

Time Abstraction in Timed μ CRL à la Regions

Jan Friso Groote Michel A. Reniers Yaroslav S. Usenko

Laboratory for Quality Software, Department of Mathematics and Computer Science,
Technical University of Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

1 Introduction

We present the first step towards combining the best parts of the real-time verification methods based on timed automata (the use of regions and zones), and of the process-algebraic approach of languages like LOTOS and μ CRL. This could provide with additional verification possibilities for Real-Time systems, not available in existing timed-automata-based tools like UPPAAL [14].

The language μ CRL, see [12], offers a uniform framework for the specification of data and processes. Data are specified by equational specifications (cf. [5]): one can declare sorts and functions working upon these sorts, and describe the meaning of these functions by equational axioms. Processes are described in process algebraic style, where the particular process syntax stems from ACP [6, 3], extended with data-parametric ingredients: there are constructs for conditional composition, and for data-parametric choice and communication. As is common in process algebra, infinite processes are specified by means of (finite systems of) recursive equations. In μ CRL such equations can also be data-parametric.

Several timed extensions have been proposed for different kinds of process algebras. For an overview of ACP extensions with time we refer to [2]. According to [2], timed process algebras can be categorized by three criteria: *discrete vs. continuous time*; *relative vs. absolute time*; *two-phase vs. timed-stamped model*. In [9], μ CRL is extended with time, and in [15] a sound and complete axiomatization of timed μ CRL is presented. Timed μ CRL makes use of absolute time, timed stamped model, and the time domain can be defined by the user (both discrete and continuous domains are possible).

In [16], we outlined a method to describe and analyze real-time systems using timed μ CRL. Most descriptions of such systems contain operators such as parallel composition that complicate analysis. As a first step towards the analysis of such systems, we *linearize* the given description using the algorithm from [18]. The result is a *Timed Linear Process Equation* (TLPE) which is equivalent to the original description and has a very simple structure. We also presented a transformation of a TLPE into an LPE, i.e., a linear process equation without time. This transformation, called *time-free abstraction*, has been used for non-recursive timed μ CRL processes in [15]. Crucial for this transformation is that the TLPE is transformed into a *well-timed* TLPE.

Finally, all time-stamping is captured in the parameters of atomic actions. The result is an LPE for which the machinery of untimed μ CRL can be put to use for further analysis [12, 10]. These are based on symbolic analysis of the specifications, such as invariants, term rewriting and theorem proving, or on explicit state space generation and model-checking.

In the present paper, we use the existing results of [19] to translate a timed automaton to a timed μ CRL processes in the form of a TLPE. This translation uses a very simple sort *Time* to represent the real-time clock values. As a result we obtain a semantically equivalent specification in timed μ CRL. The resulting timed μ CRL process only uses a very restricted subset of the full syntax of the language: it has the form of a TLPE.

We aim to transfer the successful techniques of *regions* and *zones* [1] as used for the analysis of timed automata to the realm of timed μ CRL. First, we aim at replacing all parameters of sort *Time* occurring in the resulting process equation by parameters of discrete sorts. To achieve this goal we apply process-algebraic transformations and abstraction techniques to the given process equation. As a result we obtain a process equation that is closely related to the given one in the following sense. If we abstract from the fractional parts of the time stamps in the actions, both of the equations will be timed bisimilar.

2 Timed μ CRL

In this section, we present those parts of timed μ CRL that are needed for a good understanding of the material that follows in subsequent sections. For a more complete treatment we refer to [11, 12] for untimed μ CRL and to [9, 15] for timed μ CRL.

Timed μ CRL specifications contain algebraic specifications of several abstract data types. The only data types that are required are booleans and time. The algebraic specifications of booleans are standard (see [8, Chapter IV]).

Time can be represented in many different ways. In timed μ CRL the time domain has to satisfy a set of properties. We present these properties as an algebraic specification of sort *Time* by defining its signature and the axioms. The signature of sort *Time* consists of: a constant $\mathbf{0}$; functions $\leq, = : Time \times Time \rightarrow Bool$; function *if* : $Bool \times Time \times Time \rightarrow Time$; and functions *min, max* : $Time \times Time \rightarrow Time$.

The axioms of sort *Time* can be found in [18, Page 84]. They say that \leq is a total order on the *Time* domain, and $\mathbf{0}$

is the least element. Any other data type in μCRL is specified in a similar way by providing a signature and axioms from which all other identities are derived. Other data sorts have generally different axioms, and sometimes induction principles (cf. [13]) are required to describe them.

The signature of timed μCRL consists of data sorts (or ‘data types’) including *Bool* and *Time* as defined above, and a distinct sort *Proc* of processes. The process operations used in this paper are the ones listed below:

- actions $\mathbf{a} : \overrightarrow{D}_a \rightarrow \text{Proc}$ where $\mathbf{a} \in \text{ActLab}$ is an action label and \overrightarrow{D}_a is a list of parameter types of \mathbf{a} .
- deadlock $\delta : \rightarrow \text{Proc}$. The constant δ models inaction, or the inability to perform actions.
- alternative composition $+$: $\text{Proc} \times \text{Proc} \rightarrow \text{Proc}$. The process $p + q$ behaves like p or like q , depending on which of the two performs the first action.
- sequential composition \cdot : $\text{Proc} \times \text{Proc} \rightarrow \text{Proc}$. The process $p \cdot q$ first performs the actions of p , until p terminates, and then continues as q .
- conditional operator $_ \triangleleft _ \triangleright _$: $\text{Proc} \times \text{Bool} \times \text{Proc} \rightarrow \text{Proc}$. The process term $p \triangleleft b \triangleright q$ behaves like p if b is equal to **t**, and if b is equal to **f** it behaves like q .
- alternative quantification $\sum_{d:D} : \text{Proc} \rightarrow \text{Proc}$, for each data variable d of sort D . The process $\sum_d p$ behaves like $p[d_1/d] + p[d_2/d] + \dots$, i.e., as the possibly infinite alternative composition of processes $p[d_i/d]$ for any data term d_i of sort D .
- at-operator \triangleleft : $\text{Proc} \times \text{Time} \rightarrow \text{Proc}$. The process $p \triangleleft t$ behaves as p , with the restriction that the first action of p must start at time t . The process $p \triangleleft t$ can delay till at most time t . If p consists of several alternatives, then only those with the first actions starting at time t will remain in $p \triangleleft t$. The alternatives that start earlier than t will express that $p \triangleleft t$ can delay till that earlier time. The alternatives that start later than t will express that $p \triangleleft t$ can wait till time t (but not till that later time).

Most notably we omitted parallel composition from the above list.

A key feature of timed μCRL is that it can be expressed that a process can delay till a certain time. The process $p + \delta \triangleleft t$ can certainly delay till time t , but can possibly delay longer, depending on p . Consequently, the process $\delta \triangleleft \mathbf{0}$ can neither delay nor perform actions, and the process δ can delay for an arbitrary long time, but cannot perform any action. We follow the intuition that a process that can delay till time t can also delay till an earlier moment, and a process that can perform a *first* action at time t can also delay till time t .

