\infty$ such that $N_i \ne \emptyset$, $\delta_i \in DA(N_i)$, $0 < \bar{t}_0 \le \bar{t}_1 < \ldots$ and $\lim_{i\to\infty} \bar{t}_i = \omega$ if $\mathbf{q}$ is infinite. The empty TDS stream is denoted by the symbol ε. The length $|\Theta| \in \mathbb{N} \cup \{\omega\}$ is defined as the number of triples $(N, \delta, \bar{t})$ in Θ. The execution time $\tau(\Theta)$ is ω if Θ is infinite, $\bar{t}_k$ if $|\Theta| = k + 1$,

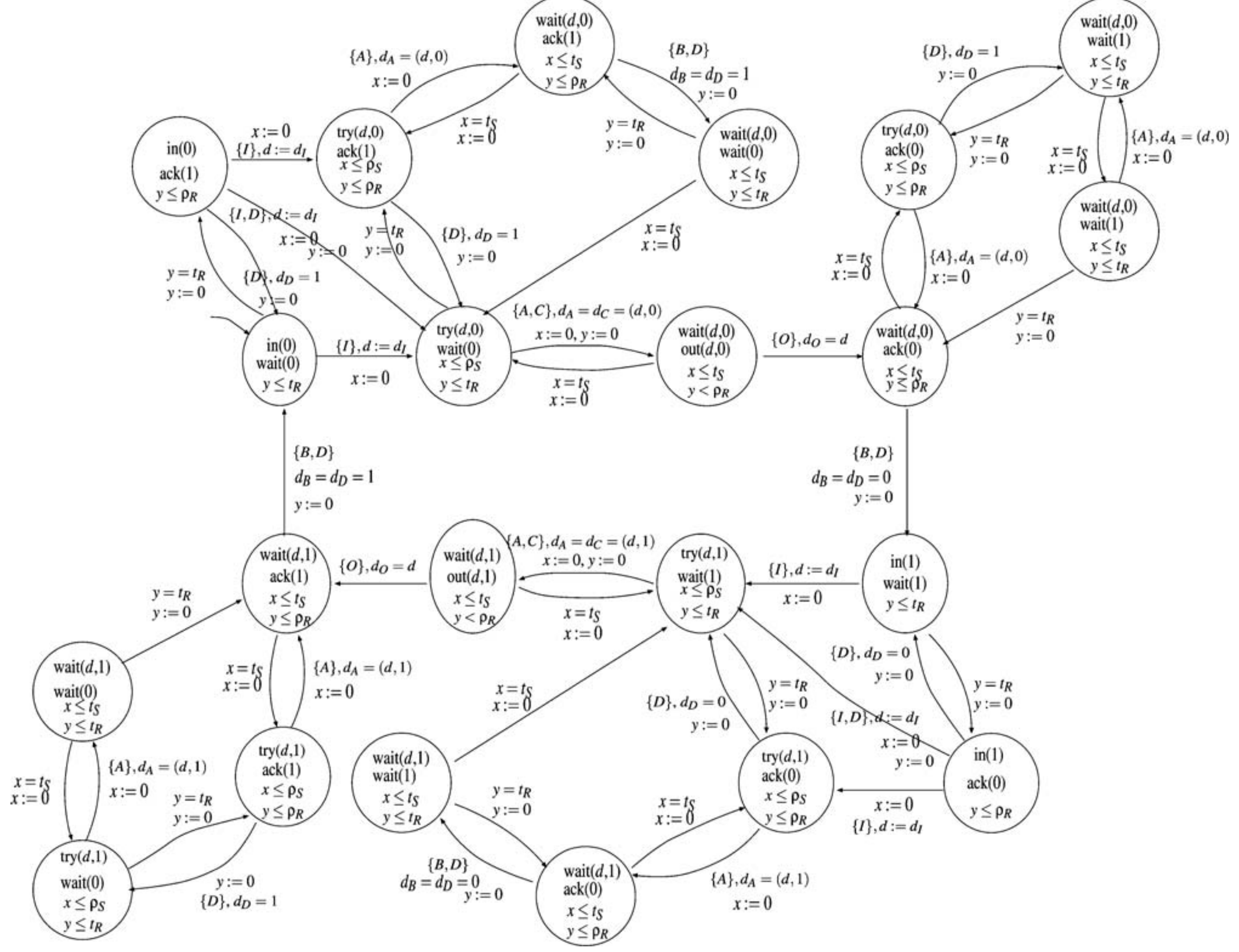

wait(d,0) ack(1) x≤t_S y≤ρ_R
{A}, d_A = (d,0) x:=0
{B,D} d_B = d_D = 1 y:=0
in(0) ack(1) y≤ρ_R
x:=0 {I}, d:=d_I
try(d,0) ack(1) x≤ρ_S y≤ρ_R
x=t_S x:=0
y=t_R y:=0
wait(d,0) wait(0) x≤t_S y≤t_R
{I,D}, d:=d_I x:=0 y:=0
y=t_R y:=0
{D}, d_D = 1 y:=0
in(0) wait(0) y≤t_R
{I}, d:=d_I x:=0
try(d,0) wait(0) x≤ρ_S y≤t_R
{A,C}, d_A = d_C = (d,0) x:=0, y:=0
wait(d,0) out(d,0) x≤t_S y<ρ_R
{O}, d_O = d
wait(d,0) ack(0) x≤t_S y≤ρ_R
{D}, d_D = 1 y:=0
wait(d,0) wait(1) x≤t_S y≤t_R
try(d,0) ack(0) x≤ρ_S y≤ρ_R
{A}, d_A = (d,0) x:=0
wait(d,0) wait(1) x≤t_S y≤t_R
{B,D} d_B = d_D = 0 y:=0
wait(d,1) ack(1) x≤t_S y≤ρ_R
{O}, d_O = d
wait(d,1) out(d,1) x≤t_S y<ρ_R
{A,C}, d_A = d_C = (d,1) x:=0, y:=0
try(d,1) wait(1) x≤ρ_S y≤t_R
{I}, d:=d_I x:=0
in(1) wait(1) y≤t_R
wait(d,1) wait(0) x≤t_S y≤t_R
{A}, d_A = (d,1) x:=0
try(d,1) ack(1) x≤ρ_S y≤ρ_R
wait(d,1) wait(1) x≤t_S y≤t_R
{D}, d_D = 0 y:=0
{I,D}, d:=d_I x:=0 y:=0
try(d,1) ack(0) x≤ρ_S y≤ρ_R
in(1) ack(0) y≤ρ_R
try(d,1) wait(0) x≤ρ_S y≤t_R
{D}, d_D = 1 y:=0
{B,D} d_B = d_D = 0 y:=0
wait(d,1) ack(0) x≤t_S y≤ρ_R
{A}, d_A = (d,1) x:=0

Fig. 4 TCA $\mathcal{T}_{ABP}$ for the ABP

and 0 if $\Theta = \varepsilon$. We write $TSDS(\mathcal{N})$ or simply $TSDS$ to denote the set of all TSD streams for the node-set $\mathcal{N}$. □

Notation 2.8 (TSDS-language of a TCA) If $\mathbf{q}$ is a run in a TCA $\mathcal{T}$ as above then the induced TSD stream $\Theta(\mathbf{q}) = (N_{i_0}, \delta_{i_0}, \bar{t}_{i_0}), (N_{i_1}, \delta_{i_1}, \bar{t}_{i_1}), \ldots$ is obtained from $\mathbf{q}$ by (1) removing all transitions in $\mathbf{q}$ with the empty node set, (2) building the projection on the transition labels, and (3) replacing the sojourn times t_i by the absolute time points $\bar{t}_i = t_0 + \cdots + t_i$. The generated language of a state q in $\mathcal{A}_{\mathcal{T}}$ is

$$\mathcal{L}(\mathcal{T}, q) = \{\Theta(\mathbf{q}) : \mathbf{q} \text{ is a maximal q} - \text{run}\}.$$

The language $\mathcal{L}(\mathcal{T})$ consists of all TSD streams $\Theta(\mathbf{q})$ where $\mathbf{q}$ is a maximal and initial run. □

For instance, the language of the timed sequencer in Fig. 2 consists of all TSD streams $\Theta = ((N_i, \delta_i, \bar{t}_i))_i$ where $N_i \in \{\{A\}, \{B\}\}$ and $\bar{t}_{i+1} - \bar{t}_i > 3$ if $N_{i+1} = N_i$.

3 A Reo primer

Reo [6] is a channel-based exogenous coordination model wherein complex coordinators, called *connectors*, are compositionally built out of simpler ones. The simplest connectors in Reo are a set of *channels* with well-defined behavior supplied by users. Components can instantiate, compose, connect to, and perform I/O operations through connectors. Here, as in [7,9], we do not consider the dynamic creation, composition, and reconfiguration of connectors by components. We restrict our attention to connectors that have a static graphical representation as a *Reo circuit* which coordinates the data-flow through the channels connecting the input /output ports of components.

Channels. Reo's notion of *channel* is far more general than its common interpretation and allows for any primitive communication medium with exactly two ends. The channel ends are classified as *source* ends through which data enter and *sink* ends through which data leave a channel. Although Reo allows for an open-ended set of channel-types with user-defined semantics, for our purposes in this paper, we restrict ourselves to only a small set of channel-types, defined below.

The simplest form of an asynchronous channel is a *FIFO channel* with one buffer cell (called a 1-bounded FIFO channel or simply a FIFO1 channel). It has a source- and a sink-end. We graphically represent a FIFO1 channel by a small box in the middle of an arrow. The buffer is assumed to be initially empty if no data item is shown in the box. The graphical representation of a FIFO1-channel whose buffer initially contains a data element *d* shows *d* inside the box. FIFO channels with two or more buffer cells can be produced by composing several FIFO1 channels, as for instance, explained in [7,9].

A *synchronous channel* (depicted as a simple solid arrow) has a source- and a sink-end, and no buffer. It accepts a data item through its source end iff it can simultaneously dispense it through its sink. A *lossy synchronous channel* (depicted as a dashed arrow) is similar to a synchronous channel, except that it always accepts all data items through its source end. If it is possible for it to simultaneously dispense the data item through its sink (e.g., there is a take operation pending on its sink) the channel transfers the data item; otherwise the data item is lost.

More exotic channels permitted in Reo are synchronous and asynchronous *drains* that have two source ends. Because drains have no sink end, no data value can ever be obtained from these channels. Thus, a synchronous drain accepts a data item through one of its ends iff a data item is also available for it to simultaneously accept through its other end as well. All data accepted by this channel are lost. An asynchronous drain accepts and loses data items through its two source ends, but never simultaneously. Synchronous and asynchronous *spouts* are duals of their corresponding drain channel types, as they have two sink ends.

Reo circuits. A complex connector has a graphical representation, called a *Reo circuit*, as a finite graph where the *nodes* are labeled with pairwise disjoint, non-empty sets of channel ends and where the edges represent the established channels. The major operations for creating Reo connector circuits are join and hide.

To construct a Reo circuit, we start with several instances of basic channels and organize them in a graph where initially each channel end constitutes a separate node, and each pair of nodes is connected by an edge representing its respective channel. We then apply a series of join operations, each of which takes as input two nodes A and B and combines them into a new node C. In this way, several channel ends may coincide on one node.

Reo nodes are *not* physical locations nor represent components. A node is a fundamental concept in Reo representing an important topological property: coincidence of its channel ends. As described below, this property entails specific implications in Reo regarding the flow of data among the channel ends that coincide on a node, irrespective of concern for their locations or any component that may perform I/O operations on that node.

The set of channel ends coincident on a node A is disjointly partitioned into the sets $\mathsf{Src}(A)$ and $\mathsf{Snk}(A)$, denoting the sets of source and sink channel ends that coincide on A, respectively. A node is called a *source node* if $\mathsf{Src}(A) \neq \emptyset \wedge \mathsf{Snk}(A) = \emptyset$. Analogously, A is called a *sink node* if $\mathsf{Src}(A) = \emptyset \wedge \mathsf{Snk}(A) \neq \emptyset$. Node A is called a *mixed node* if $\mathsf{Src}(A) \neq \emptyset \wedge \mathsf{Snk}(A) \neq \emptyset$.

Intuitively, source nodes of a circuit are analogous to the input ports, and sink nodes to the output ports of a component, while mixed nodes are its hidden internal details. Components cannot connect to, read from, or write to mixed nodes. Instead, data-flow through mixed nodes is totally specified by the circuits they belong to.

A component can write data items to a source node of a Reo circuit that it is connected to. A write operation succeeds only if all (source) channel ends coincident on the node accept the data item, in which case the data item is transparently written to every source end coincident on the node. A source node, thus, acts as a *replicator*.

A component can obtain data items from a sink node of a Reo circuit that it is connected to through input operations.[1] A take operation succeeds only if at least one of the (sink) channel ends coincident on the node offers a suitable data item; if more than one coincident channel end offers suitable data items, one is selected nondeterministically. A sink node, thus, acts as a nondeterministic *merger*.

