Achieving Discrete Relative Timing with Untimed Process Algebra

A.J. Wijs
CWI, Department of Software Engineering,
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
wijs@cwi.nl

Abstract

For many systems, timing aspects are essential. Therefore, when modelling these systems, time should somehow be represented. In the past, many timed process algebras have been developed, using untimed process algebras as initial inspiration. In this paper, we take another approach, considering the possibility to model timing aspects with an untimed process algebra. The advantage is that the algebra itself does not need to be extended, and the available tools can be reused. In contrast to other work, where this approach has been looked at, we focus on ease of modelling, and single delay steps of varying sizes. We present the timing mechanism used, our approach, and some examples.

1 Introduction

Model checking has proven to be very useful in finding bugs in embedded system specifications. μCRL [13, 23], for instance, has been used to verify properties of many systems and protocols. Many cases, however, are time-critical, meaning that time should also play a role in specifications of those systems, in order to be able to check relevant properties. Over the years, the inclusion of time in modelling languages has shown to be complex, both on a theoretical and on a practical level. As can be found in the literature, in theory, subjects like the extension of modelling languages with time [1, 2, 17, 20] and the design of relations between systems such as timed branching bisimilarity [9, 24] are very complicated, and at times difficult to get, and prove, correct. On the practical side, a major problem in model checking is the so-called *state explosion problem*, meaning that a linear growth of the number of processes placed in parallel in a specification leads to an exponential growth of the resulting state space. Adding time to a specification makes this problem even more difficult, often leading to infinite state spaces, such as when an action is allowed to happen at any time.

In the past, based on the modelling language μCRL, which is basically the process algebra ACP [4] with abstract data types [15], a timed language, called *timed* μCRL [11], has been developed. For practical reasons, one of which is the aforementioned tendency of state spaces to be infinite, there are currently no tools for that language yet.

In [6, 14], another approach was taken. There, the possibility to model time in regular, untimed μCRL was investigated, which would enable modellers to use the fully developed and highly optimised μCRL toolset [5] when dealing with timed systems. Moreover, existing relations between systems, such as branching bisimilarity [10], can then be checked on timed systems. What they finally presented was a framework, a recipe, to express processes involving some notion of time. It is very important that a modeller follows this recipe faithfully, otherwise unwanted and bizarre timing behaviour might occur, such as the violation of principles like time determinism [2] and maximal progress [2], to which we will return later in this paper. Besides that, a delay of one time unit corresponds directly with one transition in the system, resulting in large sequences of delays whenever a big time jump has to be made.

After that, in [22], this recipe was used as an inspiration for a translation scheme from the timed language χ_t [3] to μCRL. By this, it was again shown that time could be modelled using an untimed modelling language, but in some cases, most notably when using alternative composition, it led to complex process terms. The complexity of the resulting process was not so problematic, since it was the result of an automatic translation, but a high complexity cannot be demanded when a modeller has to create the term directly.

The work on modelling time in [6, 14] and the translation scheme in [22] inspired us to investigate the possibilities of modelling time in an untimed process algebra, such that:

1. The resulting timing mechanism makes sense, i.e.

principles such as time determinism and maximal progress hold.

2. The modeller can use the process algebra as freely as when modelling untimed systems.

3. Arbitrarily big time jumps can be made in a single transition, provided that the system cannot fire any action during the time interval.

4. The existing tools can be applied on the timed systems, i.e. relations, such as branching bisimilarity, and properties, expressed using temporal logics, can be checked.

The creation of this paper was a back and forth of designing a timing mechanism on the one hand and trying to model it in μCRL on the other. On a number of occasions the mechanism had to be changed somewhat due to the limitations of modelling, while still making sure that the mechanism remained reasonable from the perspective of the theory of timed systems.

In this paper, we start with an explanation of μCRL, and a discussion of the timing mechanism we wish to have, leading to, as we call it, the extended language μCRL^{tick}. Then we explain how this mechanism is achieved for a μCRL^{tick} specification through the transformation to a μCRL specification. The correctness proofs can be found in appendix A. Finally, we show some examples, briefly discuss possible extensions and future work, and provide a conclusion.

Contributions We propose a way to achieve time through modelling in an untimed process algebra. Contrary to earlier attempts, we focus on ease of modelling and time jumps of arbitrary size, the latter since it often practically leads to smaller state spaces. We achieve this by the introduction of a timed extension of the algebra and a transformation to the original. Through doing so, this paper offers more insight into the relation between untimed and timed process algebras.

Related Work There are many papers on extending a specific process algebra with time. Here, we focus on papers on the basic timing concepts and comparisons of timing mechanisms. In [2, 17], a range of timing mechanisms are discussed and compared according to a list of time properties (mentioned in this paper in section 3); additional axioms and transition rules are provided for a number of process algebras, and extra operators are introduced. Some of the time properties are further elaborated on in [1]. A general framework for designing timed process languages is proposed in [20]. The idea in these papers is to embed untimed into timed process algebra; in a sense one could say that we take an opposite approach, namely embedding timed into untimed process algebra, by means of a transformation procedure.

2 Preliminaries

μCRL The language μCRL is based on the process algebra ACP [4], extended with equational abstract data types [15]. It comes with a toolset [5] that can build a state space from a specification and store it in the `.aut` format, one of the input formats of the model checker CADP [7]. Next to that, in order to strive for precision in proofs, an important research area is to use theorem provers such as PVS [18] to help in finding and checking derivations in μCRL. A large number of distributed systems have been verified in μCRL, often with the help of a proof checker or theorem prover, e.g. [8, 12].

We will give a short overview of the language necessary for understanding this paper. For a complete reference, see [13, 23].