To prove identities in timed μCRL we use a combined many-sorted calculus, which for the sort of processes has the rules of binding-equational calculus [17], for the sorts of booleans and time has the rules of equational calculus, while other data sorts may include induction principles which could be used to derive process identities as well. We note that the derivation rules of binding-equational calculus do not allow to substitute terms containing free variables if they become bound.

We consider *systems* of *process equations* with the right

hand sides being timed μCRL process terms extended with parameterized recursive calls of the form $Y(\vec{t})$ for process name Y with parameters \vec{t} . A timed μCRL process equation is in TLPE if it is of the form displayed below (where I and J are disjoint).

$$\begin{aligned} X(\overrightarrow{d}:\overrightarrow{D}) &= \sum_{i \in I} \sum_{\overrightarrow{e}_i: \overrightarrow{E}_i} \mathbf{a}_i(\overrightarrow{f}_i(\overrightarrow{d}, \overrightarrow{e}_i)) \triangleleft t_i(\overrightarrow{d}, \overrightarrow{e}_i) \cdot X(\overrightarrow{g}_i(\overrightarrow{d}, \overrightarrow{e}_i)) \\ &\quad \triangleleft c_i(\overrightarrow{d}, \overrightarrow{e}_i) \triangleright \delta \triangleleft \mathbf{0} \\ &+ \sum_{j \in J} \sum_{\overrightarrow{e}_j: \overrightarrow{E}_j} \mathbf{a}_j(\overrightarrow{f}_j(\overrightarrow{d}, \overrightarrow{e}_j)) \triangleleft t_j(\overrightarrow{d}, \overrightarrow{e}_j) \triangleleft c_j(\overrightarrow{d}, \overrightarrow{e}_j) \triangleright \delta \triangleleft \mathbf{0} \\ &+ \sum_{\overrightarrow{e}_\delta: \overrightarrow{E}_\delta} \delta \triangleleft t_\delta(\overrightarrow{d}, \overrightarrow{e}_\delta) \triangleleft c_\delta(\overrightarrow{d}, \overrightarrow{e}_\delta) \triangleright \delta \triangleleft \mathbf{0} \end{aligned}$$

The equation is explained as follows. The process X , being in a state vector \overrightarrow{d} , can for any \overrightarrow{e}_i , that satisfy the condition $c_i(\overrightarrow{d}, \overrightarrow{e}_i)$, perform an action \mathbf{a}_i parameterized by $\overrightarrow{f}_i(\overrightarrow{d}, \overrightarrow{e}_i)$ at the absolute time $t_i(\overrightarrow{d}, \overrightarrow{e}_i)$, and then proceed to the state $\overrightarrow{g}_i(\overrightarrow{d}, \overrightarrow{e}_i)$. Moreover, it can for any \overrightarrow{e}_j , that satisfy the condition $c_j(\overrightarrow{d}, \overrightarrow{e}_j)$, perform an action \mathbf{a}_j parameterized by $\overrightarrow{f}_j(\overrightarrow{d}, \overrightarrow{e}_j)$, and then terminate successfully. The last summand indicates that for any $\overrightarrow{e}_\delta: \overrightarrow{E}_\delta$, that satisfies $c_\delta(\overrightarrow{d}, \overrightarrow{e}_\delta)$, the process can wait till the absolute time $t_\delta(\overrightarrow{d}, \overrightarrow{e}_\delta)$.

3 Representing Timed Automata in Timed μCRL

Timed automata [1, 4] can be represented in timed μCRL by associating a recursion variable with each location of the automaton as follows (see [19] for the initial idea).

Definition 3.1 (Timed Automaton). A *timed automaton* is a tuple $A = \langle L, l^0, \Sigma, C, i, E \rangle$, where L is a finite set of *locations*, $l^0 \in L$ is the *initial location*, Σ is a finite set of *transition labels*, C is a finite set of *clocks*, i is a mapping that assigns to each location an *invariant*, and E is a set of *transitions (edges)*. A transition is a quintuple $(l, a_e, \phi_e, \lambda_e, l_e)$ with l and $l_e \in L$ the start and end location of the transition, $a_e \in \Sigma$ the label of the transition, ϕ_e the *guard* associated with the transition, and $\lambda_e \subseteq C$ the set of clocks that are to be reset by the transition. All $\phi(e)$ and $i(l)$ are *clock constraint* formulas with the following syntax: $c \equiv n \mid c_1 - c_2 \equiv n \mid \phi_1 \wedge \phi_2$, where $\equiv \in \{<, \leq, =, \geq, >\}$ and $n \in \text{Nat}$.

The operational semantics of a timed automaton is defined as a transition system where a state consists of the current location and the current values of the clocks. There are two types of transitions between states. The automaton may either delay for some time (a delay transition), or follow an enabled transition (an action transition).

Definition 3.2 (Operational Semantics of Timed Automata). Given a finite set of clocks C , a *clock valuation* is

a function from C to non-negative real numbers \mathbb{R}^+ . A clock valuation v satisfies a clock constraint ϕ (notation $v \in \phi$) if $\phi[\vec{c} := v(\vec{c})]$ is equal to true. For any $d \in \mathbb{R}^+$, the clock valuation $v + d$ is defined as $(v + d)(c_i) = v(c_i) + d$. For any $\lambda \subseteq C$ the clock valuation v_λ is defined as $v_\lambda(c_i) = \text{if}(c_i \in \lambda, 0, v(c_i))$.

The semantics of a timed automaton $A = \langle L, l^0, \Sigma, C, i, E \rangle$ is a *relative time two-phased transition system* where states are pairs (l, v) for location $l \in L$ and a clock valuation v ; and transitions are defined by the rules:

- for any $d \in \mathbb{R}^+$, $(l, v) \xrightarrow{d} (l, v + d)$ if $v \in i(l)$ and $v + d \in i(l)$;
- for any $(l, a_e, \phi_e, \lambda_e, l_e) \in E$, $(l, v) \xrightarrow{a_e} (l_e, v_{\lambda_e})$ if $v \in \phi_e$ and $v_{\lambda_e} \in i(l_e)$.