A mixed node is a self-contained "pumping station" that combines the behavior of a sink node (merger) and a source node (replicator) in an atomic iteration of an endless loop: in every iteration a mixed node nondeterministically selects and takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. A data item is suitable for selection in an iteration only if it can

[1] We consider only the destructive take operation here which, e.g., on a FIFO channel, reads and removes the first data item in its buffer.

be accepted by all source channel ends that coincide on the mixed node.

Reo nodes contain no memory. While a component that performs a write operation on a source node may suspend (if the circuit that the node belongs to is not ready to allow the write to succeed), holding the value in its blocked write operation (indefinitely or until an optional time-out specified in the write operation), a Reo node cannot "hold" or represent any data. All data transfer through a Reo node is strictly synchronous (i.e., atomic).

The hide operator allows to create "components" by putting a thick box around a circuit. This insulates all mixed nodes of the circuit inside the box and allows access to its sink and source nodes only, which are placed on the border of the box. The idea is that mixed nodes are internal to the component and no other component can modify or connect to them. Formally, we make hidden (mixed) nodes invisible and abstract their names away.

Example 3.1 (Exclusive router and shift-lossy FIFO1 channel) Fig. 5a shows an implementation of an exclusive router built by composing five synchronous channels, two lossy synchronous channels and a synchronous drain. The intuitive behavior of this circuit is that through its source node A, it obtains a data item d from its environment and delivers d to one of its sink nodes B or C. If both B and C are willing to accept d then the exclusive router nondeterministically decides to deliver d to either B or C.

The key to understanding the behavior of this circuit is that for data flow to occur at A, data flow must synchronously also occur at the bottom node of the synchronous drain in Fig. 5a. This is a mixed node, with two sink and one source coincident channel ends. Data flow at this node can occur only if one of the two lossy synchronous channels actually transfers (rather than losing) the data item available at A. This precludes the possibility of both lossy synchronous channels losing this data item, while the merger behavior of the mixed node prevents the possibility of both making a transfer. If data flow is possible at B or C, the merge behavior of the mixed node allows its respective lossy synchronous channel to pass data, forcing the other to lose it. If data flow is possible at both B or C, the merge behavior of the mixed node non-deterministically selects the value available at one of its two sink channel ends, allowing its corresponding lossy synchronous channel to pass, and the other to lose, its data.

The circuit in Fig. 5b shows an implementation of a shift-lossy FIFO1 channel with source node A and sink node B. This implementation uses four synchronous channels, a synchronous drain, a FIFO1 channel whose buffer initially contains a token data item, o, an empty FIFO2 channel, and an instance of the exclusive router of Fig. 5a shown as the box labeled EXR. A shift-lossy FIFO1 channel behaves the same as a FIFO1 channel, except that writing to its source end is never blocked. If at the time of a write operation its buffer is full, the stored data item in the buffer is lost and the new data item replaces it in the buffer.

If the FIFO2 channel in Fig. 5b is not empty and there is a pair of write and take operations pending, respectively, on the nodes A and B, it is possible for this circuit to either (1) lose the contents of the FIFO2 channel and accept the data item through A to replace it, delaying the take on B; or (2) delay the write on A and dispense the contents of the FIFO2 channel through B. The non-deterministic behavior of the EXR circuit used here makes the choice between these two alternatives non-deterministic. Thus, the shift-lossy FIFO1 channel constructed here breaks the tie non-deterministically, when its buffer is full, and data flow is possible at both of its ends (otherwise, i.e., when the FIFO2 channel is empty, or data flow is not possible at A or B, the circuit has no choice). While, generally, we prefer this non-deterministic behavior, it is also possible to construct similar shift-lossy FIFO1 channels that deterministically prefer one of the two alternatives, by replacing the EXR in Fig. 5b with a priority router.

Derivation of the constraint automata representing the observable behavior of each of these Reo circuits as compositions of the constraint automata representing the behavior of the individual primitives used in their respective Reo circuits appears in [9]. □

In spite of its simplicity, the semantics of Reo is indeed very rich, yielding a surprisingly expressive language [6]. For instance, the relational (as opposed to functional) dependencies that result in "propagation of synchrony and exclusion" as well as the way in which the local behavior of, e.g., lossy synchronous channels imposes non-local constraints on a circuit, are already evident in the exclusive router of Fig 5a. Examples of Reo circuits with more interesting behavior can be found elsewhere and the reader is encouraged to see [30] (in [26]) and [7] for the simple, rich, and expressive formal semantics of Reo.

4 Timed Reo circuits

We now extend the set of primitive channels that we use in the Reo framework by adding channels with timing constraints for the enabledness of their I/O-operations.

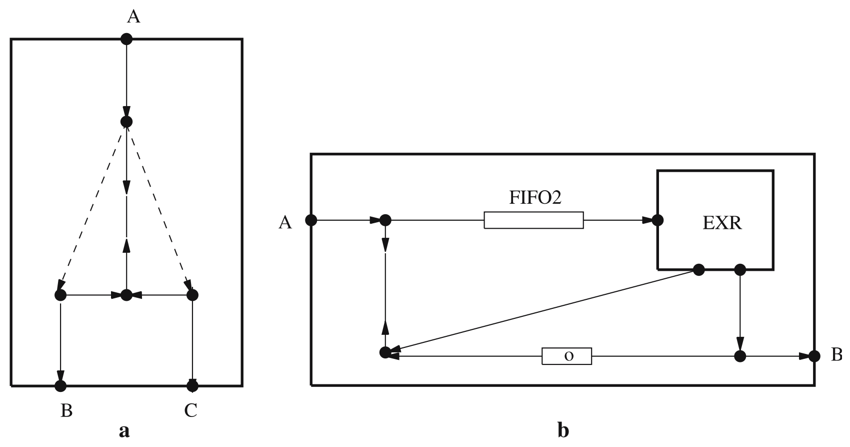


Fig. 5 Exclusive router and shift-lossy FIFO1 channel

We first give some examples for "timed channels" and provide their semantics by means of TCA (Sect. 4.1). Next, we explain how the concepts of join and hiding can be realized with TCA, which yields a compositional way for constructing the TCA for a given Reo circuit with timed channels (Sect. 4.3).

4.1 Untimed and timed primitive channels

Reo defines what a channel is and how channels, as atomic connectors, can be composed into more complex connectors; however, it offers no specific channels. Instead, it allows an open-ended set of user-defined channel types as primitives for constructing connector circuits. This makes it easy to extend Reo circuits to cover timed behavior by introducing a few primitive channels with time-sensitive behavior. In the sequel, we define a number of channel types that we will later use in our timed Reo circuit examples.

FIFO channels. Analogous to a FIFO1 channel, shown on the left-hand-side of the figure below, on the right-hand-side of this figure we show a *timed-lossy* variant of this channel, called *expiring FIFO1*, where a data item is lost if it is not taken out of buffer through the sink end of the channel within t time units after it enters through its source end.

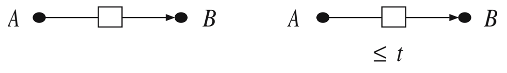


Figure 6 shows the TCA for the FIFO1 and expiring FIFO1 channels. The edge from s to $\bar{s}(d)$ models A's write action. The two edges from $\bar{s}(d)$ to s stand for the event where B takes the message out of the buffer and for the event where the message is lost if B's read action does not occur before t time units after A's write action. The loop at location $\bar{s}(d)$ covers the case where A puts a message d into the buffer at some point in time ϑ, followed by the next message d' exacty three time units later (at time $\vartheta + 3$). In this case, the old message d is replaced by d' and the TCA moves from location $\bar{s}(d)$ to $\bar{s}(d')$ without passing through location s.

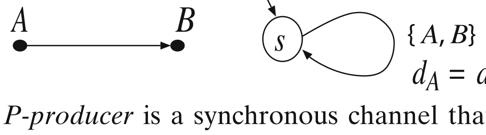
If we skip the loop at $\bar{s}(d)$ the TCA will have a different behavior. The modified TCA, shown later in Example 4.1 (Fig. 12), is called a TCA for an *expiring FIFO1 channel with delay*. For instance, if B is never enabled to take an element out of the buffer then the original TCA allows A to write at time points $0, t, 2t, 3t, \ldots$, while the TCA for an expiring FIFO1 channel with delay requires some delays between the loss of the stored message (where the TCA moves from $\bar{s}(d)$ back to s) and A's next write operation. Formally,

$$(\{A\}, d_A = d, 0), (\{A\}, d_A = d, t), (\{A\}, d_A = d, 2t),$$
$$(\{A\}, d_A = d, 3t), \ldots$$

is in the TSDS-language of the TCA shown on the right of Fig. 6, but not of the TCA for an expiring FIFO1 channel with delay.

Synchronous channels. In the examples we discuss later, we use different types of synchronous channels. Here, we briefly explain their behavior and show how they can be modelled by TCA. The TCA for these synchronous channels do not have proper timing constraints (and do not use any clock).

We start with a standard synchronous channel, depicted as a solid arrow, where the write and take operations must synchronize. The behavior of a (standard) synchronous channel, is formalized by a TCA with a single location:



A *P-producer* is a synchronous channel that, like a standard synchronous channel, allows write and take operations to succeed atomically on its source and sink ends, respectively, except that the value dispensed

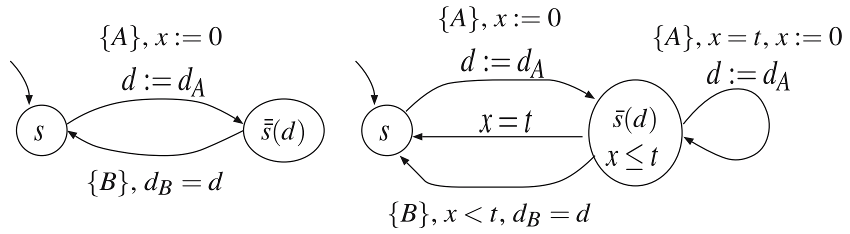
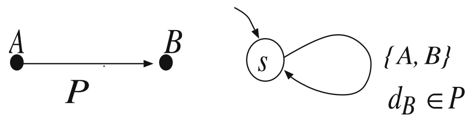
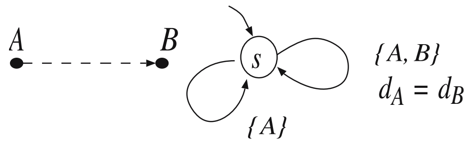


Fig. 6 TCA for a standard and an expiring FIFO1 channel

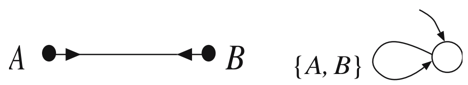
through this channel's sink end is always a data element $d \in P$, regardless of the value it consumes through its source end. If $|P| \geq 2$ then the dispended data element in P is chosen non-deterministically.



The figure below shows a TCA that captures the general "possible" behavior of a *lossy synchronous* channel. To model the context-sensitive behavior of a lossy channel where the $\{A\}$-transition is impossible if B is ready to synchronize, the concept of priorities can be used. The rough idea is to assign a higher priority to the $\{A, B\}$-edge than to the $\{A\}$-edge stating that A and B must synchronize whenever possible. The technical details of constraint automata with priorities are more difficult and will be presented in the forthcoming paper [8].



The above mentioned types of synchronous channels have one source and one sink ends. An example of a channel with two source ends is a *synchronous drain* that accepts a data item through one of its ends iff a data item is also available for it to simultaneously accept through its other end as well. The values written at the sources of a drain are irrelevant. The picture for a synchronous drain and its TCA is as follows:[2]



Timers. We now describe a few timer channels that can serve to measure the time between two events and produce timeout signals. Each of these timer channels has one source end and one sink end.

The source end of a *t-timer* channel (see Fig. 7) accepts any input value $d \in Data$ and returns on its sink end a timeout signal after a delay of t time units. The intuitive explanation for the loop at state $\bar{s}$ is as for the expiring FIFO1 channel.

A t-timer with the *off-option* allows the timer to be stopped before the expiration of its delay when a special "off" value is consumed through its source end. Similarly, the *reset-option* allows the timer to be reset to 0 after it has been activated when a special "reset" value is consumed through its source end. Figure 8 shows a t-timer with both the reset- and the off-options.

A *timer with early expiration*, shown in Fig. 9, makes the timer produce its timeout signal through its sink and reset itself when it consumes a special "expire" value through its source.

In some cases, it is useful to have a timer that is initially activated. In the graphical representation of this timer, we simply put the word "on" under its circle-symbol. In its TCA, we declare $\bar{s}$ as the initial location (rather than s).

4.2 Examples for timed Reo circuits

Before presenting the formal definitions of the composition operators join and hide on TCA, we provide a few examples for timed Reo circuits. These are obtained by combining channel instances through a series of join and hide operations.

