Models and Temporal Logics for Timed Component Connectors

Farhad Arbab[1], Christel Baier[2], Frank de Boer[1,3], Jan Rutten[1,4]
[1]*Centrum voor Wiskunde en Informatica, Department of Software Engineering, Amsterdam, The Netherlands*
[2]*Universität Bonn, Institut für Informatik I, Germany*
[3] *Universiteit Leiden, The Netherlands*
[4] *Vrije Universiteit Amsterdam, The Netherlands*

Abstract

The coordination language Reo supports compositional system construction through connectors with real-time properties that exogenously coordinate the interactions among the constituent components into a coherent collaboration. 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.

1. Introduction

The task of designing a complex concurrent system with several components requires a *coordination model* that formalizes their mutual interactions. Reo [3] offers a powerful 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 can involve *timing constraints*. 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).

As the operational model for Reo connector circuits, we use *timed constraint automata* (TCA) which extend their untimed version [4] with the concepts borrowed from classical timed automata with location invariants [1, 10]. 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 [4], 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.

The semantics of the TCA and timed Reo circuits relies on *timed data streams* as in [5, 4], 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) 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 [6]. 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) [8] or real-time variants of I/O-automata, e.g., [12, 9, 11], 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 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.

Although compositionality in timed Reo and TCA is in the spirit of real-time process algebras, e.g., [13, 16], Reo focusses on composition of connectors out of a variety of basic channel types.

Organization of the paper. Timed constraint automata are introduced in Section 2. In Section 3 we explain the main features of Reo circuits and how timed constraint automata can serve as their operational model. Timed scheduled-data-stream logic (TSDSL) is introduced in Section 4. Section 5 concludes the paper.

2. Timed constraint automata

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. Fig. 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 picture on the right shows the TCA corresponding to this Reo circuit. In the TCA on the right-

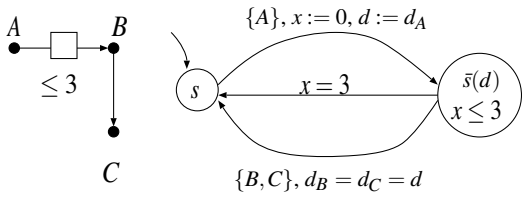


Figure 1. Reo circuit and timed constraint automaton

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 transfer 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 $\mathscr{C}$ be a finite set of clocks. A clock assignment means a function $\nu : \mathscr{C} \to \mathbb{R}_{\geq 0}$. If $t \in \mathbb{R}_{\geq 0}$ then $\nu + t$ denotes the clock assignment that assigns the value $\nu(x) + t$ to every clock $x \in \mathscr{C}$. If $C \subseteq \mathscr{C}$ 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 $x \in \mathscr{C} \setminus C$. A clock constraint (denoted cc) for $\mathscr{C}$ is a conjunction of atoms of the form "$x \bowtie n$" where $x \in \mathscr{C}$, $\bowtie \in \{<, \leq, >, \geq, =\}$ and $n \in \mathbb{N}$. $CA(\mathscr{C})$ (or CA) denotes the set of all clock assignments and $CC(\mathscr{C})$ (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 $\mathscr{T} = (S, \mathscr{C}, \mathcal{N}, \mathscr{E}, S_0, ic)$ where S is a finite set of control states (also called locations), $\mathscr{C}$ 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 $\mathscr{E}$ is a subset of $S \times 2^{\mathcal{N}} \times DC \times CC \times 2^{\mathscr{C}} \times S$ such that $dc \in DC(N)$ for any edge $e = (s, N, dc, cc, C, \bar{s}) \in \mathscr{E}$. 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 $dc \equiv \mathsf{true}$.)* □

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 $S = \{s\} \cup \{\bar{s}(d) : d \in \mathit{Data}\}$. The assignment "$d := d_A$" in the parametric version stands for

the data constraint $d_A = d$ in the TCA. An interface specification of a *timed sequencer* that coordinates the dataflow of two components via synchronous channels is shown in Fig. 2. We assume the deadline $t = 3$ for the write-operations, that is, the sequencer in location s waits up to t 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.