Definition 3.3 (Representation of TA in timed μCRL). The following timed μCRL process equation for A_l is a translation of a location $l \in L$ of a timed automaton $A = \langle L, l^0, \Sigma, C, i, E \rangle$:

$$\begin{aligned} A_l(t^a : \text{Time}, v : \text{ClVals}) = & \\ & \sum_{e \in E_l} \sum_{t' : \text{Time}} a_e \cdot (t^a + t') \cdot A_{l_e}(t^a + t', (v + t')_{\lambda_e}) \\ & \triangleleft \text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t') \wedge \\ & \text{sat_cond}_{\phi_e}(v + t') \wedge \text{sat_inv}_{l_e}((v + t')_{\lambda_e}) \triangleright \delta \cdot \mathbf{0} \\ + \sum_{t' : \text{Time}} \delta \cdot (t^a + t') \triangleleft \text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t') \triangleright \delta \cdot \mathbf{0} \end{aligned}$$

where

- $E_l \subseteq E$ is the set of outgoing transitions from location l with the elements of the form $e = (l, a_e, \phi_e, \lambda_e, l_e)$;
- $\text{sat_inv}_l : \text{ClVals} \rightarrow \text{Bool}$ is, for $l \in L$, defined as $\text{sat_inv}_l(v) = i(l)[\vec{c} := v(\vec{c})]$;
- $\text{sat_cond}_{\phi_e} : \text{ClVals} \rightarrow \text{Bool}$ is, for $e \in E$, defined as $\text{sat_cond}_{\phi_e}(v) = \phi_e[\vec{c} := v(\vec{c})]$.

The process theory of timed μCRL uses absolute time instead of the relative time clocks of timed automata. As a consequence the above process equation is parameterized by a variable t^a representing the current *absolute* time. The parameter $v : \text{ClVals} \subseteq C \rightarrow \text{Time}$ represents the current values of the clocks (in *relative* time). We assume that the time domain *Time* is represented by \mathbb{R}^+ .

The conditions $\text{sat_inv}_l(v)$ and $\text{sat_inv}_l(v + t')$ express that the invariant of location l has to hold in the start state of the transition and in the state just before the transition is taken. Condition $\text{sat_cond}_{\phi_e}(v + t')$ expresses that the guard ϕ_e of transition e has to be satisfied at the moment transition e is taken, and condition $\text{sat_inv}_{l_e}((v + t')_{\lambda_e})$ means that the invariant $i(l_e)$ of the end location l_e of transition e has to be satisfied (after clock resets λ_e have been applied).

Proposition 3.4. *The syntax of the conditions is $v(c) \equiv n \mid v(c_1) - v(c_2) \equiv n \mid v(c) + t' \equiv n \mid \phi_1 \wedge \phi_2$, where $\equiv \in \{<, \leq, =, \geq, >\}$ and $n \in \text{Nat}^1$*

The operational semantics of timed μCRL processes is defined in [15]. In our concrete case the semantics of the system of equations for A_l corresponding to timed automaton

¹Or a condition can be equal to \mathbf{t} (in this case it disappears) or \mathbf{f} .

A in the *absolute time-stamped transition model* is defined as follows:

Definition 3.5 (Operational semantics of the timed μCRL representations of TAs). Let A_l be as defined before.

- for any $t' : \text{Time}$ and any $(l, a_e, \phi_e, \lambda_e, l_e) \in E_l$,
 $A_l(t^a, v) \xrightarrow{a_e}_{t^a+t'} A_{l_e}(t^a + t', (v + t')_{\lambda_e})$ if $\text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t') \wedge \text{sat_cond}_{\phi_e}(v + t') \wedge \text{sat_inv}_{l_e}((v + t')_{\lambda_e})$,
where transition relation $p \xrightarrow{a}_t p'$ expresses that the process p evolves into process p' by performing action a at time t ;
- for any $t' : \text{Time}$, $U_{t^a+t'}(A_l(t^a, v))$ if $\text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t')$, where the *delay relation* $U_t(p)$ expresses that process p can idle until time t at least.

Our timed μCRL representation corresponds to the standard semantics of timed automata [1, 4] in the following respect.

Theorem 3.6 (Correspondence between the operational semantics). *Let timed automaton A and its representation in timed μCRL A_l be as defined before. Then*

- for any $l, l' \in L$, $t, t' \in \text{Time}$ and $v, v' \in \text{ClVals}$ such that $A_l(t, v) \xrightarrow{a}_{t+t'} A_{l'}(t + t', v')$ the transition system of A contains the transitions $(l, v) \xrightarrow{t''} (l, v + t'')$ for any t'' such that $0 \leq t'' \leq t'$ and $(l, v + t') \xrightarrow{a} (l', v')$.
- for any $l \in L$, $t, t' \in \text{Time}$ and $v \in \text{ClVals}$ such that $U_{t+t'}(A_l(t, v))$ the transition system of A contains the transition $(l, v) \xrightarrow{t''} (l, v + t'')$ for any t'' such that $0 \leq t'' \leq t'$.
- for any $l, l' \in L$ and $v, v' \in \text{ClVals}$ such that $(l, v) \xrightarrow{a} (l', v')$ the transition system of A_l contains the transition: $A_l(u, v) \xrightarrow{a}_u A_{l'}(u, v')$ for any $u \in \text{Time}$.
- for any $l, l' \in L$ and $v, v' \in \text{ClVals}$ such that $(l, v) \xrightarrow{t} (l, v + t)$ and $(l, v + t) \xrightarrow{a} (l', v')$ the transition system of A_l contains the transition: $A_l(u, v) \xrightarrow{a}_{u+t} A_{l'}(u + t, v')$ for any $u \in \text{Time}$.
- for any $l \in L$, $t \in \text{Time}$ and $v \in \text{ClVals}$ such that $(l, v) \xrightarrow{t} (l, v + t)$ the transition system of A_l contains the predicate: $U_{u+t'}(A_l(u, v))$ for any $u, u' \in \text{Time}$ such that $0 \leq u' \leq t$.

In the future we would like to abstract from the precise distribution of the conditions, and just use the following two abstracted ones. The predicate $\text{is_enabled}_e(v, t')$ is an abstraction that says that the transition e is enabled. To simulate the timed automata setting we can define this predicate as $\text{is_enabled}_e(v, t') = \text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t') \wedge \text{sat_cond}_{\phi_e}(v + t') \wedge \text{sat_inv}_{l_e}((v + t')_{\lambda_e})$. Another predicate $\text{can_wait}_l(v, t')$ has to do with the second set of summands of the process A_l . If we define it as $\text{can_wait}_l(v, t') = \text{sat_inv}_l(v) \wedge \text{sat_inv}_l(v + t')$ we are back in the world of timed

automata. The resulting process X_l will look as follows:

$$\begin{aligned} X_l(t^a:Time, v:CIVals) = & \\ & \sum_{e \in E_l} \sum_{t^r:Time} \mathbf{a}_e \cdot (t^a + t^r) \cdot X_{l_e}(t^a + t^r, (v + t^r)_{\lambda_e}) \\ & \triangleleft is_enabled_e(v, t^r) \triangleright \delta \cdot \mathbf{0} \\ + & \sum_{t^r:Time} \delta \cdot (t^a + t^r) \triangleleft can_wait_l(v, t^r) \triangleright \delta \cdot \mathbf{0} \end{aligned}$$