Definition 1 (μCRL specification). *We define a μCRL specification $\mathcal{M}$ as a sextuple $(\mathcal{D}, \mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{P}, \mathcal{I})$, where*

- *$\mathcal{D}$ is the set of data domains used;*
- *$\mathcal{F}$ is the set of functions defined over the data domains in $\mathcal{D}$;*
- *$\mathcal{A}$ is the set of actions used;*
- *$\mathcal{C}$ is the set of communication rules for the actions;*
- *$\mathcal{P}$ is the set of processes in the specification;*
- *$\mathcal{I}$ is the initialisation line, combining and initialising the processes.*

Following is a more detailed description of each element in the sextuple.

Data domains and functions In order to intertwine processes with data, actions and recursion variables can be parametrised with data types. Each specification should start by defining the necessary data types and the functions that work on them. In fact, it is mandatory to define the boolean type in each specification, since the conditional construct, wich is one of the μCRL operators, works with boolean expressions. One can virtually define any data type. In this paper we assume the presence of the domains of the booleans ($\mathbb{B}$) and the natural numbers ($\mathbb{N}$), and definitions of

the most common functions, such as equality. In other words, for a specification $\mathcal{M}$, $\mathbb{B} \in \mathcal{D}$, $\mathbb{N} \in \mathcal{D}$, and, for example, $(=: \mathbb{N} \times \mathbb{N} \to \mathbb{B}) \in \mathcal{F}$. It should also be noted, that a variable d of type $\mathbb{D}$ is written as $d : \mathbb{D}$, while an element of $\mathbb{D}$ is written as $d \in \mathbb{D}$, e.g. $x : \mathbb{N}$ and $2 \in \mathbb{N}$.

Actions In μCRL one can declare actions. These actions may have zero, one or several data parameters. In this paper, we denote actions a, b, etc. appearing in a specification $\mathcal{M}$ as being elements of $\mathcal{A}$. Finally, the process deadlock (δ), which cannot terminate successfully, and the internal action τ are predefined, and $\tau, \delta \notin \mathcal{A}$.

Communication rules It is possible to define communication rules for actions. For instance, for $a, b, c \in \mathcal{A}$, one can define the rule $a \mid b = c$, meaning that a and b can synchronise with each other, forming action c. For every rule $a \mid b = c$, we have $(a, b, c) \in \mathcal{C}$. Communication can only take place, if the data parameters of a and b have the same types and values. The rules are both commutative and associative; for instance, $(a, b, c) \in \mathcal{C}$ iff $(b, a, c) \in \mathcal{C}$.

Processes Processes can be created by combining actions from $\mathcal{A}$ and other processes using a given set of operators and guarded recursion. Next, we will give an informal description of the operators used, followed by a description how processes are defined in μCRL. We define a special process $\checkmark \in \mathcal{P}$, which immediately terminates successfully.

Operators There are four operators for creating processes in μCRL.

1. The alternative composition operator ($+$). A process $P + Q$ proceeds (non-deterministically) as P or Q (if they can proceed).

2. The sum operator ($\sum_{d:\mathbb{D}} X(d)$), with $X(d)$ a mapping from domain $\mathbb{D} \in \mathcal{D}$ to $\mathcal{P}$, behaves as $X(d_1) + X(d_2) + \ldots$, i.e. as the possibly infinite choice between $X(d)$ for any data term d taken from $\mathbb{D}$. This operator is mostly used to describe a process that is reading some input over a data type [16].

3. The sequential composition operator ($\cdot$). A process $P \cdot Q$ proceeds as P which upon successful termination is followed by Q.

4. The process expression $P \lhd b \rhd Q$ where $P, Q \in \mathcal{P}$, and $b : \mathbb{B}$, behaves as P if b is equal to T(true) and behaves as Q if b is equal to F(false). This operator is called the conditional operator.

Linear Process Equations The heart of $\mathcal{M}$ is $\mathcal{P}$, where the behaviour of the system is declared. This set consists of recursion equations which are rewritable to the normal form called *linear process equation* (LPE) [23]. In essence an LPE is a vector of data parameters together with a list of summands consisting of a condition, action and effect triple, describing when an action may happen and what its effect is on the vector of data parameters. It is of the following form:

$$X(d : \mathbb{D}) = \sum_{i \in I} \sum_{e_i : \mathbb{D}_i} a_i(f_i(d, e_i)) \cdot X_i(g_i(d, e_i)) \lhd\ h_i(d, e_i) \rhd \delta$$

where I is a finite index set, $\mathbb{D}, \mathbb{D}_i, \mathbb{D}_{a_i} \in \mathcal{D}$, $a_i \in \mathcal{A} \cup \{\tau\}$, $a_i : \mathbb{D}_{a_i}$, $f_i : \mathbb{D} \times \mathbb{D}_i \to \mathbb{D}_{a_i}$, $g_i : \mathbb{D} \times \mathbb{D}_i \to \mathbb{D}$ and $h_i : \mathbb{D} \times \mathbb{D}_i \to \mathbb{B}$.
Here, the different states of the process are represented by the data parameter $d : \mathbb{D}$. We note that actually types $\mathbb{D}$ and $\mathbb{D}_i$ may be Cartesian products of data types. The data parameter e_i can influence the parameter of action a_i, the condition h_i and the resulting state g_i. Parameter e_i is typically used to let a read action range over a data domain (i.e. choice quantification). In the future, when writing I^P, we refer to the index set of process P.