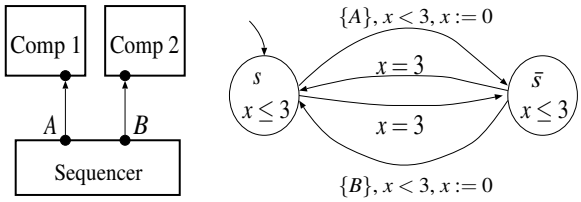


Figure 2. Timed sequencer

Definition 2.4 (State-transition graph of a TCA) *Given a TCA $\mathscr{T}$ as above, $\mathscr{T}$ induces a state-transition graph $\mathscr{A}_{\mathscr{T}} = (Q, \longrightarrow, Q_0)$ 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 $\longrightarrow \subseteq Q \times 2^{\mathscr{N}} \times DA \times \mathbb{R}_{\geq 0} \times Q$ is defined by the following rules:*

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

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

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

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

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

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

where $q_0 = q$. $\mathbf{q}$ is called initial if $q_0 \in Q_0$. $\mathbf{q}$ 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 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.

Notation 2.6 (TSD stream) A timed scheduled data stream for a node-set $\mathscr{N}$ denotes any (finite or infinite) sequence $\Theta = (N_0, \delta_0, t_0),\ (N_1, \delta_1 t_1), \ldots \in (2^{\mathscr{N}} \times DA \times \mathbb{R}_{\geq 0})^{\infty}$ such that $\delta_i \in DA(N_i)$, $0 < t_0 < t_1 < \ldots$ and $\lim_{i \to \infty} t_i = \omega$ 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, δ, t) in Θ. The execution time $\tau(\Theta)$ is ω if Θ is infinite, t_k if $|\Theta| = k + 1$, and 0 if $\Theta = \varepsilon$. We write $TSDS(\mathscr{N})$ or simply $TSDS$ to denote the set of all TSDS for node-set $\mathscr{N}$. □

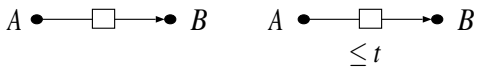
Notation 2.7 (TSDS-language of a TCA) If $\mathbf{q}$ is a run in a TCA $\mathscr{T}$ as above then the induced TSD stream $\Theta(\mathbf{q}) = (N_{i_0}, \delta_{i_0}, \bar{t}_{i_0}), (N_{i_1}, \delta_{i_1}, \bar{t}_{i_1}), \ldots$ is obtained from $\mathbf{q}$ by (1) removing all transitions in $\mathbf{q}$ 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 $\bar{t}_i = t_0 + \ldots + t_i$. The generated language of a state q in $\mathscr{A}_{\mathscr{T}}$ is $\mathscr{L}(\mathscr{T}, q) = \{\Theta(\mathbf{q}) : \mathbf{q} \text{ is a maximal } q\text{-run}\}$. The language $\mathscr{L}(\mathscr{T})$ 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. Timed Reo circuits

Reo's notion of *channel* is far more general than its common interpretation and encompasses any primitive communication medium with exactly two ends. Channel ends are classified into *source* ends through which data enter and *sink* ends through which data leave their respective channels. A write operation can be performed on the source end of a channel, providing data to enter into the channel, while a take operation can be performed on the sink end of a channel to obtain data out of the channel. We explain the workings of Reo with a few examples of its basic channel types and formalize their behavior by TCA.

FIFO channels. The simplest form of an asynchronous channel is a FIFO channel with one buffer cell, which we denote as *FIFO1*. A *FIFO1* channel is graphically represented 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 in its graphical representation (as in the example below). The graphical representation of a *FIFO1* channel whose buffer initially contains a data element d is the same except that it also shows a d inside the box representing its buffer.



On the left in this figure, we have a normal *FIFO1* channel which keeps a data item in its buffer until it is taken out through its sink. On the right we show a *lossy* variant, called *expiring FIFO1*, where a data item is lost if it is not taken out of the buffer through the sink end of the channel within t time units after it enters through its source end.

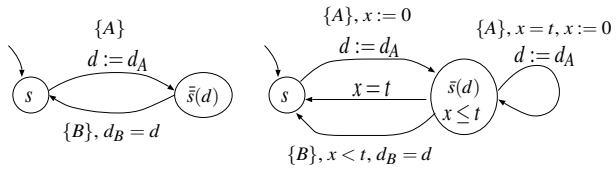
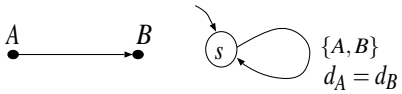
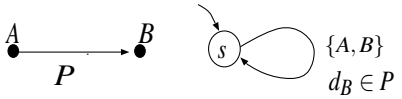