4 Splitting into the Integral and Fractional Parts

4.1 Splitting Sums and Parameters

As the first step we shift the bound variable t^r with $fr(t^a)$ backwards in time to avoid the summing up of two fractional values t^a and t^r in the process description (namely at time stamps and at the new value of parameter t^a). We name the resulting bound variable u . Thus we perform a coordinate transformation where the old variable t^r and the new variable u are related according to the equation $u = fr(t^a) + t^r$. As the result we get the following equation for X'_l , where $fl(x)$ represents the ‘‘floor’’ function that returns the biggest integer less than or equal to x , and $fr(x)$ represents the ‘‘fraction’’ function that returns $x - fl(x)$:

$$\begin{aligned} X'_l(t^a:Time, v:CIVals) = & \\ & \sum_{e \in E_l} \sum_{u:Time} \mathbf{a}_e \cdot (fl(t^a) + u) \cdot X'_{l_e}(fl(t^a) + u, (v + (u - fr(t^a)))_{\lambda_e}) \\ & \triangleleft fr(t^a) \leq u \wedge is_enabled_e(v, u - fr(t^a)) \triangleright \delta \cdot \mathbf{0} \\ + & \sum_{u:Time} \delta \cdot (fl(t^a) + u) \\ & \triangleleft fr(t^a) \leq u \wedge can_wait_l(v, u - fr(t^a)) \triangleright \delta \cdot \mathbf{0} \end{aligned}$$

The additional condition $fr(t^a) \leq u$ is a result of applying the above coordinate transformation to the implicit condition $0 \leq t^r$.

Theorem 4.1. *For any $t^a:Time$ and $v:CIVals$:*

$$X_l(t^a, v) = X'_l(t^a, v).$$

From the previous proposition on the syntax of the conditions in X_l and the above coordinate transformation it is easy to see that the following also holds.

Proposition 4.2. *The syntax of the conditions is $v(c) \equiv n \mid v(c_1) - v(c_2) \equiv n \mid v(c) + (u - fr(t^a)) \equiv n \mid \phi_1 \wedge \phi_2$, where $\equiv \in \{<, \leq, =, \geq, >\}$ and $n \in Nat$.*

As a second step, we split the parameters t^a and v and the bound variable u in two parts: integral and fractional. This is done to abstract from the fractional parts, and show that only the *order relations* on them, but not the *exact values*, are important for the behavior of X'_l . An obvious thing to do would be to split each of these variables into its integral and fractional parts using the floor and fraction functions. We do it for t^a and u , namely $t^a = t_i^a + t_f^a$ for $t_i^a \in Nat$ and $t_f^a \in [0, 1)$ and $u = u_i + u_f$ for $u_i \in Nat$ and $u_f \in [0, 1)$. In this way, the new value of the parameter t^a of X'_l will have

the form $fl(t^a) + u = fl(t_i^a + t_f^a) + (u_i + u_f) = (t_i^a + u_i) + u_f$, which gives us the new value of the parameters t_i^a that depends on the integral parameters only, and t_f^a that is in $[0, 1)$.

Given the choice of the other parameters, we would also like that the parameter v is split into a discrete v_i and a fractional v_f in a way that the new value of v_i depends on the discrete parameters only, and the new value of v_f does not. It is clear that the obvious split would not do: the new value of v in X'_l is defined as $(v + (u - fr(t^a)))_{\lambda_e} = (v_i + v_f + (u_i + u_f - t_f^a))_{\lambda_e} = ((v_i + u_i) + (v_f + u_f - t_f^a))_{\lambda_e}$. This cannot be easily split into an integral and fractional parts because $v_f + u_f - t_f^a$ may be ≥ 1 , or ≤ 0 . So, to achieve our goal, we need something else.

Let us consider the absolute time when a clock c was reset for the last time. It can be expressed as $t^a - v(c)$ (abusing the notation we shall call it $l^r(c)$), so instead of keeping the (relative time) clock values v we could keep the (absolute time) values of the last clock resets l^r . The nice thing about this is that if we split l^r into l_i^r and l_f^r we get the desired property: the new value of $l^r(c)$ (according to X'_l) is $(fl(t^a) + u) - (v(c) + (u - fr(t^a)))_{\lambda_e}$ which in case $c \notin \lambda_e$ is equal to $fl(t^a) - v(c) + fr(t^a) = t^a - v(c) = l^r$ (the last reset time does not change in this case), and in case $c \in \lambda_e$ is equal to $(fl(t^a) + u)$ (the last reset time becomes the current time). In both cases the result can be easily split into an integral and a fractional part.

However, using the parameter l_i^r has a disadvantage because in this case we keep two absolute (discrete) time values, namely t_i^a and l_i^r . To improve the situation we keep the difference $t_i^a - l_i^r$ (abusing the notation, let us call it v_i) instead of l_i^r . So the values of the new parameters v_i and l_f^r can be computed in the following way: $l_f^r = fr(l^r) = fr(t^a - v)$ and $v_i = t_i^a - l_i^r = fl(t^a) - fl(t^a - v) = if(fr(t^a) \geq fr(v), fl(v), fl(v) + 1) = fl(v) + if(fr(t^a) \geq fr(v), 0, 1)$.

If we formulate the above coordinate transformation from the other direction we get:

$$\begin{aligned} t^a &= t_i^a + t_f^a & t_i^a &\in Nat & t_f^a &\in [0, 1) \\ v &= v_i + t_f^a - l_f^r & 0 &\leq l_f^r < 1, \\ u &= u_i + u_f & u_i &\in Nat & u_f &\in [0, 1). \end{aligned}$$

So we found a way to split all parameters so that the discrete and fractional parts are independent (maybe there is another way but we don't investigate it here). Let us consider the resulting equation and have a look at the conditions we get (maybe we can split them nicely as well?).

Application of the above data transformation on the process equation for X'_l results in the following process equation for X''_l .

$$\begin{aligned} X''_l(t_i^a:Nat, t_f^a:Time, v_i:CIValsN, l_f^r:CIVals) = & \\ & \sum_{e \in E_l} \sum_{u_i:Nat} \sum_{u_f:Time} \mathbf{a}_e \cdot (t_i^a + u_i + u_f) \cdot \\ & X''_{l_e}(t_i^a + u_i, u_f, (v_i + u_i)_{\lambda_e}, l_f^r) \\ & \triangleleft u_f < 1 \wedge (u_f \geq t_f^a \vee u_i > 0) \wedge \\ & is_enabled'_e(v_i, t_f^a, l_f^r, u_i, u_f) \triangleright \delta \cdot \mathbf{0} \end{aligned}$$

$$+ \sum_{u_i: \text{Nat}} \sum_{u_f: \text{Time}} \delta^c (t_i^a + u_i + u_f) \\ \triangleleft u_f < 1 \wedge (u_f \geq t_f^a \vee u_i > 0) \wedge \\ \text{can_wait}'_1(v_i, t_f^a, l_f^r, u_i, u_f) \triangleright \delta^c \mathbf{0}$$

where

- $l_f^r(c) = \text{if}(c \in \lambda_e, u_f, l_f^r(c))$ represents the new fractional values of the times the clocks were last reset (in case the clock is reset $l_f^r(c)$ becomes equal to the fractional value of the (new) current time, and it remains unchanged otherwise);
- the new conditions $\text{is_enabled}'_e(v, t_f^a, l_f^r, u_i, u_f)$ and $\text{can_wait}'_1(v, t_f^a, l_f^r, u_i, u_f)$ are respectively obtained from the X'_1 conditions $\text{is_enabled}_e(v, u - \text{fr}(t^a))$ and $\text{can_wait}_1(v, u - \text{fr}(t^a))$ by replacing the X'_1 parameters $\text{fl}(t^a)$, v and u by their representations with the parameters of X''_1 .