Furthermore, we define $en_I(X, d)$, with $en_I : \mathcal{P} \times \mathbb{D} \to 2^I$, to be the set of *enabled* indices of process $X(d)$. It is defined as follows: $en_I(X, d) = \{i \in I \mid \exists e.h_i(d, e)\}$. Note that from $en_I(X, d)$, we can derive which actions a_i are enabled, and which terms are reachable from $X(d)$. The set of enabled actions of $X(d)$ is referred to as $en_{\mathcal{A}}(X, d)$. An enabled transition is a triple $(X(d), a_i(e), X_i(d'))$ with $e : \mathbb{D}_{a_i}$, which means that from process term $X(d)$ an enabled action $a_i(e)$ can be fired, leading to a process term $X_i(d')$. We denote such a transition as $X(d) \xrightarrow{a_i(e)} X_i(d')$. The set of enabled transitions of $X(d)$ is referred to as $en(X, d)$. Of course, $en(\checkmark) = \emptyset$ and $en(\delta) = \emptyset$.

Initialisation line $\mathcal{I}$ defines the initial situation of a μCRL specification. It is in general of the following form:[1]

$$\partial_H(X_0(d_0) || \ldots || X_m(d_m))$$

Here $X_0, \ldots, X_m \in \mathcal{P}$ and $\forall 0 \leq j \leq m . X_j : \mathbb{D}_j \wedge d_j \in \mathbb{D}_j$. Note, that if $m = 0$, there is only a single process initialised. The following operators are used here:

1. The parallel composition operator ($||$). A process $P||Q$ executes P and Q concurrently in an interleaved fashion, i.e. the actions of P and Q are executed in arbitrary order. For all actions a, b, c,

[1] We omit the abstraction operator since we do not use it in this paper.

such that $(a, b, c) \in \mathcal{C}$, if one process can execute a and the other one can execute b, then P and Q can communicate (i.e. $P||Q$ executes the communication action c). Related to this, we write $P||Q \xrightarrow{a} P'||Q$ iff $P \xrightarrow{a} P'$, $P||Q \xrightarrow{a} P||Q'$ iff $Q \xrightarrow{a} Q'$, and $P||Q \xrightarrow{a} P'||Q'$ iff $\exists b, c \in \mathcal{A}.P \xrightarrow{b} P' \wedge Q \xrightarrow{c} Q' \wedge (b, c, a) \in \mathcal{C}$.

2. The encapsulation operator (∂_H). In $\partial_H(P)$ all actions of P that occur in the set $H \subseteq \mathcal{A}$ are disabled. Typically this operator is used to enforce that certain actions synchronise. We have $en_{\mathcal{A}}(\partial_H(P)) = en_{\mathcal{A}}(P) \setminus H$.

3. The renaming operator (ρ_f), with $f : \mathcal{A} \to \mathcal{A}$, is suitable for reusing a process definition using different action names. The subscript f signifies that the action a must be renamed to $f(a)$. The construction $\rho_f(P)$ behaves as P with its action names renamed according to f (after elimination of $||$ and ∂_H).

3 Modelling Time with μCRL

3.1 The Concepts

First in [6, 14], a form of discrete relative timing was modelled with μCRL. In short it works like this: An action *tick* is used to represent the end of a time slice and the beginning of a new one, i.e. to model a time transition. In order to share this notion of time, all running processes need to synchronise their *tick* actions. If at least one of these processes is busy and therefore unable to perform a *tick*, the *tick* action will not take place. In our notation, $(P_0||\ldots||P_m \xrightarrow{tick} \Leftrightarrow \forall 0 \leq i \leq m.tick \in en_{\mathcal{A}}(P_i))$. This synchronisation aspect is essential if one wants to use global timing.

Timing can be either *absolute* or *relative* and the time scale can be either *continuous* or *discrete* [2]. When using absolute timing, all actions are equipped with a time stamp, indicating when the action has to be fired (e.g. 12:05 PM). In relative timing the time of an action is expressed relative to the execution of a previous action. Here, time stamps are not needed; instead, we can choose for the so-called *two-phase* model [17], where a system seperately fires action and time transitions. Note that, in the approach just explained, relative timing is modelled in a two-phase manner, since the time unit, in which an action can be fired, is expressed relative to the time unit of actions fired earlier, using the special *tick* action for time transitions. On a discrete time scale, time is divided in finite time units. In a model using such a scale, a jump in time cannot be smaller than a single time unit. Note that, using the technique from [6, 14], we get discrete time in μCRL.

Later, in [22], this approach was extended with the usage of a second time action, called *tick2*. It was used to make actions *delayable*, which we will describe in more detail further on. In this paper, the time action *tock* takes over this role, but is also used to perform *partial delays*, i.e. parts of specified delays, as a major extension introduced in this paper is to allow time jumps of more than one time unit to be performed in a single transition. This is modelled by parameterising *tick* with a delay duration.

For timed process algebras, usually a number of time properties hold [1, 2, 3, 17, 20]. Which hold and which do not differs from one process algebra to another. By only adding time actions to μCRL, we do not achieve all necessary properties. Because of this, we introduce an extension of μCRL, called μCRL^{tick}. The idea is that in the end, a μCRL^{tick} specification can be transformed into a μCRL specification.

In [17], a list of main points in which timing mechanisms can differ from one another is provided. We list these properties here and explain our choices. The choices depend both on whether the properties would be achievable through our approach or not, and whether it makes sense in relation to existing literature. By doing so, this section sketches the main setting of subsequent sections.

- **Time determinism.** The progress of time should be deterministic. This property is essential [17]. It has the biggest influence on alternative composition; if two alternatives can delay, then they delay together. A further distinction can be made between *strong choice* and *weak choice*. In strong choice an undelayable alternative prevents all delays in the alternative composition, in weak choice it does not and the passage of time can therefore result in making a choice. We choose weak choice for our mechanism, since it more naturally fits in our approach. In section 7 we return to this decision.