Figure 10 demonstrates how to build a Reo circuit via join and hide.[3] The resulting circuit repeatedly produces a timeout signal through T after t time units unless a data transfer occurs from A to B within that interval. Mixed node I serves as an initializer which activates the timer. Either A and B synchronize before the timer expires or the timeout signal occurs at T (after exactly t time units). In either case, the buffer is refilled and the whole procedure restarts.

In (timed) constraint automata models of Reo circuits, locations stand for the configurations of the circuits (e.g., contents of the FIFO channels) while transitions stand for the possible data-flow at one time instance and its effect on the configuration. Intuitively, if we regard a circuit itself as a component, the source nodes of the

[2] Recall that we skip valid data constraints. That is, in the TCA for the synchronous drain the data constraint true is not mentioned in the label of the edge.

[3] In this picture, the buffer of the FIFO channel between F and I is initially filled with the data item 0. The corresponding TCA is as shown on the left of Fig. 6, except that $\bar{s}(0)$ serves as starting location.

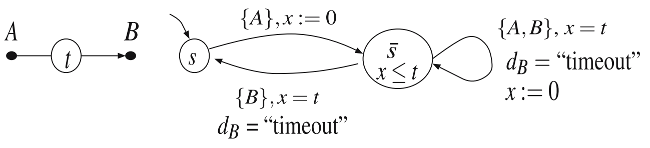


Fig. 7 *t*-timer and its TCA

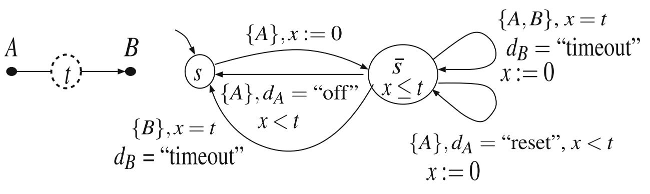


Fig. 8 *t*-timer with off- and reset-option and its TCA

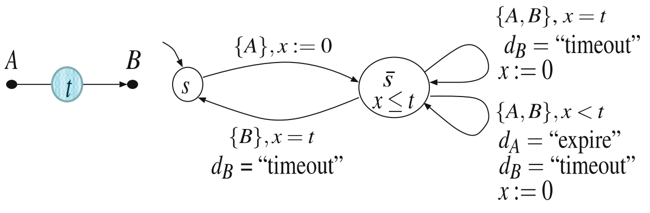


Fig. 9 *t*-timer with eraly expiration and its TCA

circuit act as the input ports, and its sink nodes as the output ports of the component. The data-flow through mixed nodes is totally specified by the circuit.

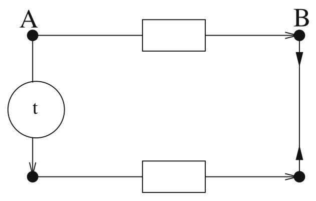
There is a subtle difference between the roles of the sink and source nodes on the one hand and that of the mixed nodes on the other. If an edge contains at least one sink or source node A then the transition must be regarded as conditional: it can be taken if and only if the environment that controls the data-flow at node A (the component that uses A as an in- or output port) performs the corresponding I/O-operation. On the other hand, any transition with a node-set consisting of mixed nodes only can be taken without any involvement by the environment.

Example 4.1 (Expiring FIFO1 channel) Figure 11 shows how an expiring FIFO1 channel with delay can be constructed out of a standard FIFO1 channel and a timer set to expire after *t* time units.

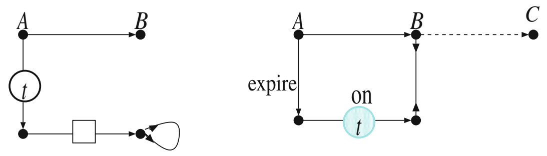
A successful write to A fills up the buffer of the FIFO1 channel CD, and (re)sets the timer channel FG. Another write to A will suspend until the FIFO1 channel becomes empty. While it is full, two things can happen: (1) the timer may expire, and (2) a take can be performed on B. If the timer expires, nodes G, H, and D can fire. GH acts as a synchronous channel and DH accepts but loses the data at D. So the value in the FIFO1 channel gets lost in the drain DH. A take on B will replicate the value in the FIFO1 channel at D and again at E. One copy goes out through B to satisfy the take. The other two copies get lost in the drain DH. Now that the FIFO1 channel is empty and the timer is still running, two things can happen: (3) there is a new write on A, and (4) the timer expires. If there is a new write on A, it succeeds and resets the timer, and we are back to the first case we considered. If the timer expires while the buffer is empty, then its token is accepted and lost in the lossy synchronous channel GH. The special case where the take on B happens at exactly the same time when the timer expires is non-deterministically resolved by the merger behavior of the node H. It either accepts the timer's token from G, or the copy of the data item from E. If it accepts the timer's token first, then it is as if the take has been performed after the expiration of the timer. If it takes the data item first, it is as if the timer expired after the take (which means the timer's token gets lost in the GH channel).

The TCA in Fig. 12 yields a formalization of the above explanation for the possible data flow in the Reo circuit of Fig. 11, after hiding all mixed nodes, i.e., all nodes execpt for A and B.

Example 4.2 (Lower and upper time bounds for I/O-operations) Below we have a circuit that ensures the lower bound "$>$ *t*" for a take operation on B; it yields a FIFO1 channel that guarantees every data item will remain in its buffer at least *t* time units.



We may also control the frequency of data transfer in synchronous channels with time-constrained channels. In the following figure, on the left, data-flow from A to B is possible only once every $\geq t$ time units.



The *t*-timer with early expiration in the circuit on the right ensures that as long as data items are available at A, they will be consumed at least once every *t* time units. Whenever a take operation is performed on C, the data item available at A is transferred through B to C via the synchronous and the lossy synchronous channels that connect these nodes. The transfer at A simultaneously produces an "expire" signal (through the

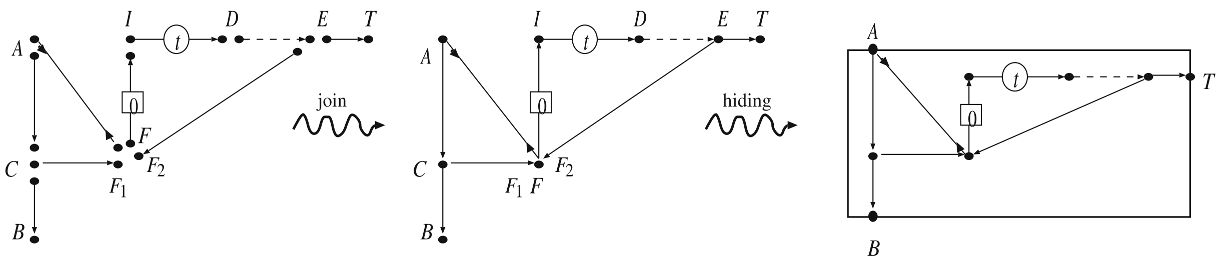


Fig. 10 Example construction of a Reo circuit

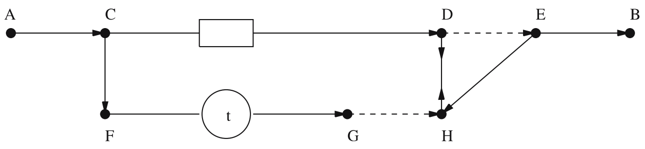


Fig. 11 Reo circuit for an expiring FIFO1 channel

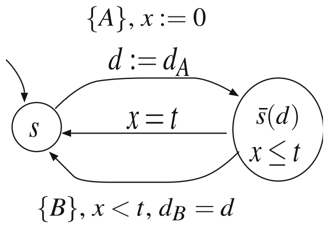


Fig. 12 TCA for an expiring FIFO1 channels

P-producer connected to A, where P is the singleton data set {expire}) which prematurely fires the timer channel, enabling the synchronous drain to allow the data transfer at B. If no take operation occurs at C, the timer produces its timeout-signal after t time units, enabling the transfer of a data item from A to B, because the lossy synchronous channel at B always accepts (and in this case loses this data item). (Because the two ends of the timer always have to synchronize in this circuit, the assumption that the timer is initially on is essential, since otherwise it can never be started.) □

Example 4.3 (Timed sequencer) The timed sequencer in Fig. 2 can be realized by the Reo circuit shown in Figure 13 (and hiding all nodes except for A and B). Here, we use a t-timer with early expiration which is assumed to be initially switched on. A can transfer a value only if D simultaneously also takes a value from the upper buffer. The expiring FIFO1 channel allows this to happen only at some point in time $t_0 < t$. If this happens, an expire-signal is sent (via the P-producer from D to G where P is the singleton data set {expire}) which forces the timeout-signal to become available at H. Because the buffer of the left FIFO1 channel is full and it is connected at E through a synchronous drain and a lossy synchronous channel via J to H, the availability of the timeout-signal at H triggers the synchronous transfer of the contents of the left FIFO1 channel into the right FIFO1. The replication behavior of H also attempts to simultaneously write a copy of the timeout-signal into the top lossy synchronous channel connected to H. However, because at this point in time (i.e., t_0), there is no data available at C, the synchronous drain connected to C prevents I from participating in the transfer of this copy of the timeout-signal from H; therefore, the lossy synchronous channel connecting H to I loses this data. At this point, the same behavior symmetrically repeats with B.

If A has no value to transfer within the first t time units then D does not transfer the data element out the buffer but the timeout signal becomes available at H at time t. Simultaneously, the message in the buffer of the upper expiring FIFO1 channel is lost. At this point in time (i.e., t), there is no data available at C, and the synchronous drain connected to C prevents I from participating in the transfer of a copy of the timeout-signal from H; the lossy synchronous channel connecting H to I loses this data. On the other hand, node E can take the data element out of the buffer of the left FIFO1 channel. Also G is ready to start the timer again. Thus, H synchronizes with the nodes J, E and G which yields a configuration symmetric to the initial one with B instead of A.

Fig. 14 shows the TCA (before hiding) where we skip the data constraints.[4]

Remark 4.4 (Time-constraints for the I/O-operations) In the Reo circuit in Fig. 15, node B is a mixed node which is "always" ready to consume a message from the buffer of the expiring FIFO1 channel because the

[4] In addition to the node-names used in the circuit, we use the names G_E, G_C, G_D and G_F to make clear which take-operation is performed on node G. Such auxiliary names will also be used in the compositional approach to model the merge semantics.

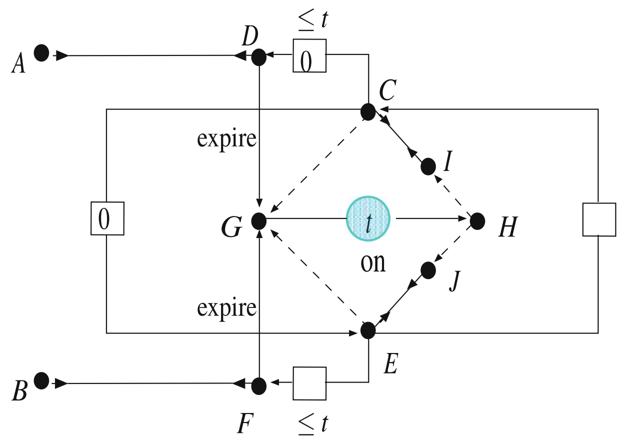
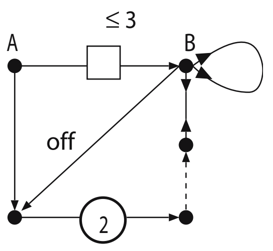


Fig. 13 Reo circuit for a timed sequencer

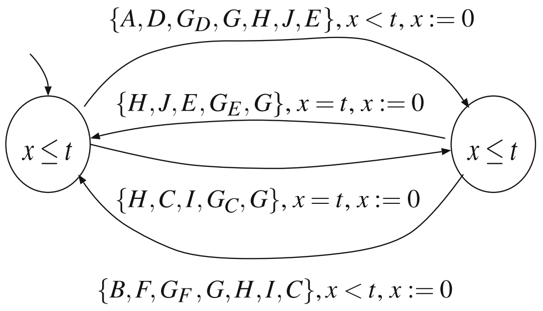
synchronous drain on its right is "always" ready to dispose of any value.

The TCA for this circuit has a TSD stream of the form $(\{A\},[A \mapsto d],0),(\{A\},[A \mapsto d],4),(\{A\},[A \mapsto d],8),\ldots$ where A continuously transfers data items into the buffer of the expiring FIFO1 channel, which in turn loses them all because the data transfer at B takes longer than the specified expiration bound of 3 time units (e.g., because the synchronous drain is too slow). In fact, the above circuit makes no assumptions about the possible delay of B's data transfer operation. Its TCA involves an enabled transition with a node-set consisting of a mixed node with an unbounded delay.