The following theorem states that the coordinate transformation described in this section is correct.

Theorem 4.3 (Correctness of splitting). *For any $t^a: \text{Time}$ and $v: \text{CIVals}$*

$$X'_1(t^a, v) = X''_1(\text{fl}(t^a), \text{fr}(t^a), \\ \text{fl}(v) + \text{if}(\text{fr}(t^a) \geq \text{fr}(v), 0, 1), \text{fr}(t^a - v))$$

Proposition 4.4. *The syntax of conditions $\text{is_enabled}'_e$ and $\text{can_wait}'_1$ is*

$$\begin{aligned} \phi &::= \text{if}(\chi_1 \equiv \chi_2, \psi_1, \psi_2) \mid (\chi_1 = \chi_2) \wedge \psi \mid \phi_1 \wedge \phi_2; \\ \chi &::= t_f^a \mid u_f \mid l_f^r(c); \\ \psi &::= v_i(c) \equiv n \mid v_i(c_1) - v_i(c_2) \equiv n \mid v_i(c) + u_i \equiv n, \end{aligned}$$

where $\equiv \in \{<, \leq, =, \geq, >\}$ and $n \in \text{Nat}$.

Proof. Given the specific syntax of the conditions in X'_1 we can consider the following cases:

- for the case of $v(c) < n$ constraint we get $v_i(c) + t_f^a - l_f^r(c) < n$, which is equivalent to $v_i(c) < n \vee (v_i(c) = n \wedge t_f^a < l_f^r(c))$, which is in turn equivalent to $\text{if}(t_f^a < l_f^r(c), v_i(c) \leq n, v_i(c) < n)$;
- for the case of $v(c) \leq n$ constraint we get $v_i(c) + t_f^a - l_f^r(c) \leq n$ which is equivalent to $v_i(c) < n \vee (v_i(c) = n \wedge t_f^a \leq l_f^r(c))$, which is in turn equivalent to $\text{if}(t_f^a \leq l_f^r(c), v_i(c) \leq n, v_i(c) < n)$;
- for the case $v(c) = n$ we get $v_i(c) + t_f^a - l_f^r(c) = n \approx v_i(c) = n \wedge t_f^a = l_f^r(c)$;
- similarly for $v(c) \geq n$ and $v(c) > n$ we get the needed form;
- for the case of $v(c_1) - v(c_2) < n$ constraint we get $(v_i(c_1) + t_f^a - l_f^r(c_1)) - (v_i(c_2) + t_f^a - l_f^r(c_2)) < n$ which is equivalent to $(v_i(c_1) - v_i(c_2)) - l_f^r(c_1) + l_f^r(c_2) < n$, or equivalently $(v_i(c_1) - v_i(c_2)) < n \vee ((v_i(c_1) - v_i(c_2)) = n) \wedge l_f^r(c_1) > l_f^r(c_2)$, which is in turn equivalent to $\text{if}(l_f^r(c_2) < l_f^r(c_1), v_i(c_1) - v_i(c_2) \leq n, v_i(c_1) - v_i(c_2) < n)$.
- similarly for the other $v(c_1) - v(c_2) \equiv n$ cases we get the right forms;
- for the case of $v(c) + (u - \text{fr}(t^a)) < n$ constraint we get $(v_i(c) + t_f^a - l_f^r(c)) + (u_i + u_f - t_f^a) < n$, which is

equivalent to $(v_i(c) + u_i) - l_f^r(c) + u_f < n$, or equivalently $(v_i(c) + u_i) < n \vee (v_i(c) + u_i = n \wedge u_f < l_f^r(c))$, or $\text{if}(u_f < l_f^r(c), v_i(c) + u_i \leq n, v_i(c) + u_i < n)$;

- similarly for the other $v(c) + (u - \text{fr}(t^a)) \equiv n$ cases we get the right forms. \square

4.2 Splitting Conditions

It is visible from the syntax of the conditions that the actual values of the real-valued parameters (t_f^a and l_f^r) and the bound variable u_f are not important, but the relations between pairs of them may be. Therefore we introduce an abstraction of these parameters and use this abstraction instead of the real-valued parameters in the conditions. This corresponds to the use of regions in timed automata ([1]).

Let c_0 be a new clock name ($c_0 \notin C$) and let C^0 be $C \cup \{c_0\}$. Let us extend the vector l_f^r to c_0 and set $l_f^r(c_0) := t_f^a$. We use the domain Ord and the predicate $\text{is_leq} : \text{Ord} \times C^0 \times C^0 \rightarrow \text{Bool}$ to represent the ordering on the values of the l_f^r element (and t_f^a being represented by $l_f^r(c_0)$) with respect to their equality and the \leq relation. We want $\text{is_leq}(\text{ord}, c_1, c_2)$ to represent the fact that according to the ordering ord it holds that $l_f^r(c_1) \leq l_f^r(c_2)$. This domain can be implemented as a subset of functions from $C^0 \rightarrow \{0, \dots, |C|\}$, and $\text{is_leq}(\text{ord}, c_1, c_2)$ can be implemented as $\text{ord}(c_1) \leq \text{ord}(c_2)$. The other conditions is_cond can be defined from is_leq in the usual way.

We also use the domain Pos to represent a position in the ordering, which can be a position of an element or a position in-between two consecutive element in the ordering, as well as before the first and after the last elements. The functions $\text{is_pos_leq} : \text{Ord} \times \text{Pos} \times C^0 \rightarrow \text{Bool}$ and $\text{is_pos_eq} : \text{Ord} \times C^0 \times \text{Pos} \rightarrow \text{Bool}$ are defined in a way that $\text{is_pos_leq}(\text{ord}, \text{pos}, c)$ represents that, according to the ordering ord and a position pos in it, $u_f \leq l_f^r(c)$ holds, and $\text{is_pos_eq}(\text{ord}, c, \text{pos})$ represents that $l_f^r(c) = u_f$ holds. This domain can be implemented as a subset of $\{0, \dots, |C| + 1\} \times \text{Bool}$, where in position (p, b) the element p represents a position in the ordering and the element b says if (p, b) is at the position p (by \mathbf{t}) or just before it. The function $\text{is_pos_leq}(\text{ord}, (p, b), c)$ could be implemented as $p \leq \text{ord}(c)$ and $\text{is_pos_eq}(\text{ord}, c, (p, b))$ as $\neg b \wedge p = \text{ord}(c)$. The other conditions is_pos_cond can be defined from the above two in the usual way.