Figure 3. TCA for a normal and an expiring FIFO1 channels

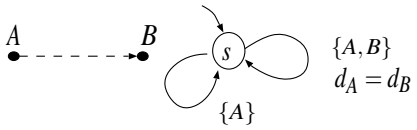
Synchronous channels. A synchronous channel, depicted as a solid arrow, has one source- and one sink-end. Write and take operations must occur simultaneously on the two ends of this channel, which is formalized by a TCA with a single location:



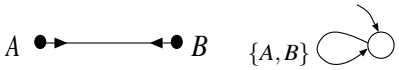
A *P-producer* is a synchronous channel that, like a normal synchronous channel, allows write and take operations to succeed atomically on its source and sink ends, respectively, except that the value dispensed through this channel's sink end is always a data element $d \in P$, regardless of the value it consumes through its source end.



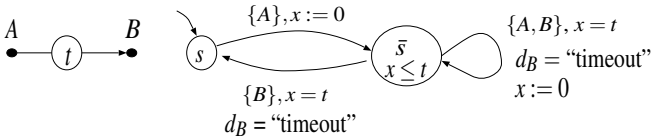
A *lossy synchronous channel* (depicted as a dashed arrow) is similar to a normal synchronous channel, except that it always accepts all data items through its source end. If it is possible for it to simultaneously dispense the data item through its sink (e.g., there is a take operation pending on its sink) the channel transfers the data item; otherwise the data item is lost.



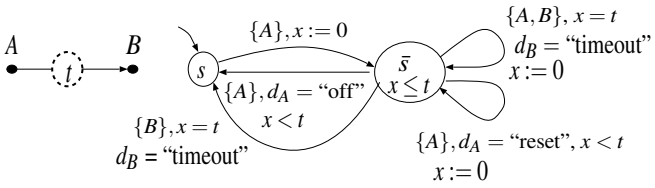
The above figure shows a TCA that captures the general "possible" behavior of a lossy synchronous channel. To model the context-sensitive behavior of a lossy channel where the $\{A\}$-transition is impossible if B is ready to synchronize, the concept of priorities can be used. More exotic channels permitted in Reo include the *synchronous drain* that has two source ends. Because a drain has no sink end, no data value can ever be obtained from this channel. Thus, all data accepted by this channel are lost. 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.



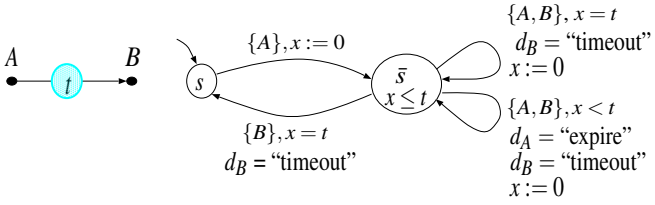
Timer. The source end of a *t-timer* channel accepts any input value $d \in Data$ and returns on its sink end a timeout signal after a delay of t time units.



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. The following figure shows a t-timer with both the reset- and the off-options.



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



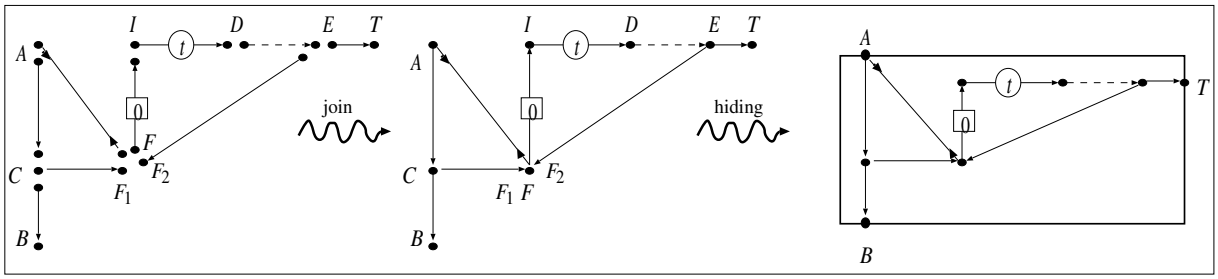


Figure 4. Example construction of a Reo circuit

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).

Reo circuits. Complex connectors have graphical representations, called *Reo circuits*, which can be generated by applying certain composition operators to channels. We may think of 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 to create Reo connector circuits are join and hiding.