One possibility to avoid such scenarios is to assign *deadlines* to edges $e = (s,N,dc,cc,C,\bar{s})$ where N consists of mixed nodes. For instance, assigning a deadline of 2 to the $\{B\}$-edge in the above example ensures that all values transferred by A are eventually taken out of the buffer by B. However, the timing behavior of the nodes (deadlines or lower time bounds for I/O-operations) can also be made explicit at the syntax level of Reo circuits, using an appropriate combination of Reo's timed channels. For instance, the deadline of 2 in the above example can be guaranteed by a 2-timer with the off-option as follows:



□



{A, D, G_D, G, H, J, E}, x < t, x := 0
{H, J, E, G_E, G}, x = t, x := 0
x ≤ t
x ≤ t
{H, C, I, G_C, G}, x = t, x := 0
{B, F, G_F, G, H, I, C}, x < t, x := 0

Fig. 14 TCA for the timed sequencer

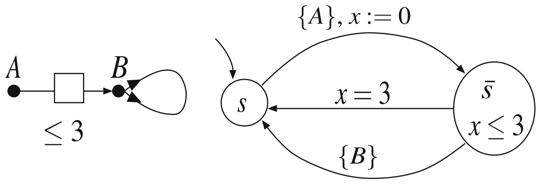


Fig. 15 When does B perform a take-operation?

4.3 Join and hide on TCA

The examples provided in the previous subsection served to illustrate the Reo framework for composing component connectors out of channel instances via join and hide. We now provide composition operators on TCA that capture the meaning of Reo's join and hide operators and that can serve to construct the TCA for a Reo circuit in a compositional way.

Join (replicator semantics). We start with the join operator on TCA which captures the replicator semantics of source (or mixed) nodes. It can serve as the semantic operator for the join of two nodes where at least one of them is a source node. We assume that we are given the TCA $\mathcal{T}_1$ and $\mathcal{T}_2$ for two fragments R_1 and R_2 of a Reo circuit and that we want to perform the join operations for the nodes B_i (in $\mathcal{T}_1$) and $\tilde{B}_i$ (in $\mathcal{T}_2$), $i = 1,\ldots,n$, where at least one of the nodes B_i or $\tilde{B}_i$ is a source node (i.e., has no coincident sink channel end). We first rename $\tilde{B}_i$ into B_i and then apply the following join operator to $\mathcal{T}_1$ and $\mathcal{T}_2$.

Definition 4.5 (Join for TCA) *Given two TCA $\mathcal{T}_i = (S_i,\mathcal{C}_i,\mathcal{N}_i,\mathcal{E}_i,S_{0,i},ic_i)$, $i = 1,2$, with disjoint clock sets, i.e., $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$, the product $\mathcal{T}_1 \bowtie \mathcal{T}_2$ is defined as a TCA with the location space $S = S_1 \times S_2$, the set $S_0 = S_{0,1} \times S_{0,2}$ of initial locations, the node-set $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$, and the clock set $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. The location invariance is given by $ic(\langle s_1,s_2\rangle) = ic_1(s_1) \wedge ic_2(s_2)$. The edge relation $\mathcal{E}$ is obtained through the following rules. The first rule concerns the "synchronization case" where two edges with common nodes are combined as well as the case where* *two edges with non-empty "local" node-sets are taken simultaneously:*

$$\frac{\begin{array}{l}(s_1, N_1, dc_1, cc_1, C_1, \bar{s}_1) \in \mathcal{E}_1,\\ (s_2, N_2, dc_2, cc_2, C_2, \bar{s}_2) \in \mathcal{E}_2,\\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1,\ N_1 \neq \emptyset,\ N_2 \neq \emptyset, dc_1 \wedge dc_2 \neq \mathsf{false}\end{array}}{(\langle s_1, s_2\rangle, N_1 \cup N_2, dc_1 \wedge dc_2, cc_1 \wedge cc_2, C_1 \cup C_2, \langle \bar{s}_1, \bar{s}_2\rangle) \in \mathcal{E}}.$$

The second rule applies to edges all of whose involved nodes are local to only one of the automata:

$$\frac{(s_1, N_1, dc_1, cc_2, C_1, \bar{s}_1) \in \mathcal{E}_1,\ N_1 \cap \mathcal{N}_2 = \emptyset,\ s_2 \in S_2}{(\langle s_1, s_2\rangle, N_1, dc_1, cc_1, C_1, \langle \bar{s}_1, s_2\rangle) \in \mathcal{E}}$$

and its symmetric rule. In particular, the latter rule applies to transitions with empty node-sets. □

A correctness result for the join operator is presented in Lemma 4.9 and Corollary 4.10.

Join (merge semantics). To mimic the merge semantics of sink (or mixed) nodes we use the same technique as in [7,9]. To join two nodes A and B where each of them contains at least one sink end we (1) choose a new node-name, say C, and (2) return $\mathcal{T}_{Merger}(A, B, C) \bowtie \mathcal{T}_A \bowtie \mathcal{T}_B$ where $\mathcal{T}_A$ and $\mathcal{T}_B$ are the TCA that model the sub-circuits containing A and B, respectively, and the TCA $\mathcal{T}_{Merger}(A, B, C)$ shown in Fig. 16.

Hide. Hiding a node-set M in a TCA removes all M-nodes from its edges. However, given an edge with a node-set consisting of M-nodes only, we must ensure that this edge can be taken only after some positive delay. We model this by using an additional clock.

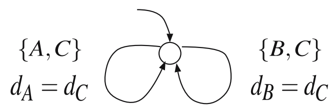


Fig. 16 A merger and its TCA $\mathcal{T}_{Merger}(A, B, C)$

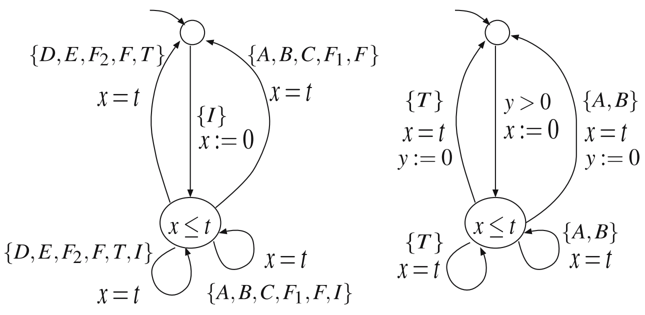


Fig. 17 TCA for the circuit in Figure 10 before and after hiding

Definition 4.6 (Hide for TCA) *Given a TCA $\mathcal{T} = (S, \mathcal{C}, \mathcal{N}, \mathcal{E}, S_0, ic)$, a new clock $y \notin \mathcal{C}$, and $M \subseteq \mathcal{N}$, we define $\exists M[\mathcal{T}] = (S, \mathcal{C} \cup \{y\}, \mathcal{N} \setminus M, \mathcal{E}', S_0, ic)$ where $\mathcal{E}'$ is obtained by the rule:*

$$\frac{(s, N, dc, cc, C, \bar{s}) \in \mathcal{E},\ (N = \emptyset \vee N \setminus M \neq \emptyset)}{(s, N \setminus M, \bigvee_{\delta \in DA(M)} dc[A/\delta_A : A \in M], cc, C \cup \{y\}, \bar{s}) \in \mathcal{E}'}$$

$$\frac{(s, N, dc, cc, C, \bar{s}) \in \mathcal{E},\ \emptyset \neq N \subseteq M}{(s, \emptyset, \mathsf{true}, cc \wedge (y > 0), C \cup \{y\}, \bar{s}) \in \mathcal{E}'}.$$

Here, $dc[A/\delta_A : A \in M]$ is derived from dc by the syntactic replacement of the term d_A with the value $\delta_A \in Data$ for all $A \in M$. (More precisely, we replace "$d_A \in P$" with true *or* false*, depending on whether or not δ_A belongs to P.)* □

Example 4.7 The TCA for the circuit in Fig. 10 can be obtained by joining the TCA for all of its involved channels together with $\mathcal{T}_{Merger}(F_1, F_2, F)$. The resulting TCA before and after hiding are shown in Fig. 9 (For simplicity, we skip the data constraints and irrelevant resettings of y).

We state the correctness of the join and hide operators on TCA by means of their TSDS-languages (see Notation 2.8). For this, we define join and hide operators on TSDS-languages and establish a compositionality result in Lemma 4.9.

Notation 4.8 (Join and hide for TSD-streams and TSDS-languages) Let Θ be a TSD stream over $\mathcal{N}$ and $B \in \mathcal{N}$. The projection $\Theta|_B \in (Data \times \mathbb{R}_{\geq 0})^\infty$ of Θ on B denotes the sequence of pairs $(d, t) \in Data \times \mathbb{R}_{\geq 0}$ that is obtained from Θ by (1) removing all triples (N, δ, t) where $B \notin N$; and (2) replacing any remaining triples (N, δ, t) with the pair (δ_B, t).

- If $M \subseteq \mathcal{N}$ then $\mathsf{hide}(\Theta, M)$ denotes the unique TSD stream $\bar{\Theta} \in TSDS(M)$ such that $\bar{\Theta}|_B = \Theta|_B$ for all $B \in M$.
- Given two TSD streams $\Theta_1 \in TSDS(\mathcal{N}_1)$ and $\Theta_2 \in TSDS(\mathcal{N}_2)$, their join is undefined if there is a node $B \in \mathcal{N}_1 \cap \mathcal{N}_2$ such that $\Theta_1|_B \neq \Theta_2|_B$. Otherwise we define their join $\Theta_1 \bowtie \Theta_2 \in TSDS(\mathcal{N}_1 \cup \mathcal{N}_2)$ as the unique TSD stream such that $(\Theta_1 \bowtie \Theta_2)|_A = \Theta_i|_A$ if $A \in \mathcal{N}_i$.

Given two TSDS-languages $L_1 \subseteq TSDS(\mathcal{N}_1)$ and $L_2 \subseteq TSDS(\mathcal{N}_2)$, their join $L_1 \bowtie L_2 \subseteq TSDS(\mathcal{N}_1 \cup \mathcal{N}_2)$ consists of all TSD streams Θ that can be obtained by joining the TSD streams $\Theta_1 \in L_1$ and $\Theta_2 \in L_2$. If $M \subseteq \mathcal{N}$ and $L \subseteq TSDS(\mathcal{N})$ then $\exists M[L] = \{\mathsf{hide}(\Theta, M) : \Theta \in L\}$. □

The following lemma can be proved using similar arguments as in the untimed case (see [9]):

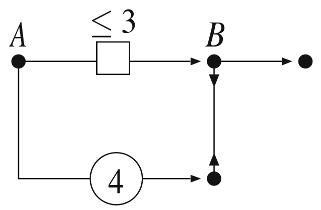


Fig. 18 A Reo circuit with a time-lock

Lemma 4.9 *Let $\mathcal{T}$, $\mathcal{T}_1$ and $\mathcal{T}_2$ be TCA. Then,*

(a) $\mathcal{L}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = \mathcal{L}(\mathcal{T}_1) \bowtie \mathcal{L}(\mathcal{T}_2)$.
(b) $\mathcal{L}(\exists M[\mathcal{T}]) = \exists M[\mathcal{L}(\mathcal{T})]$.

The join of TSDS-languages with the same node-set agrees with their intersection. Thus, we obtain:

Corollary 4.10 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TCA with the same node-set then $\mathcal{L}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = \mathcal{L}(\mathcal{T}_1) \cap \mathcal{L}(\mathcal{T}_2)$.*

4.4 The problem of time-locks in Reo circuits

Of course, using arbitrary combinations of timed channels can lead to TCA with time-locks (see below for an example). However, using (modifications of) standard region- or zone-graph algorithms [2,16] we may check the time-lock freedom of a given Reo circuit.

An example of a Reo circuit with a time-lock is shown in the Fig. 18. Here, A starts the timer and simultaneously puts a data item into the buffer. On the one hand, the synchronous drain forces B to take the data item from the buffer simultaneously with the expiration of the timer (which occurs exactly 4 time units after A's write operation). On the other hand, the data value written by A in the buffer is lost exactly 3 time units after A's write operation. Thus, the write operation at A causes a time-lock.

5 Timed scheduled-data-stream logic

To specify the behavior of timed Reo circuits, one can use a TCA $\mathcal{T}$ and require that the TSD-language generated by a given Reo circuit is contained in $\mathcal{L}(\mathcal{T})$. In this sense, $\mathcal{T}$ specifies the "legal" behavior of the circuit. However, it is often easier to use a logical formalism to express the desired properties rather than using an automata model.

In this section, we introduce Time scheduled-data-stream logic (TSDSL) which is a real-time variant of LTL and allows to reason about the observable data-flow of a Reo circuit by means of the TSD streams generated by its underlying TCA. Instead of the modality $\bigcirc$ (next step), TSDSL uses formulas of the type $\langle\alpha\rangle\varphi$ which consist of a so-called timed scheduled-data expression α and a formula φ. This type of formula is inspired by propositional dynamic logic [12] and extended temporal logic [34]. The timed scheduled-data expressions are variants of timed regular expressions [10] built from atoms of the form $\langle N, dc\rangle$. The TSD expressions specify *sets of finite TSD streams*. The intuitive meaning of $\langle\alpha\rangle\varphi$ is that every initial run has a finite prefix generating a word of the language of α such that φ holds for its corresponding suffix.