- **Time additivity.** If a process can delay $d + d'$ time units, then it can delay for d and then for d' time units. The behaviour of these two cases is the same. This property very often holds in a timing mechanism, but not in ours. Since we achieve time steps through regular action steps, as we will see later, two delays in sequence result in at least two steps, while one delay might be done in one step. It may seem a serious lack of our mechanism, but there are timed languages known to lack time

additivity in certain situations, such as χ_t [3], in the case where one uses the Δd construction for a delay of d time units. As a positive note, not having time additivity allows us to use standard bisimilarity for the comparison of systems.

- **Deadlock-freeness.** Time can always pass, even though nothing else can be done. This practically allows for livelocks instead of deadlocks. This property holds in our mechanism. Besides that, as in [1, 2], we are able to introduce an undelayable deadlock (see section 7).

- **Action urgency.** Often referred to as *maximal progress*, it allows actions to have priority over the passage of time. In section 5 we explain our version of it, which is a mix of the ones found in [2, 3].

- **Persistency.** The passage of time cannot suppress the ability to perform an action. As in many timed process algebras, also in our case, this property does not hold, due to weak time determinism.

- **Finite variability and bounded variability.** Also known as *non-Zenoness*, i.e. in every time unit only a finite number of actions can occur. In [17], it is reported that only in the process algebra TCSP these properties hold. In our mechanism they do not.

- **Bounded Control.** There exists a time period d, such that the enabled set of actions of a process only changes in time, if the delay is bigger than or equal to d. In a discrete time domain this automatically holds, therefore also in our setting (take $d = 1$).

In the next section, a transformation procedure is presented, which transforms a μCRL^{tick} specification $\mathcal{M}_\mathsf{T}$ to a μCRL specification $\mathcal{M}$, in which practically the chosen set of timed properties is achieved.

3.2 The Axioms and Transition Rules

We will present some axioms and transition rules, which we would like to hold in our timed setting in order to achieve the mechanism chosen in the previous section. In appendix A, we prove that they indeed hold in our setting. In Tables 1 and 2, the additional axioms and transition rules are listed for the special actions *tick*, *tock*, and *ring*, which make discrete relative timing possible in our setting, the latter being an action which indicates that at least one delay is finishing. Actually, in μCRL^{tick}, only *tick* is available as an action when modelling, the other two only appear in resulting transitions. The *tick* action is used to model delays (rules 1, 2, 6 to 8, 16, 17, 19), *tock* indicates the delayability of actions and partial delays (rules 1, 3 to 5, 7 to 9, 16 to 19), and *ring* represents the finishing of at least one delay (rules 6, 10 to 15). Besides that, we identify sets $\mathcal{A}_U$ and $\mathcal{A}_D$ in relation to the action set $\mathcal{A}_\mathsf{T}$ of a μCRL^{tick} specification, with $\mathcal{A}_\mathsf{T} \cup \{\tau, \delta\} = \mathcal{A}_U \cup \mathcal{A}_D \cup \{tick\}$, $tick \notin \mathcal{A}_U \cup \mathcal{A}_D$, $\mathcal{A}_U \cap \mathcal{A}_D = \emptyset$, $\tau \in \mathcal{A}_U$ and $\delta \in \mathcal{A}_D$. Now, $\mathcal{A}_U$ constitutes the *urgent* or *undelayable* actions, while $\mathcal{A}_D$ contains all *delayable* actions. An enabled urgent action is an action, which will be disabled once a delay

Table 1. Extra axioms of μCRL^{tick}

$\forall n < 0.tick(\mathtt{n}) = \delta$	DRT1
$tick(\mathtt{n}) \cdot x + tick(\mathtt{n}) \cdot y = tick(\mathtt{n}) \cdot (x + y)$	DRT2

Table 2. Extra transition rules of μCRL^{tick}

1 $\dfrac{m, n > 0}{tick(m+n) \xrightarrow{tock(n)} tick(m)}$ 2 $\dfrac{m > 0}{tick(m) \xrightarrow{tick(m)} tick(0)}$

3 $\dfrac{m > 0 \quad a \in \mathcal{A}_D}{a \xrightarrow{tock(m)} a}$ 7 $\dfrac{x \xrightarrow{tick(\mathtt{m})} x' \quad y \xrightarrow{tick(\mathtt{m}) \vee tock(\mathtt{m})} y'}{x + y \xrightarrow{tick(\mathtt{m})} x' + y'}$

4 $\dfrac{m > 0 \quad a \in \mathcal{A}_U}{a \xrightarrow{tock(m)} \delta}$ 8 $\dfrac{x \xrightarrow{tick(\mathtt{m}) \vee tock(\mathtt{m})} x' \quad y \xrightarrow{tick(\mathtt{m})} y'}{x + y \xrightarrow{tick(\mathtt{m})} x' + y'}$

5 $\dfrac{m > 0}{\checkmark \xrightarrow{tock(m)} \checkmark}$ 9 $\dfrac{x \xrightarrow{tock(\mathtt{m})} x' \quad y \xrightarrow{tock(\mathtt{m})} y'}{x + y \xrightarrow{tock(\mathtt{m})} x' + y'}$

6 $\dfrac{}{tick(0) \xrightarrow{ring} \checkmark}$ 10 $\dfrac{x \xrightarrow{ring} x' \quad y \xrightarrow{ring} y'}{x + y \xrightarrow{ring} x' + y'}$

11 $\dfrac{x \xrightarrow{ring} x' \quad y \not\xrightarrow{ring}}{x + y \xrightarrow{ring} x'}$ 12 $\dfrac{x \not\xrightarrow{ring} \quad y \xrightarrow{ring} y'}{x + y \xrightarrow{ring} y'}$

13 $\dfrac{x \xrightarrow{ring} x'}{x || y \xrightarrow{ring} x' || y}$ 14 $\dfrac{y \xrightarrow{ring} y'}{x || y \xrightarrow{ring} x || y'}$