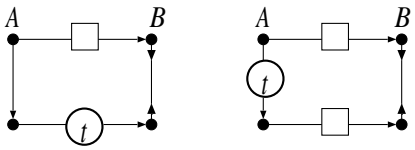
To construct a Reo circuit, we start with several instances of basic channels and organize them in a graph where initially each channel end constitutes a separate node, and each pair of nodes are connected by an edge representing their respective channel. We then apply a series of join operations that take as input two nodes A and B and combine them into a new node C. In this way, several channel ends may coincide on one node. If all channel ends coincident on a node C are source ends, C is called a *source node* and it acts as a *replicator*: writing a data item to a source node succeeds when all of its coincident channel ends are capable of accepting the data item simultaneously, in which case the data item is atomically copied into every one of the source ends coincident on C. If all channel ends coincident on C are sink ends, C is called a *sink node* and it behaves as a *merger*: an attempt to take a data item from a sink node succeeds when at least one of its coincident channel ends has a suitable value to offer, in which case the suitable value available through one of these channel ends is non-deterministically selected for the take operation. If C contains both source and sink channel ends then C is called a *mixed node* and it behaves as a self-contained pumping station, combining the replicator and merger behavior of source and sink nodes. No take or write operation can be performed on a mixed node; a mixed node autonomously selects suitable values available through its coincident sink ends (merger behavior) and copies them to its coincident source ends (replicator behavior).

The hiding operator allows to create "components" by putting a thick box around a circuit, insulating all of its mixed nodes inside the box and allowing access to its sink and source nodes, placed on the border of the box, only. The idea is that the 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.

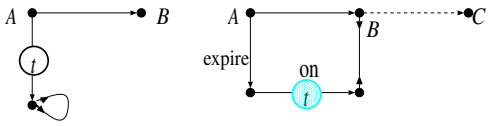
Fig. 4 demonstrates how to built a Reo circuit via join and hiding. Mixed node I serves as an initializer which activates the timer. Either A and B synchronize before the timer expires or the timeout signal occurs at T (after exactly t time units). In either case, the buffer is refilled and the whole procedure restarts.

When modeling Reo circuits by (timed) constraint automata the locations stand for the configurations of the circuits (e.g., contents of the FIFO channels) while the 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 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.

Example 3.1 The following figure shows on its left how an expiring FIFO1 channel can be constructed out of a normal FIFO1 channel and a timer set to expire after t time units. On the right 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 P-producer connected to A, where P is the singleton data set $\{\text{expire}\}$) 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 3.2 (Timed sequencer) The timed sequencer in Fig. 2 can be realized by the Reo circuit shown in Fig. 5 (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 $\{\text{expire}\}$) 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.

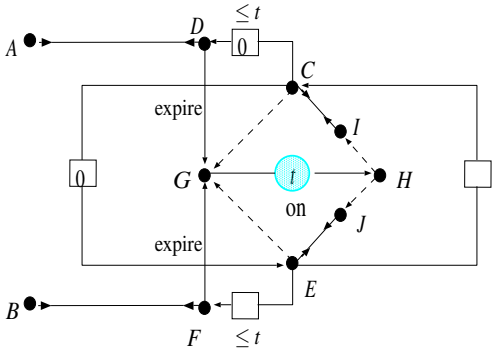
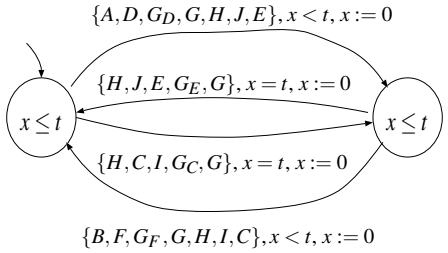


Figure 5. Reo circuit for a timed sequencer

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.



The above figure shows the TCA (before hiding) where we skip the data constraints.[1] □

Remark 3.3 (Time-constraints for the I/O-operations) In the Reo circuit in Fig. 6, node B is a mixed node which is "always" ready to consume a message from the buffer of the expiring FIFO1 channel because the 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

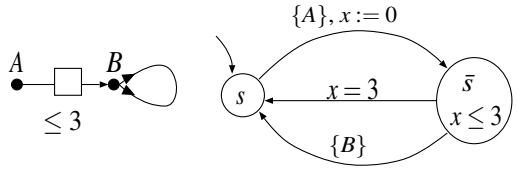
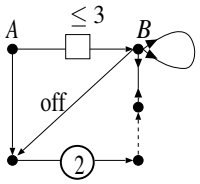


Figure 6. When does B perform a take-operation?

[1] 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.