We also use the predicates $\text{confirm_ord}(\text{ord}, t_f^a, l_f^r)$ and $\text{confirm_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, u_f)$ to respectively represent the facts that the ordering ord preserves the actual relations on t_f^a and l_f^r , and the fact that u_f conforms to its position pos in the ordering ord . They can be defined in the following way (remembering that $l_f^r(c_0)$ is defined as t_f^a):

$$\begin{aligned} \text{confirm_ord}(\text{ord}, t_f^a, l_f^r) &:= \\ \forall_{c_1, c_2 \in C^0} l_f^r(c_1) \leq l_f^r(c_2) &\rightarrow \text{is_leq}(\text{ord}, c_1, c_2) \end{aligned}$$

and

$$\begin{aligned} \text{confirm_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, u_f) &:= \\ \forall c \in C^0 (u_f \leq l_f^r(c) \rightarrow \text{is_pos_leq}(\text{ord}, \text{pos}, c)) \wedge \\ (u_f = l_f^r(c) \rightarrow \text{is_pos_eq}(\text{ord}, \text{pos}, c)) \end{aligned}$$

The resulting process X_l''' will look as described in Table 1, where

- $\text{upd_ord}(\text{ord}, \text{pos}, \lambda_e)$ gives the new ordering based on the old one, the position of u_f and the clock resets. The order of the clocks that are not reset do not change; the new position of t_f^a and the clocks that are reset will be the position of u_f .
- the conditions of X_l''' $\text{is_enabled}_e''(v_i, u_i, \text{ord}, \text{pos})$ and $\text{can_wait}_l''(v_i, u_i, \text{ord}, \text{pos})$ are respectively obtained from the X_l' conditions $\text{is_enabled}_e'(v_i, t_f^a, l_f^r, u_i, u_f)$ and $\text{can_wait}_l'(v_i, t_f^a, l_f^r, u_i, u_f)$ by replacing the relations $l_f^r(c_i) \text{ cond } l_f^r(c_j)$ with $\text{is_cond}(\text{ord}, c_i, c_j)$; the relation $t_f^a \text{ cond } l_f^r(c_j)$ with $\text{is_cond}(\text{ord}, c_0, c_j)$; and the relation $u_f \text{ cond } l_f^r(c_j)$ with $\text{is_pos_cond}(\text{ord}, \text{pos}, c_j)$.

Obviously, the following proposition will hold.

Proposition 4.5. *The syntax of the conditions is*

$$\begin{aligned} \phi &::= \phi_1 \wedge \phi_2 \mid \text{if}(\text{is_cond}(\text{ord}, c_1, c_2), \psi_1, \psi_2) \\ &\mid \text{if}(\text{is_pos_cond}(\text{ord}, \text{pos}, c), \psi_1, \psi_2) \\ &\mid \text{is_eq}(\text{ord}, c_1, c_2) \wedge \psi \mid \text{is_pos_eq}(\text{ord}, \text{pos}, c) \wedge \psi, \\ \psi &::= v_i(c) \equiv n \mid v_i(c_1) - v_i(c_2) \equiv n \mid v_i(c) + u_i \equiv n \end{aligned}$$

where $\equiv \in \{<, \leq, =, \geq, >\}$ and $n \in \text{Nat}$.

Theorem 4.6. *Suppose that $\text{upd_ord}(\text{ord}, \text{pos}, \lambda_e)$ is defined in such a way that for all $\text{ord}:\text{Ord}$, $t_f^a:\text{Time}$ and $l_f^r:\text{CIVals}$ such that $\text{conform_ord}(\text{ord}, t_f^a, l_f^r)$ and for all $\text{pos}:\text{Pos}$ and $u_f:\text{Time}$ such that $u_f < 1 \wedge \text{conform_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, u_f)$ and for all $\lambda_e \in C$ we have*

$$\text{conform_ord}(\text{upd_ord}(\text{ord}, \text{pos}, \lambda_e), u_f, l_f^r).$$

Then $X_l''(t_i^a, t_f^a, v_i, l_f^r) = X_l'''(t_i^a, v_i, \text{ord}, t_f^a, l_f^r)$ for all parameters $\text{ord}:\text{Ord}$, $t_f^a:\text{Time}$ and $l_f^r:\text{CIVals}$ such that $\text{conform_ord}(\text{ord}, t_f^a, l_f^r)$.

5 Abstraction from the Fractional Parts

Suppose we are not interested in the fractional parts of the action and the deadlock time stamps. E.g. we replace $\mathbf{a}_e \cdot (t_i^a + u_i + u_f)$ by $\mathbf{a}_e \cdot (t_i^a + u_i)$ in X_l''' . The resulting process variable we call Y_l . For this process we prove that the exact values of the parameters t_f^a and l_f^r do not influence the behavior of Y_l .

Lemma 5.1. *For any $\text{ord}:\text{Ord}$, $t_f^{a1}, t_f^{a2}:\text{Time}$ and $l_f^{r1}, l_f^{r2}:\text{CIVals}$ such that $\text{conform_ord}(\text{ord}, t_f^{a1}, l_f^{r1})$ and $\text{conform_ord}(\text{ord}, t_f^{a2}, l_f^{r2})$ we have that for any $t_i^a:\text{Nat}$ and $v_i:\text{CIValsN}$*

$$Y_l(t_i^a, v_i, \text{ord}, t_f^{a1}, l_f^{r1}) = Y_l(t_i^a, v_i, \text{ord}, t_f^{a2}, l_f^{r2})$$

Proof. We use the fact that in this case for any $u_f:\text{Time}$

$$\begin{aligned} \text{conform_pos}(\text{ord}, \text{pos}, t_f^{a1}, l_f^{r1}, u_f) &= \\ \text{conform_pos}(\text{ord}, \text{pos}, t_f^{a2}, l_f^{r2}, u_f) \end{aligned}$$

and the fact that t_f^a does not occur anywhere else, and l_f^r may only occur in the new value of itself. \square

Now we apply *sum elimination* (cf. [12]) to Y_l in order to get rid of the summation with u_f and the condition conform_pos . As a result we obtain the process equation for Y_l' in Table 1 where $\text{get_trf}(\text{ord}, \text{pos}, t_f^a, l_f^r) < 1$ is any such value that $\text{conform_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, \text{get_trf})$; and $l_f^r''(c) = \text{if}(c \in \lambda_e, \text{get_trf}, l_f^r(c))$.