15 $\dfrac{x \xrightarrow{ring} x'}{x \cdot y \xrightarrow{ring} x' \cdot y}$ 16 $\dfrac{x \xrightarrow{tick(\mathtt{m})} x' \quad y \xrightarrow{tick(\mathtt{m}) \vee tock(\mathtt{m})} y'}{x || y \xrightarrow{tick(\mathtt{m})} x' || y'}$

17 $\dfrac{x \xrightarrow{tick(\mathtt{m}) \vee tock(\mathtt{m})} x' \quad y \xrightarrow{tick(\mathtt{m})} y'}{x || y \xrightarrow{tick(\mathtt{m})} x' || y'}$

18 $\dfrac{x \xrightarrow{tock(\mathtt{m})} x' \quad y \xrightarrow{tock(\mathtt{m})} y'}{x || y \xrightarrow{tock(\mathtt{m})} x' || y'}$

19 $\dfrac{a \in \{tick, tock\} \quad x \xrightarrow{a(\mathtt{m})} x'}{x \cdot y \xrightarrow{a(\mathtt{m})} x' \cdot y}$

is fired (rule 4). An enabled delayable action, on the other hand, is an action, which, in principle, can also be fired in a later time unit, i.e. can be postponed (rule 3). Note, that we consider τ to be urgent, and δ to be delayable (which is essential for the deadlock-freeness property).

We composed the tables by examining the rules of $\underline{\text{BPA}}^{drt}$-ID and ACP^{drt} [2], which are the Basic Process Algebra extended with discrete relative timing by means of a delay operator, and the Algebra of Communicating Processes extended with discrete relative timing, respectively, and comparing them with the existing, untimed axioms and transition rules of μCRL [13]. In Table 1, DRT1 says that a negative delay constitutes a deadlock. DRT2 is related to the previously mentioned time determinism principle, in that it says that the passage of time is deterministic. In Table 2, rules 1 and 2 reflect the fact that a delay can be performed (partially). Rules 3 and 4 say that delayable actions can delay and undelayable actions are disabled after a delay, respectively. Rules 5 and 6 indicate that at termination, a process can still delay , and a delay of length 0 terminates with a *ring* action. Rules 7, 8 and 9 ensure the time determinism principle. The additional transition rules for the *ring* action are stated in rules 10 to 15. Rules 16, 17 and 18 express time progress for parallel composition. Finally, in rule 19, the delayability of a sequential composition is explained.

These axioms and transition rules imply weak time determinism, no time additivity, no persistency, no finite or bounded variability and bounded control. As said before, that these axioms and transition rules actually indeed yield these properties for μCRL^{tick} is proven in appendix A.

4 Transforming a μCRL^{tick} Specification

4.1 Requirements and Approach

The basic idea of achieving discrete relative timing in this paper, is that a modeller can create a μCRL^{tick} specification $\mathcal{M}_\mathsf{T}$(the definition of which is like definition 1), which will then be transformed into a μCRL specification $\mathcal{M}$, in which the presented set of timing properties automatically holds. Though the goal is to require as little as possible of $\mathcal{M}_\mathsf{T}$, some requirements are inevitable.

4.1.1 The Input Specification

An input specification $\mathcal{M}_\mathsf{T} = (\mathcal{D}_\mathsf{T}, \mathcal{F}_\mathsf{T}, \mathcal{A}_\mathsf{T}, \mathcal{C}_\mathsf{T}, \mathcal{P}_\mathsf{T}, \mathcal{I}_\mathsf{T})$ can be transformed into a specification $\mathcal{M}$, if the following holds:

- Besides the usual domains $\mathbb{N}$ and $\mathbb{B}$, we have a time domain $\mathbb{T}$. Since we need a discrete and totally ordered time domain, and for practical reasons, negative values are needed, the structure of the domain can be a copy of $\mathbb{Z}$.
- Similarly, there should be functions using $\mathbb{T}$, based on the usual functions using $\mathbb{Z}$, in $\mathcal{F}$. Later on, we assume the presence of an *if-then-else* construct, written as $b \rightarrow x, y$, with $b : \mathbb{B}$ and $x, y : \mathbb{T}$, where $\mathrm{T} \rightarrow x, y = x$ and $\mathrm{F} \rightarrow x, y = y$.
- $tick \in \mathcal{A}_\mathsf{T}$ with $tick : \mathbb{T}$, which the modeller can use to model delays of t time units.
- *tick* is not involved in any rule of $\mathcal{C}$.
- In $\mathcal{P}_\mathsf{T}$, for all $X \in \mathcal{P}_\mathsf{T}$ there are no enumerations over $\mathbb{T}$. Why this would cause problems will be explained later.

4.1.2 The Transformation

In this section, we describe how $\mathcal{M}_\mathsf{T}$, which meets the given requirements, can be transformed into a specification $\mathcal{M}$. The majority of the work, of course, is performed on $\mathcal{P}_\mathsf{T}$.