5.1 Syntax of TSDSL

In the sequel, we assume a fixed finite and non-empty set $\mathcal{N}$ of nodes. The abstract syntax of TSDSL-formulas is given by the following grammar:

$$\varphi ::= \mathsf{true} \Big| \varphi_1 \wedge \varphi_2 \Big| \neg\varphi \Big| \langle\alpha\rangle\varphi \Big| \varphi_1 \mathsf{U} \varphi_2,$$

where α is a timed scheduled-data expression (TSD expression) built by the grammar:

$$\alpha = \langle N, dc\rangle \Big| \alpha_1 \vee \alpha_2 \Big| \alpha_1 \wedge \alpha_2 \Big| \alpha_1 ; \alpha_2 \Big| \alpha^* \Big| \alpha^I.$$

Here, N is a non-empty node-set, dc a satisfiable data constraint for N, and $I \subseteq \mathbb{R}_{\geq 0} \cup \{\omega\}$ a (possibly unbounded) time interval with its upper-bound in $\mathbb{N} \cup \{\omega\}$. The meanings of $\alpha_1 \vee \alpha_2$ (union, choice), $\alpha_1 \wedge \alpha_2$ (intersection)[5], $\alpha_1;\alpha_2$ (concatenation, sequential composition), and α^* (Kleene closure, finitely many repetitions) are obvious. α^I has the same meaning as α, except for the additional requirement that the total execution time falls in the time interval I.

Intuitively, $\langle\alpha\rangle\varphi$ holds for a TCA iff all its TSD streams have a finite prefix that generates an α-stream and φ holds for its remaining suffix. The dual operator for $\langle\alpha\rangle\varphi$ is $[\![\alpha]\!]\varphi = \neg\langle\alpha\rangle\neg\varphi$ which holds for a TCA iff for each of its TSD streams Θ and all prefixes of Θ that generate an α-word, the formula φ holds for the corresponding suffix of Θ. Other boolean connectives, like disjunction $\vee$ or implication $\rightarrow$, are derived in the usual way.

Remark 5.1 We can also allow for ω-regular TSD expressions that result from adding an ω-operator. Although this increases expressiveness, we skip this option here. In contrast to the real-time extensions of LTL, as, e.g., in [3,5,15], TSDSL does not use time-constrained temporal modalities such as $\mathsf{U}^{\leq t}$. These can be

[5] Standard regular expressions do not contain an intersection operator (although regular languages are closed under intersection). However, as pointed out in [10], in timed settings, the class of timed languages induced by timed regular expressions without an explicit intersection operator is not closed under intersection.

added to TSDSL, but in the examples (see below) it turned out that the time-constraints in the TSD expressions are sufficient to formulate the relevant properties of Reo circuits. □

Simplified notation. We often skip the semicolon for the concatenation operator (i.e., $\alpha\beta$ stands short for $\alpha;\beta$). We simply write $\langle N\rangle$ for $\langle N, \mathsf{true}\rangle$ and often omit brackets: e.g., $\langle A, dc\rangle$ is short-hand for $\langle\{A\}, dc\rangle$ and $\langle\!\langle N\rangle\!\rangle$ for $\langle\!\langle\langle N\rangle\rangle\!\rangle$. We write $\langle\ldots A\ldots\rangle$ to denote the disjunction of the expressions $\langle N\rangle$ where N ranges over all subsets of $\mathcal{N}$ that contain the node A. The construct $\langle\neg A\rangle$ stands for the disjunction of all expressions $\langle N\rangle$ where N ranges over all non-empty node-sets that do not contain A. The construct $\langle\cdot\rangle$ denotes the disjunction of all atoms $\langle N\rangle$ where N is an arbitrary non-empty node-set. The shorthand $\langle\!\langle\cdot\rangle\!\rangle\varphi$ stands for $\langle\!\langle\langle\cdot\rangle\rangle\!\rangle\varphi$. We also often skip true and write $\langle\!\langle\alpha\rangle\!\rangle$ for $\langle\!\langle\alpha\rangle\!\rangle\mathsf{true}$: e.g., the TCA for the normal FIFO1 channel (Fig. 6) satisfies the formula

$$[\![(\langle A\rangle\langle B\rangle)^*]\!]\langle\!\langle A\rangle\!\rangle \wedge [\![(\langle A\rangle\langle B\rangle)^*\langle A\rangle]\!]\langle\!\langle B\rangle\!\rangle$$

which states that the data-flows at nodes A and B alternate, starting with A.

Derived operators. The standard *next step* operator is derived as $\bigcirc\varphi = \langle\!\langle\cdot\rangle\!\rangle\varphi$. In particular, $\bigcirc\mathsf{true}$ asserts the occurrence of some observable data-flow, while $\neg\bigcirc\mathsf{true}$ states that data-flow has stopped. The modalities *eventually* and *always* can be derived as usual by definitions $\Diamond\varphi = \mathsf{true}\mathsf{U}\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. For instance, the following TSDSL formula specifies the behavior of a normal FIFO1 channel (see Fig. 6):

$$\Box\left(\bigwedge_{d\in Data} [\![\langle A, d_A = d\rangle]\!]\langle\!\langle B, d_B = d\rangle\!\rangle\right)$$
$$\wedge\Box(\langle\!\langle B\rangle\!\rangle \to \bigcirc\langle\!\langle A\rangle\!\rangle)$$

The expiring FIFO1 channel in Fig. 6 satisfies the TSDSL formula

$$\Box\left(\bigwedge_{d\in Data} [\![\langle A, d_A{=}d\rangle]\!]\left(\langle\!\langle B, d_B{=}d\rangle^{<t}\rangle\!\rangle \vee \neg\langle\!\langle < \cdot > ^{<t}\rangle\!\rangle\right)\right)$$

which expresses the fact that within t time units after A's write-operation either B takes the element from the buffer or there is no observable data-flow. For the timed sequencer (Fig. 2 and Example 4.3) the following formula holds

$$\Box[\![A]\!]\left(\langle\!\langle B\rangle^{\le t}\rangle\!\rangle \vee \neg\langle\!\langle\langle\cdot\rangle^{\le t}\rangle\!\rangle\right)$$

stating that whenever data-flow is observed at A, within the next t time units there is either data-flow at B or no observable data-flow at all.

The weak variant $\tilde{\mathsf{U}}$ of until is obtained as $\varphi_1\tilde{\mathsf{U}}\varphi_2 = (\varphi_1\mathsf{U}\varphi_2) \vee (\Box\varphi_1)$. For instance, the t-timer with reset-option (but without the off-option) fulfills the formula

$$\Box[\![A]\!]\left(\langle\!\langle\langle A, d_A = \text{reset}\rangle^{<t}\rangle\!\rangle\tilde{\mathsf{U}}\langle\!\langle B, d_B = \text{timeout}\rangle\!\rangle\right).$$

5.2 Semantics of TSDSL

To provide the formal definition of the semantics of TSD expressions and TSDSL-formulas we need some additional notation for working with TSD streams.

Notation 5.2 (Time cuts, concatenation, Kleene closure) Let $\Theta = (N_0, \delta_0, \bar{t}_0), (N_1, \delta_1, \bar{t}_1), \ldots$. be a TSD stream as in Notation 2.7. For a point in time $t \in \mathbb{R}_{\ge 0}$, we define $\Theta \uparrow t$ as the suffix of Θ that ignores every data-flow that occurs before t and formalizes the observable behavior in the time interval $[t, \infty[$. Formally, if Θ is as above then:

- $\Theta \uparrow t = \varepsilon$ if $|\Theta| = k + 1 < \omega$ and $\bar{t}_k < t$.
- $\Theta \uparrow t = (N_k, \delta_k, \bar{t}_k), (N_{k+1}, \delta_{k+1}, \bar{t}_{k+1}), \ldots$ if $|\Theta| = \omega$ and k is the smallest index such that $\bar{t}_k \ge t$.

We use $\Theta \downarrow t$ to denote the TSD stream that describes the data-flow in the time interval $[0, t]$. That is, $\Theta \downarrow t = \varepsilon$ if $\Theta = \varepsilon$ or $t_0 \ge t$. Otherwise, $\Theta \downarrow t = (N_0, \delta_0, \bar{t}_0), \ldots,$ $(N_k, \delta_k, \bar{t}_k)$ where k is the largest index such that $\bar{t}_k < t$.

The concatenation of finite TSD streams is defined as follows. We define $\Theta;\varepsilon = \varepsilon;\Theta = \Theta$. If $\Theta_1 = (N_0, \delta_0, \bar{t}_0),$ $\ldots, (N_n, \delta_n, \bar{t}_n)$ and $\Theta_2 = (M_0, \sigma_0, \bar{\rho}_0), \ldots, (M_m, \sigma_m, \bar{\rho}_m)$ then

$$\Theta_1;\Theta_2 = \left(N_0, \delta_0, \bar{t}_0\right), \ldots, \left(N_n, \delta_n, \bar{t}_n\right), (M_0, \sigma_0, \bar{t}_n + \bar{\rho}_0), \ldots,$$
$$\times\left(M_m, \sigma_m, \bar{t}_n + \bar{\rho}_m\right).$$

If L and $\tilde{L}$ are TSDS-languages with the same node-set $\mathcal{N}$ then $L;\tilde{L} = \left\{\Theta\ ;\ \tilde{\Theta} : \Theta \in L, \tilde{\Theta} \in \tilde{L}\right\}$ and $L^* = \bigcup_{n\ge 0} L^n$ where $L^0 = \{\varepsilon\}$, $L^{n+1} = L^n;L$. □

Semantics of TSD-expressions and TSDSL-formulas. We define $\mathcal{L}(\alpha) \subseteq TSDS$ by structural induction. $\mathcal{L}(\langle N, dc\rangle)$ is the set of all TSD streams of length 1 that have the form (N, δ, t) where $\delta \models dc$. We define $\mathcal{L}(\alpha_1 \vee \alpha_2) = \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$, $\mathcal{L}(\alpha_1 \wedge \alpha_2) = \mathcal{L}(\alpha_1) \cap \mathcal{L}(\alpha_2)$, $\mathcal{L}(\alpha_1;\alpha_2) = \mathcal{L}(\alpha_1);\mathcal{L}(\alpha_2)$, and $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$. The semantics of time-constrained expressions is formalized as

$$\mathcal{L}(\alpha^I) = \{\Theta \in \mathcal{L}(\alpha) : \tau(\Theta) \in I\}.$$

Recall that $\tau(\Theta)$ denotes the execution time of Θ (see Notation 2.7). The satisfaction relation $\models$ for TDSL-formulas and TSD-streams is defined by structural induction as shown in Fig. 19. For the derived $[\![\ldots]\!]$-operator,

$$\begin{array}{lll}
\Theta \models \mathsf{true} & & \\
\Theta \models \varphi_1 \wedge \varphi_2 & \text{iff} & \Theta \models \varphi_1 \text{ and } \Theta \models \varphi_2 \\
\Theta \models \neg\varphi & \text{iff} & \Theta \not\models \varphi \\
\Theta \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff} & \exists\, t \in \mathbb{R}_{\geq 0} \text{ s.t. } \Theta \uparrow t \models \varphi_2 \\
 & & \text{and } \Theta \uparrow \rho \models \varphi_1 \text{ for all } \rho \text{ with } 0 \leq \rho < t \\
\Theta \models \langle\alpha\rangle\varphi & \text{iff} & \exists\, t \in \mathbb{R}_{\geq 0} \text{ s.t. } \Theta \downarrow t \in \mathcal{L}(\alpha) \wedge \Theta \uparrow t \models \varphi
\end{array}$$

Fig. 19 Satisfaction relation for TSDSL-formulas

we obtain

$$\Theta \models [\![\alpha]\!]\varphi \text{ iff for all } t \geq 0 \text{ we have } : \Theta \downarrow t \in \mathcal{L}(\alpha) \times \text{ implies } \Theta \uparrow t \models \varphi.$$

With any TSDSL-formula, we associate a TSDS-language as follows:

$$\mathcal{L}(\varphi) = \{\Theta \in TSDS(\mathcal{N}) : \Theta \models \varphi\}.$$

Logical equivalence $\equiv$ of TSDSL-formulas is defined as usual by $\varphi_1 \equiv \varphi_2$ iff $\mathcal{L}(\varphi_1) = \mathcal{L}(\varphi_2)$. If $\mathcal{T}$ is a TCA and q a state in $\mathcal{A}_{\mathcal{T}}$ then $q \models \varphi$ iff $\mathcal{L}(\mathcal{T}, q) \subseteq \mathcal{L}(\varphi)$. Moreover, we define $\mathcal{T} \models \varphi$ iff $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(\varphi)$.