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:



□

We now define 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 $\mathscr{T}_1$ and $\mathscr{T}_2$ 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 $\mathscr{T}_1$) and $\tilde{B}_i$ (in $\mathscr{T}_2$), $i = 1, \ldots, n$, 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 $\mathscr{T}_1$ and $\mathscr{T}_2$.

Definition 3.4 (Join for TCA) *Given two TCA $\mathscr{T}_i = (S_i, \mathscr{C}_i, \mathscr{N}_i, \mathscr{E}_i, S_{0,i}, ic_i)$, $i = 1, 2$, with disjoint clock sets, the product $\mathscr{T}_1 \bowtie \mathscr{T}_2$ is defined as an 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 $\mathscr{N} = \mathscr{N}_1 \cup \mathscr{N}_2$, and the clock set $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$. The location invariance is given by $ic(\langle s_1, s_2 \rangle) = ic_1(s_1) \wedge ic(s_2)$. The edge relation $\mathscr{E}$ 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:*

$$\frac{\begin{array}{c}(s_1, N_1, dc_1, cc_1, C_1, \bar{s}_1) \in \mathscr{E}_1, \\ (s_2, N_2, dc_2, cc_2, C_2, \bar{s}_2) \in \mathscr{E}_2, \\ N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1,\ N_1 \neq \emptyset,\ N_2 \neq \emptyset,\ dc_1 \wedge dc_2 \not\equiv \mathsf{false}\end{array}}{(\langle s_1, s_2 \rangle, N_1 \cup N_2, dc_1 \wedge dc_2, cc_1 \wedge cc_2, C_1 \cup C_2, \langle \bar{s}_1, \bar{s}_2 \rangle) \in \mathscr{E}}$$

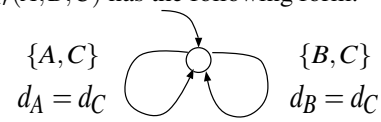
The second rule applies to edges all of whose involved nodes are local to only one of the automata:

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

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 the full version.

To mimic the merge semantics of sink (or mixed) nodes we use the same technique as in [5, 4]. 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 $\mathscr{T}_{Merger}(A, B, C) \bowtie \mathscr{T}_A \bowtie \mathscr{T}_B$ where $\mathscr{T}_A$ and $\mathscr{T}_B$ are the TCA that model the sub-circuits containing A and B respectively, and $\mathscr{T}_{Merger}(A, B, C)$ has the following form:



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.

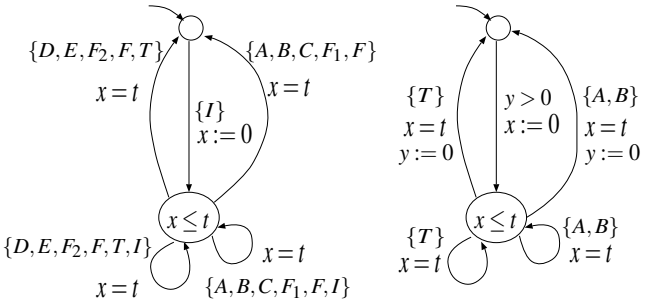
Definition 3.5 (Hiding for TCA) *Given a TCA $\mathscr{T} = (S, \mathscr{C}, \mathscr{N}, \mathscr{E}, S_0, ic)$, a new clock $y \notin \mathscr{C}$, and $M \subseteq \mathscr{N}$, we define $\exists M[\mathscr{T}] = (S, \mathscr{C} \cup \{y\}, \mathscr{N} \setminus M, \mathscr{E}', S_0, ic)$ where $\mathscr{E}'$ is obtained by the rule:*

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

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

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 3.6 The TCA for the circuit in Fig. 4 can be obtained by joining the TCA for all of its involved channels together with $\mathscr{T}_{Merger}(F_1, F_2, F)$.



The above figure shows the resulting TCA before and after hiding. (For simplicity, we skip the data constraints and irrelevant resettings of y). $\square$

Of course, using arbitrary combinations of timed channels can lead to TCA with time-locks. However, using (modifications of) standard region- or zone-graph algorithms [1, 10] we may check the time-lock freedom of a given Reo circuit.

4. Timed Scheduled-Data-Stream Logic

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 $\langle\!\langle \alpha \rangle\!\rangle \varphi$ which consist of a so-called timed scheduled-data expression α and a formula φ. This type of formulas is inspired by propositional dynamic logic and extended temporal logic [15]. The timed scheduled-data expressions are variants of timed regular expressions [6] built from atoms of the form $\langle N, dc \rangle$. The TSD expressions specify *sets of finite TSD streams*. The intuitive meaning of $\langle\!\langle \alpha \rangle\!\rangle \varphi$ 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.