Theorem 5.2. *For any $\text{ord}:\text{Ord}$, $t_f^a:\text{Time}$ and $l_f^r:\text{CIVals}$ such that $\text{conform_ord}(\text{ord}, t_f^a, l_f^r)$ and for any $t_i^a:\text{Nat}$ and $v_i:\text{CIValsN}$*

$$Y_l'(t_i^a, v_i, \text{ord}, t_f^a, l_f^r) = Y_l(t_i^a, v_i, \text{ord}, t_f^a, l_f^r)$$

Proof. For this we use the fact that the *Time* domain is dense and for every $t_i^a, \text{pos}, \text{ord}$ such that $(u_i > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0))$ and for every t_f^a and l_f^r such that $\text{conform_ord}(\text{ord}, t_f^a, l_f^r)$, there exists a $u_f < 1$ such that $\text{conform_pos}(\text{ord}, \text{pos}, u_f)$, and say $\text{get_trf}(\text{ord}, \text{pos}, t_f^a, l_f^r)$ is such a u_f . \square

Finally, we apply *parameter elimination* (cf. [10]) to the last two parameters. As a result we get the process equation for Y_l'' .

Theorem 5.3. *For any $\text{ord}:\text{Ord}$, $t_f^a:\text{Time}$ and $l_f^r:\text{CIVals}$ such that $\text{conform_ord}(\text{ord}, t_f^a, l_f^r)$ and for any $t_i^a:\text{Nat}$ and $v_i:\text{CIValsN}$*

$$Y_l''(t_i^a, v_i, \text{ord}) = Y_l'(t_i^a, v_i, \text{ord}, t_f^a, l_f^r).$$

6 Making the state space finite

In this section we use the fact that the clocks and clock differences are only compared to constants. By taking the maximal such constant we can limit the natural-valued variables from above. For each clock c_i we define max_i to be the maximal constant this clock is compared to in $\text{is_enabled}_e''(v_i, u_i, \text{ord}, \text{pos})$ or $\text{can_wait}_l''(v_i, u_i, \text{ord}, \text{pos})$, incremented by 1 (0 if the clock is never compared to a constant). For each pair of clocks c_i and c_j we define max_{ij} to be the maximal constant the difference of c_i and c_j (or c_j and c_i) is compared to, incremented by 1 (0 if the clock difference is never compared to a constant). We write vmax for the vector of max_i and Mmax for the symmetric matrix of max_{ij} . Without loss of generality we assume that all conditions of the form $c_i - c_j < 0$ have been replaced with $c_j - c_i > 0$, all conditions $c_i - c_j \leq 0$ with $c_j - c_i \geq 0$, and all conditions of the form $c_i - c_j = 0$ with $c_i - c_j \geq 0 \wedge c_j - c_i \geq 0$. This allows us to use 0 instead of the negative values of clock differences.

Table 1. Equation for X_l'''' , Y_l' , Y_l'''' , Z_l , and T_l . The equations for Y_l , Y_l' , and T_l' can be obtained by removing the boxed parts.

$$\begin{aligned}
X_l''''(t_i^a: \text{Nat}, v_i: \text{CIValsN}, \text{ord}: \text{Ord}, t_f^a: \text{Time}, l_f^r: \text{CIVals}) = & \\
& \sum_{e \in E_1} \sum_{u_i: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \left(\sum_{u_f: \text{Time}} \mathbf{a}_e \circ (t_i^a + u_i \boxed{+u_f}) \cdot X_{l_e}''''(t_i^a + u_i, (v_i + u_i)_{\lambda_e}, \text{upd_ord}(\text{ord}, \text{pos}, \lambda_e), u_f, l_f^r) \right. \\
& \quad \left. \triangleleft u_f < 1 \wedge \text{conform_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, u_f) \triangleright \delta \circ \mathbf{0} \right) \\
& \quad \triangleleft (u_i > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{is_enabled}''_e(v_i, u_i, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0} \\
+ \sum_{u_i: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \left(\sum_{u_f: \text{Time}} \delta \circ (t_i^a + u_i \boxed{+u_f}) \triangleleft u_f < 1 \wedge \text{conform_pos}(\text{ord}, \text{pos}, t_f^a, l_f^r, u_f) \triangleright \delta \circ \mathbf{0} \right) \\
& \quad \triangleleft (u_i > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{can_wait}''_l(v_i, u_i, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
Y_l'(t_i^a: \text{Nat}, v_i: \text{CIValsN}, \text{ord}: \text{Ord}, \boxed{t_f^a: \text{Time}, l_f^r: \text{CIVals}}) = & \\
& \sum_{e \in E_1} \sum_{u_i: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \mathbf{a}_e \circ (t_i^a + u_i) \cdot Y_{l_e}'(t_i^a + u_i, (v_i + u_i)_{\lambda_e}, \text{upd_ord}(\text{ord}, \text{pos}, \lambda_e), \boxed{\text{get_trf}, l_f^r}) \\
& \quad \triangleleft (u_i > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{is_enabled}''_e(v_i, u_i, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0} \\
+ \sum_{u_i: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \delta \circ (t_i^a + u_i) \triangleleft (u_i > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{can_wait}''_l(v_i, u_i, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
Y_l''''(t^a: \text{Nat}, v: \text{CIValsN}, M: \text{CIValsNN}, \text{ord}: \text{Ord}) = & \\
& \sum_{e \in E_1} \sum_{u: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \mathbf{a}_e \circ (t^a + u) \cdot Y_{l_e}''''(t^a + u, \min(v', v_{\max}), \min(M', M_{\max}), \text{upd_ord}(\text{ord}, \text{pos}, \lambda_e)) \\
& \quad \triangleleft (u > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{is_enabled}''''_e(v, M, u, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0} \\
+ \sum_{u: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \delta \circ (t^a + u) \triangleleft (u > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{can_wait}''''_l(v, M, u, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
Z_l(t^a: \text{Nat}, v: \text{CIValsN}, M: \text{CIValsNN}, \text{ord}: \text{Ord}) = & \\
& \sum_{e \in E_1} \sum_{u: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \mathbf{a}_e \circ (t^a + \min(u, c_{\max})) \cdot Z_{l_e}(t^a + u, \min(v', v_{\max}), \min(M', M_{\max}), \text{upd_ord}(\text{ord}, \text{pos}, \lambda_e)) \\
& \quad \triangleleft (u > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{is_enabled}''''_e(v, M, u, \text{ord}, \text{pos}) \triangleright \delta \circ \mathbf{0} \\
+ \delta \triangleleft \max_d(v, M, \text{ord}) = c_{\max} \triangleright \delta \circ \mathbf{0} \\
+ \delta \circ (t^a + \max_d(v, M, \text{ord}, \text{pos})) \triangleleft 0 < \max_d(v, M, \text{ord}) < c_{\max} \triangleright \delta \circ \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
T_l(\boxed{t^a: \text{Nat}}, v: \text{CIValsN}, M: \text{CIValsNN}, \text{ord}: \text{Ord}) = & \\
& \sum_{e \in E_1} \sum_{u: \text{Nat}} \sum_{\text{pos}: \text{Pos}} \mathbf{a}_e(\min(u, c_{\max})) \cdot T_{l_e}(\boxed{t^a + u}, \min(v', v_{\max}), \min(M', M_{\max}), \text{upd_ord}(\text{ord}, \text{pos}, \lambda_e)) \\
& \quad \triangleleft (u > 0 \vee \text{is_pos_geq}(\text{ord}, \text{pos}, c_0)) \wedge \text{is_enabled}''''_e(v, M, u, \text{ord}, \text{pos}) \triangleright \delta \\
+ \Delta \triangleleft \max_d(v, M, \text{ord}) = c_{\max} \triangleright \delta \\
+ \Delta(\max_d(v, M, \text{ord}, \text{pos})) \triangleleft 0 < \max_d(v, M, \text{ord}) < c_{\max} \triangleright \delta
\end{aligned}$$

We use v_{\max} and M_{\max} to limit the values of u_i and v_i . In the first step we just limit v_i because this preserves the process equivalence. We shall drop the i index from t_i^a , v_i and u_i because no fractional parts are present any more.