Remark 5.3 TSDSL as a logic on TSD streams has the power to "separate" all TSD streams Θ_1, Θ_2 where the time-abstract data flows (formalized by the induced sequences of node-set/data-assignment pairs) are different. To see this, we may simply take a prefix $(N_1, \delta_1, \bar{t_1})$, $\ldots, (N_k, \delta_k, \bar{t_k})$ of one of the TSD streams, say Θ_1, such that Θ_2 has no prefix of the form $(N_1, \delta_1, \bar{t_1}'), \ldots, (N_k, \delta_k, \bar{t_k}')$. Then,

$$\Theta_1 \models \langle\langle N_1, \delta_1\rangle; \ldots; \langle N_k, \delta_k\rangle\rangle, \text{ while } \Theta_2 \not\models \langle\langle N_1, \delta_1\rangle; \ldots; \langle N_k, \delta_k\rangle\rangle.$$

Here, the data assigments δ_i are viewed as data constraints. If, however, the time-abstract data flows in Θ_1 and Θ_2 agree then it possible that no TSDSL-formula can distinguish between Θ_1 and Θ_2. The reason for this is that we allow for natural (lower/upper) time bounds in TSDSL-expressions only. For instance, the TSD streams

$$\begin{aligned}
\Theta_1 &= (\{A\}, d_A = d, 0.5), (\{A\}, d_A = d, 1.5), \\
&\quad (\{A\}, d_A = d, 2.5), (\{A\}, d_A = d, 3.5), \ldots \\
\Theta_2 &= (\{A\}, d_A = d, 0.6), (\{A\}, d_A = d, 1.6), \\
&\quad (\{A\}, d_A = d, 2.6), (\{A\}, d_A = d, 3.6), \ldots
\end{aligned}$$

fulfill the same TSDSL-formulas.

5.3 Example: the alternating bit protocol

The properties of the ABP (see Example 2.4 and Fig. 4) can be specified by the formula

$$\varphi_{ABP}(t) = \bigwedge_{d \in Data} \Box[\![\langle I, d_I = d\rangle]\!]\langle(\langle\neg I\rangle^* \langle O, d_O = d\rangle)^{\leq t}\rangle$$

for some time bound t. The formula $\varphi_{ABP}(t)$ states that whenever the sender receives a message d at port I, within its next t time units the receiver will output d at port O during which time the sender does not accept a new input message through port I.[6]

Arbitrary choice of the time-parameters. For an arbitrary choice of the time-parameters t_S, t_R, ρ_S and ρ_R we cannot expect that $\mathcal{T}_{ABP} \models \varphi_{ABP}(t)$. For instance, if $\rho_S \geq \rho_R = 5$ and $t_R = t_S = 2$ then the following behavior is possible. The starting state is $q_0 = \langle \mathrm{in}(0), \mathrm{wait}(0), x = 0, y = 0\rangle$. Let us assume that the first input at I arrives at time instant 3. The invariance condition "$y \leq t_R = 2$" of the receiver-location wait(0) forces the receiver to move to location ack(1) at time instant 2. Thus, we enter state

$$q_1 = \langle \mathrm{in}(0), \mathrm{ack}(1), x = 2, y = 0\rangle.$$

After one time unit (i.e., at time instant 3), the sender takes the input value d from port I which leads to state $q_2 = \langle \mathrm{try}(d, 0), \mathrm{ack}(1), x = 3, y = 1\rangle$. After another time unit (at time instant 4), the sender tries to send $(d, 0)$ through port A and moves to location wait$(d, 0)$. This yields state

$$q_3 = \langle \mathrm{wait}(d, 0), \mathrm{ack}(1), x = 0, y = 2\rangle.$$

At time instant 5, the receiver sends the control bit $b = 1$ which the sender ignores. Thus, we enter the global state

$$q_4 = \langle \mathrm{wait}(d, 0), \mathrm{wait}(0), x = 1, y = 0\rangle.$$

Staying in these location for another time unit leads to the state $\langle \mathrm{wait}(d, 0), \mathrm{wait}(0), x = 2, y = 1\rangle$ where the invariance condition "$x \leq t_S = 2$" of the sender-location wait$(d, 0)$ forces the sender to move to location try$(d, 0)$. We are now in state

$$q_5 = \langle \mathrm{try}(d, 0), \mathrm{wait}(0), x = 0, y = 1\rangle.$$

[6] As input on I can occur simultaneously with the receiver resending its acknowledgment of the previous message via port D, the atom $\langle I, d_I = d\rangle$ can be replaced with the expression $\langle I, d_I = d\rangle \vee \langle\{I, D\}, d_I = d\rangle$.

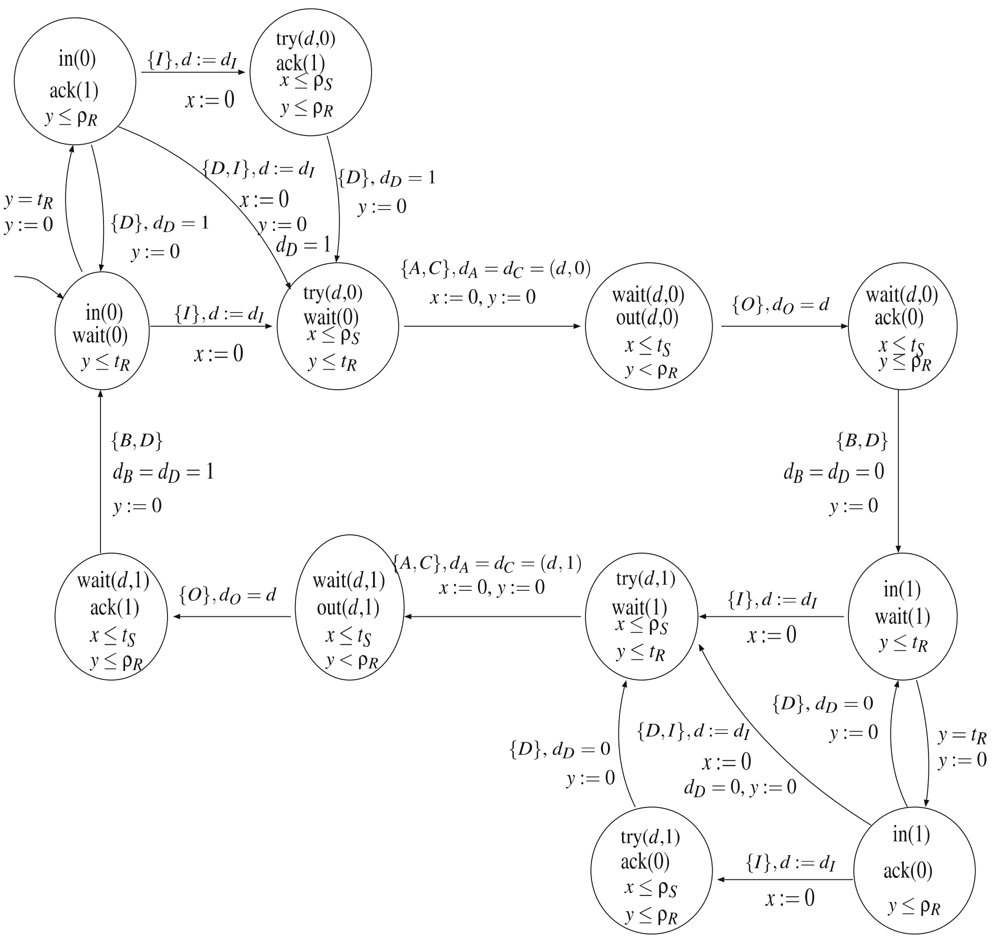


Fig. 20 TCA for the ABP for $\rho_R < \min\{\rho_S, t_S\}$ and $\rho_S < t_R$

One time unit later, clock y has the value 2 and forces the receiver to leave location wait(0). We enter the global state $q_6 = \langle \text{try}(d,0), \text{ack}(1), x = 1, y = 0\rangle$. After waiting for 1 time unit, the sender resends the pair $(d, 0)$ which leads to the global state

$$q_7 = \langle \text{wait}(d,0), \text{ack}(1), x = 0, y = 1\rangle.$$

One time unit later, the receiver resends the control bit 1 which the sender ignores again. We now reenter state q_4 and may continue in the same way, without ever producing an output at port O. Hence, for this choice of the time-parameter we obtain

$$\mathcal{T}_{ABP} \not\models \Box(\langle I\rangle \to \Diamond\langle O\rangle).$$

In particular, there is no t such that $\varphi_{ABP}(t)$ holds for $\mathcal{T}_{ABP}$.

Special choices of the time-parameters. Assuming $\rho_R < \rho_S < t_R$ and $\rho_R < t_S$ then no message sent via the lossy channel connecting A and C will be lost. In fact, it can only happen that the receiver acknowledges more than once the receipt of the last message (because no upper time bound is assumed for the arrival of messages at input port I). The reachable fragment of the TCA is shown in Fig. 20. We obtain

$$\mathcal{T}_{ABP} \models \varphi_{ABP}(\rho_S + \rho_R),$$

stating that the delay for the output at O is bounded above by the maximal sojourn time of the sender in location wait(d, b) plus the maximal delay ρ_R for the receiver to send the acknowledgment after it receives a message through port C. (This is the best bound we can expect.) The fact that messages along the AC channel are never lost can be formalized by the TSDSL formula

$$\neg\Diamond\langle A\rangle$$

which states that it is not possible to observe a data-flow at node A only (i.e., not together with C).

When $\rho_R < t_S < t_R$ and $\rho_S < t_R - t_S$, messages sent from A to C may get lost. However, when A resends the