We create a specification $\mathcal{M} = (\mathcal{D}, \mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{P}, \mathcal{I})$, by first obtaining the following, given specification $\mathcal{M}_\mathsf{T}$:

- $\mathcal{D} = \mathcal{D}_\mathsf{T}$.
- $\mathcal{F} = \mathcal{F}_\mathsf{T} \cup \{\theta : (\mathbb{B} \times \mathbb{N})^u \rightarrow \mathbb{T}\}$. The function θ is given a vector $\overrightarrow{b_i, n_i}$ and returns the smallest n_i for which b_i evaluates to Tand $n_i > 0$. The upperbound u on the size of the vector should be chosen sufficiently high, as it will imply that not more than u syntactical occurrences of *tick* are allowed in each process. For practical reasons, we fix a sufficiently large constant c (it should be larger than the largest single delay step in $\mathcal{M}_\mathsf{T}$) and say, that $\theta(\overrightarrow{b_i, n_i}) = \mathsf{c}$ iff for all (b_i, n_i), $b_i = \mathrm{F} \vee n_i \leq 0$ evaluates to Tor the vector size is 0. In the remainder of this paper, we write θ when $u = 0$, and m (for any $m \in \mathbb{T}$) whenever it is clear that $\theta(\overrightarrow{b_i, n_i}) = m$.
- $\mathcal{A} = \mathcal{A}_\mathsf{T} \cup \{ring, tock, tock', tick'\}$, where the actions $tock'$ and $tick'$ are used for intermediate results of communication. For more on this, see the following description of $\mathcal{C}$ and $\mathcal{I}$.
- $\mathcal{C} = \mathcal{C}_\mathsf{T} \cup \{(tick, tock, tick'), (tock, tick, tick'), (tick, tick, tick'), (tock, tock, tock')\}$.

Next, we describe how to obtain $\mathcal{I}$. This is done by changing $\mathcal{I}_{\mathsf{T}}$ to the following:

$$\partial_{H\cup\{tock\}}(TX_0(d_0)|^{\mathsf{T}}|\dots|^{\mathsf{T}}|TX_m(d_m))$$

Here, the $TX_i \in \mathcal{P}$ are translations of the corresponding $X_i \in \mathcal{P}_{\mathsf{T}}$ (we return to this later), and the special operator $|^{\mathsf{T}}|$ is defined using existing operators:

$$P|^{\mathsf{T}}|Q \triangleq \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(P||Q))$$

This definition is based on the one given for $|\{\mathtt{tick}\}|$ in [6, 14], except now *tock* actions are taken into account. First, in $P||Q$, all *tick* and *tock* actions are forced to communicate. As can be seen in the rules of $\mathcal{C}$, if at least one *tick* action is involved in communication, the result is $tick'$. After that, all resulting $tick'$ and $tock'$ are renamed to *tick* and *tock*. Note, that $|^{\mathsf{T}}|$ is associative and commutative. It should be stressed, that the final encapsulation of *tock* actions in $\mathcal{I}$ ensures that the system as a whole will only perform a delay if at least one process performs a complete delay, i.e. a *tick*-step.

Finally, we explain how to obtain $\mathcal{P}$. Each $X \in \mathcal{P}_{\mathsf{T}}$ is of the LPE form as described in section 2.

We divide I as follows: $I = I_U \cup I_D \cup I_C$, where $i \in I_U$ iff $a_i \in \mathcal{A}_U$, $i \in I_D$ iff $a_i \in \mathcal{A}_D$ and $i \in I_C$ iff $a_i = tick$.

Now, for each $X \in \mathcal{P}_{\mathsf{T}}$ in $\mathcal{I}_{\mathsf{T}}$, we create $TX \in \mathcal{P}$. The vector $\overrightarrow{x_{i_c}}$ consists of all x_i's for which $i \in I_C$ (similar definitions apply for $\overrightarrow{f_{i_c}(d,e_{i_c})}$ and $\overrightarrow{h_{i_c}(d,e_{i_c})}$). For clarity reasons, we write $\boldsymbol{f_i}, \boldsymbol{g_i}$ and $\boldsymbol{h_i}$ for $f_i(d,e_i), g_i(d,e_i)$ and $h_i(d,e_i)$, respectively.

$$\begin{aligned}
&TX(d:\mathbb{D}) = \\
&\quad \sum_{i\in I\setminus I_C}\sum_{e_i:\mathbb{D}_i} a_i(\boldsymbol{f_i})\cdot TX_i(\boldsymbol{g_i}) \lhd\ \boldsymbol{h_i} \rhd \delta + \\
&\quad ring\cdot T\hat{X}(d) \lhd\ \mathrm{F} \vee \bigvee_{i\in I_C} (\boldsymbol{f_i} = 0 \wedge \boldsymbol{h_i}) \rhd \delta + \\
&\quad tick(\theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}}))\cdot TX'(d, \overrightarrow{\boldsymbol{h_{i_c}} \to \boldsymbol{f_{i_c}} - \theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}}), \boldsymbol{f_{i_c}}}) \\
&\qquad \lhd\ \theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}}) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(d, \overrightarrow{\boldsymbol{h_{i_c}} \to \boldsymbol{f_{i_c}} - t, \boldsymbol{f_{i_c}}}) \\
&\qquad \lhd\ 0 < t < \theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}}) \rhd \delta
\end{aligned}$$

At this point, an explanation is in order. In the first line of $TX(d:\mathbb{D})$, essentially all the 'lines' of $X(d:\mathbb{D})$ dealing with actions other than *tick*, are adopted without change. The second line contains a *ring* action, which is fired whenever at least one delay completely finishes. Note that the guard checks whether there are any *tick* actions in X enabled with a parameter equal to 0. This effectively transforms possible occurrences of delays of length 0 in X to *ring*. This is similar to χ_t [3], where finished delay actions are followed by τ, except that here, *ring* can represent the finishing of multiple delays simultaneously. Having fired a *ring* action, the process $T\hat{X}$ is called, which will be described next. In the third line, a single *tick* alternative is added to the process, which has $\theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}})$ as its argument, in other words, the minimal non-zero argument of all $tick \in en_{\mathcal{A}}(X)$. After that, TX' is called, the definition of which will be presented later. Suffice it to say at this point, that, besides $d:\mathbb{D}$, it is equipped with arguments of type $\mathbb{T}$, each getting the initial value $\boldsymbol{f_{i_c}} - \theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}})$, if $\boldsymbol{h_{i_c}}$ holds, and $\boldsymbol{f_{i_c}}$, if not. The intuition is, that these arguments will be used to model timers, used for each *tick* appearing in X. If the *tick* in TX is fired, all 'enabled' timers (i.e. timers corresponding to enabled *tick* actions) must be decreased by the right amount. Finally, the fourth line contains a number of *tock* alternatives, ranging from $tock(1)$ to $tock(\theta(\overrightarrow{\boldsymbol{h_{i_c}},\boldsymbol{f_{i_c}}})-1)$. These alternatives allow partial delays to happen, and make delayable actions delayable. When one of these is fired, the enabled timers are updated accordingly.