The result is Y_l'''' as described in Table 1, where $M'_{i,j}$ is defined as $\text{if}(c_i \in \lambda_e, 0, M_{i,j}) + \text{if}(c_j \in \lambda_e, u, 0)$; and the conditions $\text{is_enabled}''_e$ and $\text{can_wait}''_l$ are defined from

$\text{is_enabled}''_e$ and $\text{can_wait}''_l$, respectively, by replacing every condition $v(c_i) - v(c_j) \equiv n$ with $M_{i,j} \equiv n$.

Theorem 6.1. *Let the matrix M be defined as $M_{i,j} := \min(\max(0, v(c_i) - v(c_j)), \max_{i,j})$.*

Then for any $t^a: \text{Nat}$, $v: \text{CIValsN}$ and $\text{ord}: \text{Ord}$

$$Y_l''''(t^a, \min(v, v_{\max}), M, \text{ord}) = Y_l''(t^a, v, \text{ord}).$$

In order to limit the values of u we need to drop more timing information from the actions (because u occurs there). We shall also drop the absolute time of the action t^a because it may grow unbounded, and its exact value cannot be determined if we limit the value of u . Let $cmax$ be defined as the maximal value of the elements in $vmax$.

For the δ summands we have to apply a different procedure. First we note that the process always can idle till the current time t^a . Moreover, if it can idle till $t^a + cmax$, then it can idle forever. If not, then it may idle till some time $t^a + max_d$ with max_d ranging from 0 to $cmax - 1$. The exact value of max_d can be defined as

$$max_d(v, M, ord) := \max(\{u \in Nat \mid \exists pos: Pos \text{ can_wait}'(v, M, u, ord, pos)\} \cup \{cmax\})$$

Given the boundedness of the maximum operator and the quantifiers, this formula can be expressed in the data syntax of μCRL . In case $max_d = 0$ we do not need a δ summand at all, and in case $max_d = cmax$, the process can wait forever, which is covered by a separate case.

Starting from the process Y_l''' we obtain the process Z_l by limiting the values of u , and from this process we get T_l by performing a *relativization* step. This step allows us to go from an absolute-time timed μCRL process to its *relativization* in untimed μCRL , preserving the process equivalence relations.

In the process T_l (Table 1) we get rid of the last unbounded parameter t^a and call the resulting process T_l' . They are related in the following way.

Theorem 6.2. *For any $t^a: Nat$, $v: CValsN$, $M: CValsNN$, and $ord: Ord$*

$$T_l'(v, M, ord) = T_l(t^a, v, M, ord).$$

Proof. By eliminating the first parameter in T_l . \square

This equation T_l' always has a finite number of reachable states. It corresponds to the region automaton of the initial time automaton.

7 Conclusions and Future Work

In this paper we transformed a timed μCRL process equation representing a timed automaton into a closely related timed μCRL process equation with finite discrete parameters and bound variables only. This could enable simulation and verification via enumeration of reachable states. As a result, some of the existing untimed analysis tools in the μCRL Toolset [7] could become applicable to the analysis of real-time systems.

As the next step we would like to factorize the remaining time-related parameters to be able to deal with them like with zones. Zones, as well as the operations on them could be specified as an abstract data type in μCRL , either as a set of clock constraints or using difference-bound matrices.

Another direction for future work is an adaptation of the presented technique to various extensions of timed automata available in UPPAAL [14], like *networks of timed*

automata, shared variables, urgent transitions and committed locations. This could enable a possibility to analyze UPPAAL specifications with the μCRL Toolset.

Going further, one could analyze where exactly the fact that we are dealing with timed automata has been used and try to extend some of the results to a more general setting, perhaps lifting some restrictions on the clock constraints, or moving into the direction of *hybrid automata*.

References

- [1] R. Alur. Timed automata. In *Proc. CAV'99*, LNCS 1633, pages 8–22, 1999.
- [2] J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing*. Monographs in TCS. Springer, 2002.
- [3] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge Tracts in TCS 18. Cambridge University Press, 1990.
- [4] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. Technical Report 316, UNU-IIST, P.O.Box 3058, Macau, September 2004.
- [5] J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press, ACM Press Frontier Series, 1989.
- [6] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.
- [7] S. Blom, W. J. Fokkink, J. F. Groote, I. A. v. Langevelde, B. Lissner, and J. C. v. d. Pol. μCRL : A toolset for analysing algebraic specifications. In G. Berry, H. Comon, and A. Finkel, editors, *Proc CAV'01*, volume 2102 of LNCS, pages 250–254. Springer, 2001.
- [8] S. N. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Number 78 in Graduate Texts in Mathematics. Springer-Verlag, 1981.
- [9] J. F. Groote. The syntax and semantics of timed μCRL . Report SEN-R9709, CWI, Amsterdam, 1997.
- [10] J. F. Groote and B. Lissner. Computer assisted manipulation of algebraic process specifications. *SIGPLAN Notices*, 37(12):98–107, 2002.
- [11] J. F. Groote and A. Ponse. The syntax and semantics of μCRL . In A. Ponse, C. Verhoef, and S. F. M. v. Vlijmen, editors, *Algebra of Communicating Processes 1994*, Workshop in Computing, pages 26–62. Springer, 1995.
- [12] J. F. Groote and M. A. Reniers. Algebraic process verification. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 17, pages 1151–1208. Elsevier, 2001.
- [13] J. F. Groote and J. J. v. Wamel. Algebraic data types and induction in μCRL . Report P9409, University of Amsterdam, Programming Research Group, 1994.
- [14] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
- [15] M. A. Reniers, J. F. Groote, J. J. v. Wamel, and M. B. v. d. Zwaag. Completeness of Timed μCRL . *Fund. Inf.*, 50(3-4):361–402, 2002.
- [16] M. A. Reniers and Y. S. Usenko. Analysis of timed processes with data using algebraic transformations. In *Proc. TIME'05*, pages 192–194. IEEE Computer Society, 2005.
- [17] Y. Sun. An algebraic generalization of Frege structures — binding algebras. *TCS*, 211:189–232, 1999.
- [18] Y. S. Usenko. *Linearization in μCRL* . PhD thesis, Eindhoven University of Technology, December 2002.
- [19] T. A. C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. PhD thesis, Eindhoven University of Technology, 2003.