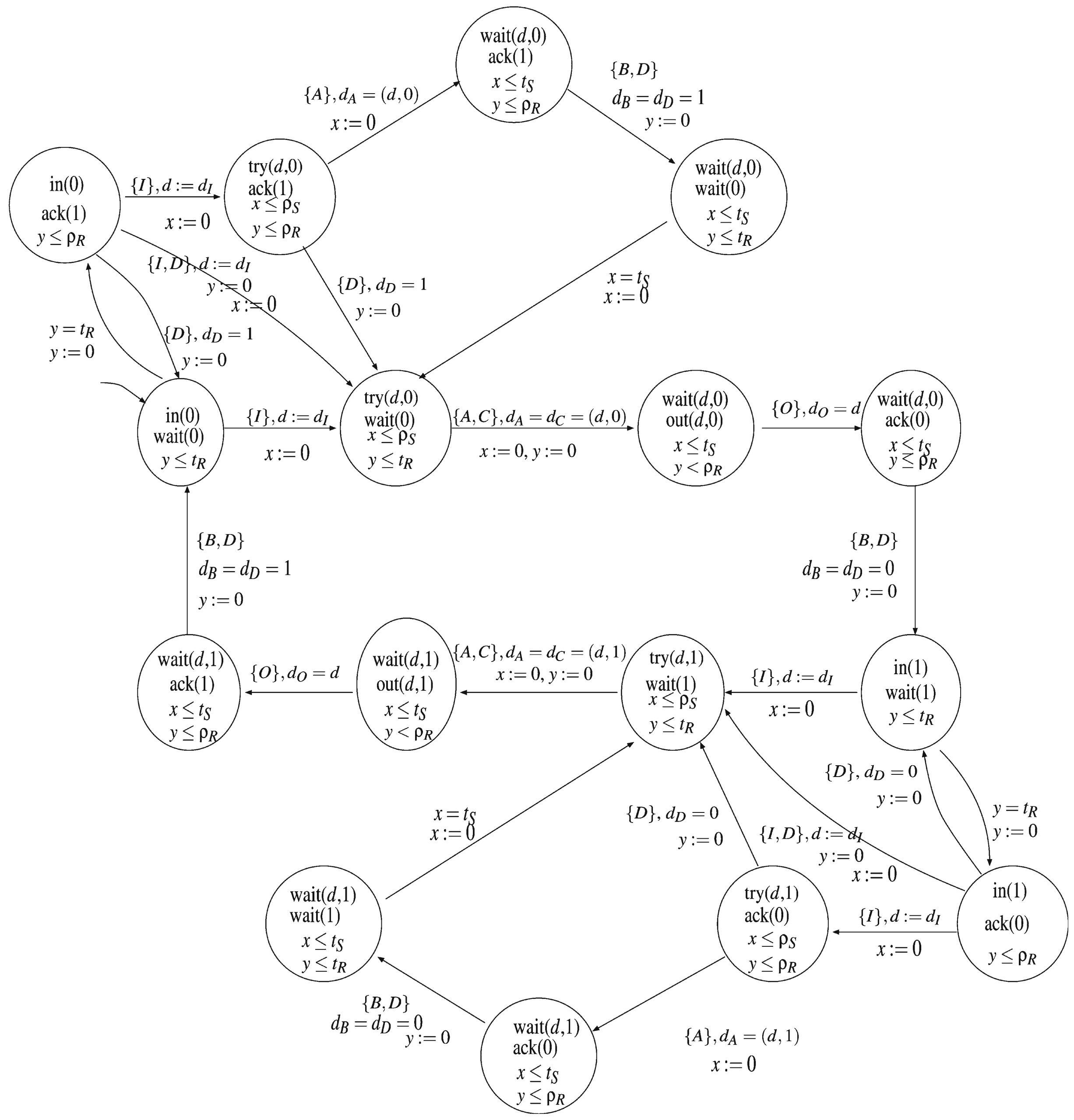

wait(d,0) ack(1) $x \leq t_S$ $y \leq \rho_R$
$\{A\}, d_A = (d,0)$ $x := 0$
$\{B,D\}$ $d_B = d_D = 1$ $y := 0$
in(0) ack(1) $y \leq \rho_R$
$\{I\}, d := d_I$ $x := 0$
try(d,0) ack(1) $x \leq \rho_S$ $y \leq \rho_R$
wait(d,0) wait(0) $x \leq t_S$ $y \leq t_R$
$\{I,D\}, d := d_I$ $y := 0$ $x := 0$
$\{D\}, d_D = 1$ $y := 0$
$x = t_S$ $x := 0$
$y = t_R$ $y := 0$
in(0) wait(0) $y \leq t_R$
try(d,0) wait(0) $x \leq \rho_S$ $y \leq t_R$
$\{A,C\}, d_A = d_C = (d,0)$ $x := 0, y := 0$
wait(d,0) out(d,0) $x \leq t_S$ $y < \rho_R$
$\{O\}, d_O = d$
wait(d,0) ack(0) $x \leq t_S$ $y \leq \rho_R$
$\{B,D\}$ $d_B = d_D = 0$ $y := 0$
wait(d,1) ack(1) $x \leq t_S$ $y \leq \rho_R$
wait(d,1) out(d,1) $x \leq t_S$ $y < \rho_R$
$\{A,C\}, d_A = d_C = (d,1)$ $x := 0, y := 0$
try(d,1) wait(1) $x \leq \rho_S$ $y \leq t_R$
in(1) wait(1) $y \leq t_R$
$\{D\}, d_D = 0$ $y := 0$
$\{I,D\}, d := d_I$ $y := 0$ $x := 0$
wait(d,1) wait(1) $x \leq t_S$ $y \leq t_R$
try(d,1) ack(0) $x \leq \rho_S$ $y \leq \rho_R$
in(1) ack(0) $y \leq \rho_R$
$\{B,D\}$ $d_B = d_D = 0$ $y := 0$
wait(d,1) ack(0) $x \leq t_S$ $y \leq \rho_R$
$\{A\}, d_A = (d,1)$ $x := 0$

Fig. 21 TCA for the ABP for $\rho_R < t_S < t_R$ and $\rho_S < t_R - t_S$

message the receiver accepts the message through port C. In this case, we have

$$\mathcal{T}_{ABP} \models \varphi_{ABP}(\rho_S + \rho_R + t_S),$$

stating that the delay for the output at port O is at most the maximal delay for the sender and receiver to send their messages along the lossy channels connecting them plus the deadline t_S which the sender uses for resending message-bit pairs. The reachable part of the TCA under these assumptions is shown in Fig. 21. The property that a message sent along the AC channel can be lost only once can be formalized by the TSDSL formula.

$$\neg\Diamond\langle\langle A\rangle\langle\neg I\rangle^*\langle A\rangle\rangle.$$

5.4 TSDSL model checking

The TSDSL model checking problem addresses the question of whether $\mathcal{T} \models \varphi$ for a given TCA $\mathcal{T}$ and TSDSL formula φ. We briefly sketch the main ideas of a

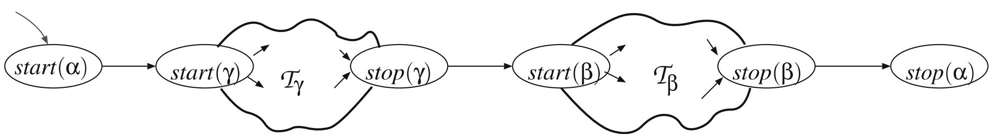


Fig. 22 TCA $\mathcal{T}_{\gamma;\beta}$

TSDSL model checking algorithm that relies on a combination of (slight variants of) standard automata-based model checking algorithms for LTL [14,33,35] and timed regular expressions [10].

The rough idea is to provide an algorithm that disproves the satisfaction of φ for $\mathcal{T}$ and "searches" for a witness for $\mathcal{L}(\mathcal{T}) \not\subseteq \mathcal{L}(\varphi)$, i.e., a TSD stream in $\mathcal{L}(\mathcal{T})$ where φ does not hold. The first step is to switch from φ to $\neg\varphi$ which is then turned into a TCA with Büchi acceptance. Formally, a Büchi TCA denotes a pair $\mathcal{F} = (\mathcal{T}, S_{acc})$ consisting of a TCA $\mathcal{T} = (S, Q, \mathcal{N}, \mathcal{E}, S_0, ic)$ and a set $S_{acc} \subseteq S$ of accepting locations. A q-run in $\mathcal{T}$ is called accepting iff it is either finite and ends in an accepting location or visits infinitely often an accepting location. $\mathcal{L}(\mathcal{F})$ denotes the set of TSD streams that can be generated by an accepting maximal run. (Note that for any TCA $\mathcal{T}$ we have $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{F}_\mathcal{T})$ where $\mathcal{F}_\mathcal{T}$ is the Büchi TCA that results from $\mathcal{T}$ by declaring all locations to be accepting.)

For the given formula $\neg\varphi$, we may apply roughly the same techniques as suggested in [33,34] for extended temporal logic, to construct a Büchi TCA $\mathcal{F}$ with $\mathcal{L}(\mathcal{F}) = \mathcal{L}(\neg\varphi)$. (The main steps for the construction of $\mathcal{F}$ are sketched below.) Then, we have:

$$\mathcal{T} \models \varphi \text{ iff } \mathcal{L}(\mathcal{T} \bowtie \mathcal{F}) = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{F}) = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\neg\varphi) = \emptyset.$$

Assuming disjoint clock sets of $\mathcal{T}$ and $\mathcal{F}$ (otherwise the clocks in $\mathcal{F}$ an be renamed to avoid name clashes), the join-operator $\mathcal{T} \bowtie \mathcal{F}$ yields a Büchi TCA which is obtained through the standard join operator (Definition 4.5) where the accepting locations $\langle s, s'\rangle$ in $\mathcal{T} \bowtie \mathcal{F}$ are those such that location s is an arbitrary location in $\mathcal{T}$ and location s' is an accepting location in $\mathcal{F}$. Finally, we may apply the standard region graph algorithms [2] to check for the emptiness of $\mathcal{T} \bowtie \mathcal{F}$. Note that for the emptiness check the Büchi TCA $\mathcal{T} \bowtie \mathcal{F}$ can be regarded as a standard timed automata à la Alur and Dill. We just need to remove all edges with an unsatisfiable data constraint, and then ignore the node-set/data-constraint labels of the remaining edges.

What remains is to explain how to obtain a Büchi TCA for TSDSL-formulas. We sketch here only the main ideas of this rather complex construction, which essentially uses techniques known from the literature, and put the emphasis on the modifications that are necessary for our purposes. The first step in the construction of $\mathcal{F}$ is to generate a (normal) TCA $\mathcal{T}_\alpha$ for every TSD-expression α that appears in a subformula of $\neg\varphi$ of the form $\langle\alpha\rangle\psi$. These automata $\mathcal{T}_\alpha$ will serve as the basic building blocks for the construction of $\mathcal{F}$.

TCA for TSD-expressions. The TCA $\mathcal{T}_\alpha$ can be constructed in a compositional way. The TCA $\mathcal{T}_\alpha$ has a unique initial location, called *start*(α), and a location *stop*(α) such that $\mathcal{L}(\alpha)$ is the set of all TSD streams Θ that are induced by a finite run in $\mathcal{T}_\alpha$ starting in *start*(α) and ending in *stop*(α).

The construction of the TCA $\mathcal{T}_\alpha$ is by structural induction, essentially as described in [10]. For the atoms $\langle N, dc\rangle$ we use a TCA with two locations *start*(α) and *stop*(α) that are connected via the edge $(start(\alpha), N, dc, \mathsf{true}, \emptyset, stop(\alpha))$. The invariance condition of both locations is true. If α is $\gamma;\beta$ then we use the construction shown in Fig. 22. Here and in the sequel, edges with no label in the figures are assumed to be labelled with $N = \emptyset$,$dc = \mathsf{true}$, $cc = \mathsf{true}$ and $C = \emptyset$.

A similar construction can be used for the choice operator $\alpha = \gamma \vee \beta$ where we use edges from *start*(α) to *start*(γ) and *start*(β) and from *stop*(γ) and *stop*(β) to *stop*(α). See Fig. 23.

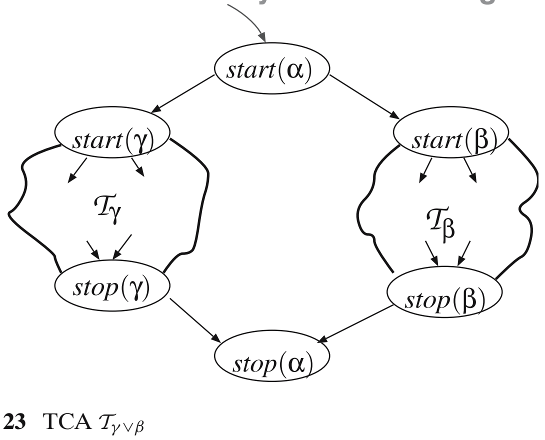


Fig. 23 TCA $\mathcal{T}_{\gamma\vee\beta}$

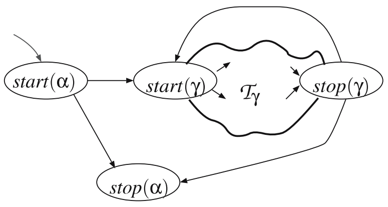


Fig. 24 TCA $\mathcal{T}_{\gamma^*}$

For the Kleene closure $\alpha = \gamma^*$, we may use a similar construction shown in Fig. 24.

The above TCA $\mathcal{T}_\alpha$ do not use any clock. In fact, proper timing constraints are needed only for TSD-expressions with (non-trivial) time bounds. For $\alpha = \gamma^I$ we introduce one new clock x which is not used in $\mathcal{T}_\gamma$ and use the construction for $\mathcal{T}_\alpha$ as illustrated in Fig. 25. The invariance condition "$x \in I$" ensures that the location $stop(\alpha)$ can be entered only by runs where the execution time lies within the time interval I. Here, the edges from $stop(\gamma)$ to $stop(\alpha)$ are labelled with the empty node-set and data and clock constraints true.

Büchi TCA for TSDSL-formulas. We now return to the problem of generating a Büchi TCA for a given TSDSL-formula φ. As mentioned above, we may apply adaptations of standard automata-based techniques for extended LTL model checking [33]. We first transform the original TSDSL-formula φ into an equivalent formula of an extended TSDS-logic. This logic is in the style of extended temporal logic ETL$_f$ à la Vardi and Wolper. Formulas of extended TSDS-logic are built by boolean combinators ($\neg$ and $\wedge$) and automata-formulas. The latter can be viewed as a generalization of the until-operator and formulas $\langle\alpha\rangle\psi$. The automata-formulas have the form $\mathcal{T}(\psi_1, \ldots, \psi_n)$ where $\psi_1, \ldots, \psi_n$ are formulas of the extended TSDS-logic and $\mathcal{T}$ is a slight variant of a TCA: the edges in $\mathcal{T}$ are either TCA-edges (i.e., labelled with a node-set, a data constraint, a clock constraint and a set of clocks) or edges with a label in $\{\psi_1, \ldots, \psi_n\}$. The other components (starting location, invariance conditions) are as in normal TCA. Moreover, $\mathcal{T}$ has a distinguished accepting state. For the purpose of TSDSL-model checking, it suffices to deal with automata-formulas of the form $\mathcal{T}_\alpha(\psi)$ (as substitute for $\langle\alpha\rangle\psi$) and $\mathcal{A}(\psi_1, \psi_2)$ (as substitute for $\psi_1 \mathsf{U} \psi_2$):

- $\mathcal{T}_\alpha(\psi)$ arises from $\mathcal{T}_\alpha$ by adding a ψ-labelled edge from $stop(\alpha)$ to a new accepting state.
- $\mathcal{A}(\psi_1, \psi_2)$ consists of an initial state q_0 with a ψ_1-labeled self-loop and an accepting state q_1 which is reached from q_0 via an ψ_2-labeled edge.

The syntax of extended TSDS-logic agrees with the syntax of ETL$_f$, except that we deal with TCA-like automata rather than nondeterministic finite automata. Roughly the same construction of Büchi automata for given ETL$_f$-formulas that has been suggested by Vardi and Wolper [33] can be applied in our setting, to obtain a Büchi TCA $\mathcal{F}$ for the given TSDSL-formula, viewed as a formula of extended TSDS-logic. The automaton $\mathcal{F}$ is obtained by combining a so-called local automaton $\mathcal{F}_L$ with an eventually automaton $\mathcal{F}_E$. In the Vardi–Wolper-construction, both automata $\mathcal{F}_L$ and $\mathcal{F}_E$ arise through a certain combination of the edges in the automata for the automata-subformulas. The same technique is applicable in our setting and allows us to "lift" the time-guards and invariance conditions in the TCA of automata-subformulas to obtain corresponding time-guards and invariance conditions in the constructed Büchi TCA.