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:

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

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)[2], $\alpha_1;\alpha_2$ (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, $\langle\!\langle \alpha \rangle\!\rangle \varphi$ 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 $\langle\!\langle \alpha \rangle\!\rangle \varphi$ is $[\![\alpha]\!]\varphi \;=\; \neg\langle\!\langle \alpha \rangle\!\rangle \neg\varphi$ 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.

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. $\langle \neg A \rangle$ stands for the disjunction of all expressions $\langle N \rangle$ where N ranges over all non-empty node-sets that do not contain A. $\langle \cdot \rangle$ denotes the disjunction of all atoms $\langle N \rangle$ where N is an arbitrary non-empty node-set. $\langle\!\langle \cdot \rangle\!\rangle \varphi$ 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. 3) 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 (cf. Fig. 3):

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

The expiring FIFO1 channel in Fig. 3 satisfies the TSDSL formula

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

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 3.2) the following formula holds

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

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

[2] Standard regular expressions do not contain an intersection operator (although regular languages are closed under intersection). However, as pointed out in [6], in timed settings, the class of timed languages induced by timed regular expressions without an explicit intersection operator is not closed under intersection.

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

Figure 7. Satisfaction relation for TSDSL-formulas

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

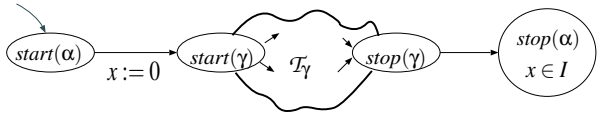
To provide the formal definition of the semantics of a TSD expressions and TSDSL-formulas we need some additional notation for working with TSD streams.

Notation 4.1 (Time cuts, concatenation, Kleene closure) Let $\Theta = (N_0, \delta_0, t_0), (N_1, \delta_1, t_1), \ldots$ be a TSD stream as in Notation 2.6. For a point in time $t \in \mathbb{R}_{\geq 0}$, 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 $[t, \infty[$. That is, $\Theta \uparrow t = \varepsilon$ if $|\Theta| = k+1 < \omega$ and $t_k < t$. Otherwise, $\Theta \uparrow t = (N_k, \delta_k, t_k), \ldots)$ where k is the smallest index such that $t_k \geq t$.
$\Theta \downarrow t$ is the TSD stream that describes the data-flow in the time interval $[0, t[$. That is, $\Theta \downarrow t = \varepsilon$ if $\Theta = \varepsilon$ or $t_0 \geq t$. Otherwise, $\Theta \downarrow t = (N_0, \delta_0, t_0), \ldots, (N_k, \delta_k, t_k)$ where k is the largest index such that $t_k < t$.
The concatenation of finite TSD streams is defined as follows. We define $\Theta;\varepsilon = \varepsilon;\Theta = \Theta$. If $\Theta_1 = (N_0, \delta_0, t_0), \ldots, (N_n, \delta_n, t_n)$ and $\Theta_2 = (M_0, \sigma_0, \rho_0), \ldots, (M_m, \sigma_m, \rho_m)$ then $\Theta_1;\Theta_2$ is $(N_0, \delta_0, t_0), \ldots, (N_n, \delta_n, t_n), (M_0, \sigma_0, t_n + \rho_0), \ldots, (M_m, \sigma_m, t_n + \rho_m)$. If L and $\tilde{L}$ are TSDS-languages with the same node-set $\mathscr{N}$ then $L;\tilde{L} = \{\Theta\,;\tilde{\Theta} : \Theta \in L, \tilde{\Theta} \in \tilde{L}\}$ and $L^* = \bigcup_{n \geq 0} L^n$ where $L^0 = \{\varepsilon\}$, $L^{n+1} = L^n;L$. □