Process $T\hat{X}$ is the transformation of $\hat{X}$, which is as follows, where e.g. a_j^i indicates a_j originating from X_i:

$$\begin{aligned}
\hat{X}(d:\mathbb{D}) = \sum_{i\in I_C}\sum_{j\in I^{X_i}}\sum_{e_j^i:\mathbb{D}_j^i} & a_j^i(f_j^i(\boldsymbol{g_i},e_j^i))\cdot X_j^i(g_j^i(\boldsymbol{g_i},e_j^i)) \\
& \lhd\ h_j^i(\boldsymbol{g_i},e_j^i) \wedge \boldsymbol{f_i} = 0 \wedge \boldsymbol{h_i} \rhd \delta
\end{aligned}$$

In words, $\hat{X}$ consists of the alternative composition of all actions from processes X_i with $i \in I_C$. Furthermore, the conditions ensure, that only actions following finished delays can be enabled. This process is transformed into $T\hat{X}$, in which the timing works correctly again.

The process TX' is also derived from X. It is of the following form:

$$\begin{aligned}
&TX'(d:\mathbb{D}, \overrightarrow{ct_{i_c}:\mathbb{T}}) = \\
&\quad \sum_{i\in I_D}\sum_{e_i:\mathbb{D}_i} a_i(\boldsymbol{f_i})\cdot TX_i(\boldsymbol{g_i}) \lhd\ \boldsymbol{h_i} \rhd \delta + \\
&\quad ring\cdot T\hat{X}'(d, \overrightarrow{ct_{i_c}}) \lhd\ \mathrm{F} \vee \bigvee_{i\in I_C} (ct_i = 0 \wedge \boldsymbol{h_i}) \rhd \delta + \\
&\quad tick(\theta(\overrightarrow{\boldsymbol{h_{i_c}}, ct_{i_c}}))\cdot TX'(d, \overrightarrow{\boldsymbol{h_{i_c}} \to ct_{i_c} - \theta(\overrightarrow{\boldsymbol{h_{i_c}}, ct_{i_c}}), ct_{i_c}}) \\
&\qquad \lhd\ \theta(\overrightarrow{\boldsymbol{h_{i_c}}, ct_{i_c}}) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(d, \overrightarrow{\boldsymbol{h_{i_c}} \to ct_{i_c} - t, ct_{i_c}}) \\
&\qquad \lhd\ 0 < t < \theta(\overrightarrow{\boldsymbol{h_{i_c}}, ct_{i_c}}) \rhd \delta
\end{aligned}$$

In the first line of TX', all delayable actions of X appear unchanged. In the second line, as in TX, a *ring* action is placed, to indicate the finishing of delays. Note that here, it is checked whether the corresponding timers ct_i have expired. In the third line,

as in TX, a *tick* alternative is offered, but also here, instead of working with the f_i's, the current values of the timers are used. Finally, in the fourth line, the *tock* alternatives are displayed, also working with the timers instead of the f_i's.

Finally, we have process $T\hat{X}'$, which is the transformation of $\hat{X}'$ (and similar to $T\hat{X}$):

$$\hat{X}'(d:\mathbb{D}, \overrightarrow{ct_{i_c}:\mathbb{T}}) = \sum_{i\in I_C}\sum_{j\in I^{X_i}}\sum_{e^i_j:\mathbb{D}^i_j} a^i_j(f^i_j(\boldsymbol{g_i}, e^i_j))\cdot X^i_j(g^i_j(\boldsymbol{g_i}, e^i_j)) \triangleleft h^i_j(\boldsymbol{g_i}, e^i_j) \wedge ct_i = 0 \wedge \boldsymbol{h_i} \triangleright \delta$$

In this manner, every encountered process X leads to $TX, T\hat{X}, TX', T\hat{X}'$. All the resulting processes together form $\mathcal{P}$. We also need to provide a translation for the special process $\checkmark$. In $\mathcal{P}$, we place a process $T\checkmark = \sum_{t\in\mathbb{T}} tock(t)\cdot T\checkmark \triangleleft 0 < t < \mathsf{c} \triangleright \delta$, which does nothing, except for allowing timesteps no bigger than c time units to pass.

At this point, we return to the earlier mentioned restriction, that enumerations over $\mathbb{T}$ are not allowed in $\mathcal{M}_\mathsf{T}$. One might question the necessity for a μCRL^{tick} process such as $X = \sum_{t\in\mathbb{T}} tick(t)$, since partial delays are already achieved in a different way. Note that the transformation cannot deal with it properly, since the θ-function cannot be applied.

5 Maximal Progress

Timed systems often use a concept called *maximal progress*. It allows actions to have priority over the passage of time [2, 17, 20]. The application of this technique differs between languages. In [2], for instance, it is defined as an operator, applicable on processes, giving actions from a given set H priority over time. In χ_t, on the other hand, a similar concept called *urgent communication* is always applied globally on a system, giving all actions priority. In our approach, we choose an intermediate form; maximal progress can be applied once, globally on a specification $\mathcal{M}_\mathsf{T}$. It is, however, not mandatory, and it applies for a given subset of actions. Next, we describe how we achieve this.