Complexity of TSDSL model checking. The major steps of the above sketched TSDSL model checking algorithm are (1) the construction of the Büchi TCA $\mathcal{F}$ for the negation of the given formula φ, and (2) checking emptiness for $\mathcal{T} \bowtie \mathcal{F}$. While the TCA $\mathcal{T}_\alpha$ for the sub-expressions of φ are linear in the length of α, the number of states in the resulting Büchi TCA $\mathcal{F}$ is exponential in the length of φ. The exponential blow-up arises in the construction of the local automaton $\mathcal{F}_L$ whose locations are sets of subformulas of φ (respectively, the corresponding formula of extended TSDS-logic). Thus, the number of locations in $\mathcal{T} \bowtie \mathcal{F}$ is $\mathcal{O}(\exp(|\varphi|) \cdot |\mathcal{T}|)$. The cost for the analysis of the region graph of $\mathcal{T} \bowtie \mathcal{F}$ for the emptiness check is dominated by the number of regions. These grow exponentially in the number of clocks (in $\mathcal{T}$ and $\mathcal{F}$) and linear in the number of locations in the product. Thus, the running time of the sketched model checking algorithm is linear in the number of locations of $\mathcal{T}$, and exponential in (a) the length of the formula φ, (b) the number of clocks in $\mathcal{T}$, and (c) the number of time-bounded subexpressions α^I of φ.

Remark 5.4 (TSDSL versus refinement relations) Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two TCA with the same node-set $\mathcal{N}$. Clearly, if $\mathcal{L}(\mathcal{T}_1) \subseteq \mathcal{L}(\mathcal{T}_2)$ then, for any TSDSL-formula φ, $\mathcal{T}_2 \models \varphi$ implies $\mathcal{T}_1 \models \varphi$. Thus, if $\mathcal{L}(\mathcal{T}_1) = \mathcal{L}(\mathcal{T}_2)$ then $\mathcal{T}_1$ and $\mathcal{T}_2$

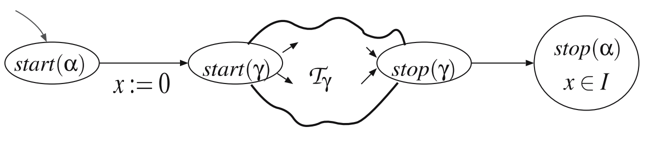


Fig. 25 TCA $\mathcal{T}_{\gamma^I}$

satisfy exactly the same TSDSL-formulas. A sufficient decidable criterion for checking (TSDLS- or) language-equivalence of two TCA is to switch to a coarser equivalence corresponding to timed bisimulation for ordinary timed automata [11]. In our setting, a timed bisimulation for a TCA $\mathcal{T}$ is the coarsest equivalence $\sim$ on the state space Q of the induced state-transition graph $\mathcal{A}_{\mathcal{T}}$ such that for all $q_1, q_2 \in Q$ with $q_1 \sim q_2$ and all $N \subseteq \mathcal{N}$, $\delta \in DA(N)$, $t \in \mathbb{R}_{\geq 0}$:

$$\forall q_1 \xrightarrow{N,\delta,t} p_1 \exists p_2 \in Q \text{ s.t. } q_1 \xrightarrow{N,\delta,t} p_2 \text{ and } p_1 \sim p_2.$$

The simulation relation is defined as the coarsest binary relation $\preceq$ on the state space Q of $\mathcal{A}_{\mathcal{T}}$ such that for all $q_1, q_2 \in Q$ with $q_1 \preceq q_2$ and all $N \subseteq \mathcal{N}$, $\delta \in DA(N)$, $t \in \mathbb{R}_{\geq 0}$:

$$\forall q_1 \xrightarrow{N,\delta,t} p_1 \exists p_2 \in Q \text{ s.t. } q_1 \xrightarrow{N,\delta,t} p_2 \text{ and } p_1 \preceq p_2.$$

The relation $\preceq$ is finer than language-inclusion, and thus, preserves all TSDSL formulas in the sense that if $q_1 \preceq q_2$ and $q_2 \models \varphi$ then $q_1 \models \varphi$. The question of whether one state of a TCA simulates another one can be answered with the help of the region graph construction as in [32]. □

6 Conclusion

In this paper, we introduced a formal model to reason about timing constraints for Reo component connectors. We presented composition operators for join and hide that can serve as a basis for the automated construction of an automaton-model from a given (timed) Reo circuit, and as a starting point for its formal verification. Particularly, (slightly modified versions of) well-known algorithms for checking time-lock freedom in ordinary timed automata can serve for checking the realizability of the coordination mechanisms of a Reo circuit with timing constraints. Moreover, we suggested a linear-time temporal logic for reasoning about the real-time behavior of component connectors based on their timed scheduled-data streams. Finally, we sketched how the standard model checking algorithms for timed automata can be adapted for our setting.

Our future work includes an implementation of the presented model checking algorithms and case studies. Moreover, we intend to study an alternating-time logic in the style of [4] that allows to reason about the possibility for certain components to cooperate such that a given (real-time) property holds.

References

1. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: Proceedings of the second international workshop on embedded software (EMSOFT), vol. 2491 of Lecture Notes in Computer Science, pp. 108–122 (2002)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor Comput Sci **126**(2) 183–235 (1994)
3. Alur, R., Henzinger, T.A.: A really temporal logic. J ACM, **41**:181–204 (1994)
4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J ACM, **49**:672–713 (2002)
5. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J ACM bf 43(1) 116–146 (1996)
6. Arbab, F.: Reo: a channel-based coordination model for component composition. Math Struct Comput Sci **14**(3):1–38 (2004)
7. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Pattinson, D., Wirsing, M., Hennicker, R. (eds), Recent trends in algebraic development techniques, proceedings of 16th international workshop on algebraic development techniques (WADT 2002), vol. 2755 of Lecture Notes in Computer Science, pp. 35–56. Springer Berlin Heidelberg New York, (2003). http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R0216.pdf
8. Arbab, F., Baier, C., de Boer, F.S., Rutten, J.J.M.M., Sirjani, M.: Modeling context-senstive behavior of component connectors with priorities. (Forthcoming paper) (2006)
9. Arbab, F., Baier, C., Rutten, J.J.M.M., Sirjani, M.: Modeling component connectors in Reo by constraint automata. In: Proc. international workshop on foundations of coordination languages and software architectures (FOCLASA 2003), vol. 97(22) of Electronic Notes in Theoretical Computer Science. Elsevier Science, July 2004. A full version will appear in Science of Computer Programming and is available under http://web.informatik.uni-bonn.de/I/baier/publikationen.html
10. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J ACM, **49**(2):172–206 (2002)
11. Cerans, K., Decidability of bisimulation equivalences for parallel timer processes. In: Proc. 4th international workshop on computer aided verification (CAV), vol. 663 of Lecture Notes in Computer Science, pp. 302–315. Springer Berlin Heidelberg New York. (1993)
12. Fischer, M.J., Ladner, R.J.: Propositional dynamic logic of regular programs. J Comput Syst Sci **8**:194–211 (1979)
13. Gawlick, R., Segala, R., Soegaard-Andersen, J., Lynch, N.: Liveness in timed and untimed systems. Inform Comput **141**(2):119–171 (1998)
14. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In protocol specification testing and verification, pp. 3–18. Chapman & Hall, London (1995)
15. Harel, E., Lichtenstein, O., Pnueli, A.: Explicit clock temporal logic. In: Proc. fifth annual IEEE symposium on logic in computer science (LICS), pp. 402–413. IEEE Computer Society Press Los Alamitos (1990)
16. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inform Comput **111**(2):193–244 (1994)
17. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: Hytech: a model checker for hybrid systems. Softw Tools Technol Transfer 1:110–122 (1997)
18. Holliday, M.A., Vernon, M.K.: A generalised timed petri net model for performance analysis. IEEE Trans Softw Eng 13(12):1279–1310 (1987)

19. Kaynar, D.K., Lynch, N.A., Segala, R., Vaandrager, F.W.: A framework for modelling timed systems with restricted hybrid automata. In: Proceedings 24th IEEE international real-time systems symposium (RTSS'03), pp. 166–177. IEEE Computer Society press, Los Alamitos (2003)
20. Guldstrand Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int J Softw Tools Technology Transfer **1**(1-2):134–152 (1997)
21. Leonard, L., Leduc, G.: An enhanced version of timed lotos and its application to a case study. In: Proc. formal description techniques VI, pp. 483–498. North-Holland, Amsterdam (1994)
22. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems. Springer, Berlin Heidelberg New York (1992)
23. Merlin, P.M.: A study of the recoverability of computing systems. PhD thesis, Department of Information and Computer Science, University of California, Irvine (1974)
24. Merritt, M. Modugno, F., Tuttle, M.R.: Time-constrained automata (extended abstract). In: Proc. 2nd international conference on concurrency theory, vol. 527 of Lecture Notes in Computer Science, pp. 408–423. Springer Berlin Heidelberg New York, (1991)
25. Milner, R.: Communication and concurrency. Prentice Hall International Series in Computer Science. Prentice Hall (1989)
26. Panangaden, P., van Breugel, F. (eds): Mathematical techniques for analyzing concurrent and probabilistic systems. CRM Monograph Series. American Mathematical Society (2004) ISSN 1065–8599
27. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77), pp. 46–57, Providence, Rhode Island, October 31–November 2. IEEE Computer Society Press, Los Alamitos (1977)
28. Ramchandani, C.: Analysis of asynchronous concurrent systems by timed petri nets. Project MAC 120, MIT (1974)
29. Reed, G.M., Roscoe, A.W.: A timed model for communication sequential processes. Theor Comput Sci **58**:249–261 (1988)
30. Rutten, J.J.M.M.: Component connectors. In [26], Chap. 5, pp 73–87 (2004)
31. Sifakis, J.: Performance evaluation of systems using nets. In: Brauer, W. (ed.) Proceedings of the advanced course on general net theory, Appeared as Lecture Notes in Computer Science 84 FRG, Springer, Berlin Heidelberg New York (1980)
32. Tasiran, S., Alur, R., Kurshan, R., Brayton, R.: Verifying abstractions of timed systems. In: Proc. 7th conference on concurrency theory (CONCUR), vol. 1119 of Lecture Notes in Computer Science, pp. 546–562 (1996)
33. Vardi, M., Wolper, P.: Reasoning about infinite computations. Inform Comput **115**:1–37 (1994)
34. Wolper, P.: Specification and synthesis of communicating processes using an extended temporal logic. In: Proc. 9th symposium on principles of programming languages (POPL), pp. 20–33 (1982)
35. Wolper, P., Vardi, M., Sistla, A.: Reasoning about infinite computation paths. In: Proc. 24th symposium on foundations of computer science (FOCS), pp. 185–194. IEEE Computer Society Press Los Alamitos (1983)
36. Yi, W.: CCS + time = an interleaving model for real time systems. In: Proceedings of the 18th international colloquium on Automata, languages and programming, vol. 510 of Lecture Notes in Computer Science, pp. 217–228. Springer, Berlin Heidelberg New York, (1991)
37. Yovine, S.: Kronos: a verification tool for real-time systems. Softw Tools Technol Transfer **1**(1–2): 123–133 (1997)

Author Biographies

Christel Baier received the diploma degree in mathematics in 1990 from the University of Mannheim in Germany. She received the PhD degree (1994) and the venia legendi (1999), both from the Department of Computer Science at the University of Mannheim. Since autumn of 1999, she has been a professor of computer science at the Rheinische Friedrich-Wilhelms Universitaet Bonn. Her research interests are the theory of concurrent and probabilistic systems, verification, semantics of programming languages and process calculi and mathematical logic.

Prof. Farhad Arbab received his PhD in computer science from University of California, Los Angeles, in 1982. Dr. Arbab is a senior researcher at the Dutch National Research Center for Mathematics and Computer Science (CWI) in Amsterdam, and professor of computer science at Leiden University, in the Netherlands. His fields of interest include software composition, coordination models and languages, service oriented computing, component based systems, and concurrency.

Frank de Boer received his PhD in computer science from the Free University, Amsterdam, 1991. In his thesis he developed a first sound and complete proof theory for a parallel object-oriented language. Dr. de Boer is a senior researcher at the Dutch National Research Center for Mathematics and Computer Science (CWI) in Amsterdam, and associate professor of computer science at Leiden University, in the Netherlands. His primary field of interest concerns the semantics and proof theory of concurrent systems.

Prof. Dr. J.J.M.M. Rutten is head of research theme 'Coordination languages' at CWI (Centre for Mathematics and Computer Science) and professor of computer science at the VUA (Free University Amsterdam). His research interests are semantics, applied logic, coalgebra and coinduction, with applications to programming languages and software engineering.