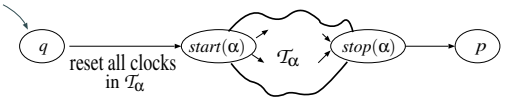
Semantics of TSD expressions and TSDSL-formulas. We define $\mathscr{L}(\alpha) \subseteq \mathit{TSDS}$ by structural induction. $\mathscr{L}(\langle N, dc\rangle)$ is the set of all TSD streams of length 1 that have the form (N, δ, t) where $\delta \models dc$. We define $\mathscr{L}(\alpha_1 \vee \alpha_2) = \mathscr{L}(\alpha_1) \cup \mathscr{L}(\alpha_2)$, $\mathscr{L}(\alpha_1 \wedge \alpha_2) = \mathscr{L}(\alpha_1) \cap \mathscr{L}(\alpha_2)$, $\mathscr{L}(\alpha_1;\alpha_2) = \mathscr{L}(\alpha_1);\mathscr{L}(\alpha_2)$ and $\mathscr{L}(\alpha^*) = \mathscr{L}(\alpha)^*$. The semantics of time-constrained expressions is formalized by $\mathscr{L}(\alpha^I) = \{\Theta \in \mathscr{L}(\alpha) : \tau(\Theta) \in I\}$.[3] The satisfaction relation $\models$ for TDSL-formulas and TSD streams is defined by structural induction as shown in Fig. 7. For the derived $[\![\ldots]\!]$-operator, we obtain $\Theta \models [\![\alpha]\!]\varphi$ iff for all $t \geq 0$ we have: $\Theta \downarrow t \in \mathscr{L}(\alpha)$ implies $\Theta \uparrow t \models \varphi$. We define $\mathscr{L}(\varphi) = \{\Theta \in \mathit{TSDS}(\mathscr{N}) : \Theta \models \varphi\}$ and define logical equivalence $\equiv$ of TSDSL-formulas as $\varphi_1 \equiv \varphi_2$ iff $\mathscr{L}(\varphi_1) = \mathscr{L}(\varphi_2)$. If $\mathscr{T}$ is a TCA and q a state in $\mathscr{A}_{\mathscr{T}}$ then $q \models \varphi$ iff $\mathscr{L}(\mathscr{T}, q) \subseteq \mathscr{L}(\varphi)$. Moreover, we define $\mathscr{T} \models \varphi$ iff $\mathscr{L}(\mathscr{T}) \subseteq \mathscr{L}(\varphi)$.

[3] Recall that $\tau(\Theta)$ denotes the execution time of Θ (see Notation 2.6).

The TSDSL Model Checking problem addresses the question of whether $\mathscr{T} \models \varphi$ holds for a given TCA $\mathscr{T}$ and TSDSL formula φ. We briefly sketch the main ideas of a TSDSL model checking algorithm that relies on variants of standard automata-based algorithms for LTL and (timed) regular expressions.
First, we switch from φ to $\neg\varphi$ which we regard as a formula of (untimed) LTL with action labels. Here, $\langle\!\langle \alpha \rangle\!\rangle$ is treated as a next step operator with the label α. Then, we may apply standard techniques modified for the action-labeled case, to construct a nondeterministic Büchi automaton $\mathscr{B}$ for $\neg\varphi$, whose transitions are labeled with the expressions α that occur in sub-formulas $\langle\!\langle \alpha \rangle\!\rangle \psi$ of φ. We now turn $\mathscr{B}$ into a TCA $\mathscr{T}_{\mathscr{B}}$ with Büchi acceptance condition.
For this, we first construct a TCA $\mathscr{T}_\alpha$ for every TSD expression α that occurs in $\mathscr{B}$ as a transition-label. $\mathscr{T}_\alpha$ has a unique initial location, called $\mathit{start}(\alpha)$, and a location $\mathit{stop}(\alpha)$ such that $\mathscr{L}(\alpha)$ is the set of all TSD streams Θ that are induced by a finite run in $\mathscr{T}_\alpha$ starting in $\mathit{start}(\alpha)$ and ending in $\mathit{stop}(\alpha)$. The construction of the TCA $\mathscr{T}_\alpha$ is by structural induction, essentially as described in [6]. For instance, for $\alpha = \gamma^I$ we introduce one new clock x that is not used in $\mathscr{T}_\gamma$ and perform the following construction for $\mathscr{T}_\alpha$:



The invariance condition "$x \in I$" ensures that location $\mathit{stop}(\alpha)$ can be entered only in runs where the execution time lies within the time interval I. (Here, the edges from $\mathit{stop}(\gamma)$ to $\mathit{stop}(\alpha)$ are labeled with the empty node-set and data and clock constraint true.)
The TCA $\mathscr{T}_{\mathscr{B}}$ is now obtained as follows. The locations in $\mathscr{T}_{\mathscr{B}}$ consist of the states in the Büchi automaton $\mathscr{B}$ and the locations in the TCA $\mathscr{T}_\alpha$.[4] We then replace every transition $q \xrightarrow{\alpha} p$ in $\mathscr{B}$ with the following fragment of $\mathscr{T}_{\mathscr{B}}$:



We then have $\mathscr{L}(\mathscr{T}_{\mathscr{B}}) = \mathscr{L}(\neg\varphi)$ where Büchi acceptance is assumed for $\mathscr{T}_{\mathscr{B}}$. Thus, by Corollary **??**, $\mathscr{T} \models \varphi$ iff $\mathscr{L}(\mathscr{T} \bowtie \mathscr{T}_{\mathscr{B}}) = \mathscr{L}(\mathscr{T}) \cap \mathscr{L}(\varphi) = \emptyset$. Hence, we may apply (modifications of) the standard region graph algorithms to check for emptiness of timed automata [1].

[4] We assume that the state spaces and clock sets are disjoint and that for any TSD expression α that occurs more than once in $\mathscr{B}$ a copy of $\mathscr{T}_\alpha$ is used.

TSDSL versus refinement relations. Let $\mathscr{T}_1$ and $\mathscr{T}_2$ be two TCA with the same node-set $\mathscr{N}$. Clearly, if $\mathscr{L}(\mathscr{T}_1) \subseteq \mathscr{L}(\mathscr{T}_2)$ then, for any TSDSL-formula φ, $\mathscr{T}_2 \models \varphi$ implies $\mathscr{T}_1 \models \varphi$. Thus, if $\mathscr{L}(\mathscr{T}_1) = \mathscr{L}(\mathscr{T}_2)$ then $\mathscr{T}_1$ and $\mathscr{T}_2$ 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 [7]. In our setting, a timed bisimulation for a TCA $\mathscr{T}$ is the coarsest equivalence $\sim$ on the state space Q of the induced state-transition graph $\mathscr{A}_{\mathscr{T}}$ such that for all q_1, $q_2 \in Q$ with $q_1 \sim q_2$ and all $N \subseteq \mathscr{N}$, $\delta \in DA(N)$, $t \in \mathbb{R}_{\geq 0}$:

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

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

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

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 [14].

5. 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 hiding that can serve as a basis for the automated construction of an automata-model from a given (timed) Reo circuit and as a starting point for its formal verification. In particular, (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 by means of their timed scheduled-data streams and explained how the standard region- or zone-graphs 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 [2] that allows to reason about the possibility for certain components to cooperate such that a given (real-time) property holds.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

[3] F. Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):1–38, 2004.

[4] F. Arbab, C. Baier, J.J.M.M. Rutten, and M. Sirjani. Modeling component connectors in reo by constraint automata. In *FOCLASA'03*, Electronic Notes in Theoretical Computer Science, 2003. To appear.

[5] F. Arbab and J.J.M.M. Rutten. A coinductive calculus of component connectors. In D. Pattinson M. Wirsing and R. Hennicker, editors, *Proceedings of 16th International Workshop on Algebraic Development Techniques (WADT 2002)*, volume 2755 of *Lecture Notes in Computer Science*, pages 35–56. Springer-Verlag, 2003.

[6] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *Journal of the ACM*, 49(2):172–206, 2002.

[7] K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. CAV*, volume 663 of *LNCS*, pages 302–315, 1993.

[8] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proc. EMSOFT*, volume 2491 of *LNCS*, pages 108–122, 2002.

[9] R. Gawlick, R. Segala, J. Soegaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141(2):119–171, 1998.

[10] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[11] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. A framework for modelling timed systems with restricted hybrid automata. In *Proceedings 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 166–177. IEEE Computer Society, 2003.

[12] M. Merritt, F. Modugno, and M. R. Tuttle. Time-constrained automata (extended abstract). In *Proc. CONCUR*, volume 527 of *LNCS*, pages 408–423, 1991.

[13] G. M. Reed and A. W. Roscoe. A timed model for communication sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[14] S. Tasiran, R. Alur, R. Kurshan, and R. Brayton. Verifying abstractions of timed systems. In *Proc. CONCUR*, volume 1119 of *LNCS*, pages 546–562, 1996.

[15] P. Wolper. Specification and synthesis of communicating processes using an extended temporal logic. In *Proc.POPL*, pages 20–33, 1982.

[16] W. Yi. CCS + time = an interleaving model for real time systems. In *Proc. ICALP*, volume 510 of *LNCS*, pages 217–228. Springer-Verlag, 1991.