Achieving maximal progress through the tranformation of specifications turns out to be very complicated, due to the fact that it depends on the interaction between processes. However, we observe that globally applied maximal progress can be straightforwardly obtained by embedding it in state space generation. There, a system is already considered as a whole, and it turns out that maximal progress can be applied on a state by state basis. At first, this approach may seem unconventional, since traditionally, maximal progress is added to a timed language through an operator. We point out, however, that conceptually, maximal progress has a lot in common with partial order reduction [19], many forms of which could in fact also be achieved with an extra operator, but in that field, it is custom to embed it in state space generation.

Algorithm 1 Maximal progress breadth-first state space generation

Require: TX_0, d_0, $H \subseteq \mathcal{A} \setminus \{tick, tock, tock', tick'\}$
  $Open \leftarrow \{d_0\}$
  $Closed \leftarrow \emptyset$
  $Succ \leftarrow \emptyset$
  while $Open \setminus Closed \neq \emptyset$ **do**
    for all $d \in Open \setminus Closed$ **do**
      if $H \cap en_\mathcal{A}(TX_0, d) = \emptyset$ **then**
        $Succ \leftarrow Succ \cup \{d' \mid (TX_0(d), a(e), TX_0(d')) \in en(TX_0, d)\}$
      else
        $Succ \leftarrow Succ \cup \{d' \mid (TX_0(d), a(e), TX_0(d')) \in en(TX_0, d) \wedge a \neq tick\}$
      end if
    end for
    $Closed \leftarrow Closed \cup Open$
    $Open \leftarrow Succ$
    $Succ \leftarrow \emptyset$
  end while

Algorithm 1 shows a breadth-first search with maximal progress. It cuts away undesired parts of the state space, i.e. those which are reached through unnecessary delays. In practice, $ring \in H$ allows for nice constructions, such as time-out (see section 6). The connection between a specification $\mathcal{M}_\mathsf{T}$ and this algorithm is achieved through a so-called *linearisation* step [21], which maps $\mathcal{I}_\mathsf{T}$ (after transformation) to a single process $TX_0(d_0)$. The set of successors of process term $TX_0(d)$ is obtained with $en(TX_0, d)$.

6 Examples

In [14], a watchdog is presented as an example of using their timing mechanism. As the construction of a watchdog is very common in timed systems, this is a nice example to show the applicability of the mechanism. It should watch whether an assigned component works properly, using two channels to communicate. On the first channel, an *ok* message must be received from the component every m time units. The moment a time-out occurs, in other words, *ok* is not received within m time units, an *alarm* message is sent over channel 2. In [14], the μCRL specification for this is presented as follows, where *Timer* is their data type for timers, the initialisation line is started with *init*, and the meaning of the functions and actions used should

be clear from the context:

$$
\begin{aligned}
&A(t : Timer, m : \mathbb{N}) = \\
&\quad expire{\cdot}B(reset(t), m) \lhd expired(t) \rhd \delta + \\
&\quad tick{\cdot}A(t-1, m) \lhd \neg expired(t) \rhd \delta + \\
&\quad recv(ok){\cdot}A(set(t, m), m) \\
&B(t : Timer, m : \mathbb{N}) = send(alarm){\cdot}A(set(t, m), m) \\
&init\ A(on(5), 5)
\end{aligned}
$$

The same watchdog can be specified with μCRL^{tick} in a much more readable way. For $a \in \mathcal{A}_U$, we write $\underline{\underline{a}}$.

$$A = tick(5){\cdot}\underline{\underline{send(alarm)}}{\cdot}A + recv(ok){\cdot}A$$

This leads to the following μCRL processes (for readability purposes, we removed alternatives with conditions which do not hold, and filled in results of the θ-function directly, where possible):

$$
\begin{aligned}
&TA = recv(ok){\cdot}TA + \\
&\quad tick(5){\cdot}TA'(0) + \sum_{t \in \mathbb{T}} tock(t){\cdot}TA'(5-t) \lhd 0 < t < 5 \rhd \delta \\
&TA'(ct_0 : \mathbb{T}) = recv(ok){\cdot}TA + \\
&\quad ring{\cdot}T\hat{A}' \lhd ct_0 = 0 \rhd \delta + tick(ct_0){\cdot}TA'(0) \lhd ct_0 \neq 0 \rhd \delta + \\
&\quad \sum_{t \in \mathbb{T}} tock(t){\cdot}TA'(ct_0 - t) \lhd 0 < t < \theta(\mathrm{T}, ct_0) \rhd \delta \\
&T\hat{A}' = send(alarm){\cdot}TA + \sum_{t \in \mathbb{T}} tock(t){\cdot}T\hat{A}'' \lhd 0 < t < \mathtt{c} \rhd \delta \\
&T\hat{A}'' = T\checkmark
\end{aligned}
$$

Note that in order to make this work as desired, maximal progress needs to be applied for *ring* and for *recv* and *send*, or, when the watchdog is part of a bigger system, for the eventual communication action (e.g. say, that $(recv, send, com) \in \mathcal{C}_\mathsf{T}$). Then, *tick* alternatives to the actions are pruned away during state space generation. If, however, the component cannot send a message yet, due to delaying, the watchdog can wait by firing a *tick* or *tock* action, and subsequently keep track of the number of time units.

An old χ_t example is a dish washing cluster, as illustrated in Figure 1.

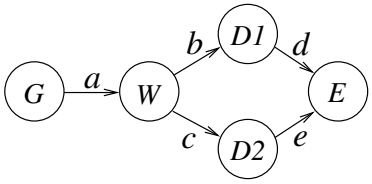

Figure 1. Dish washing cluster

A generator G supplies up to 14 plates, which are washed in sequence by W. Two driers $D1$ and $D2$ dry the plates concurrently, and finally, E removes the plates from the system. Next, we show how these processes are defined in μCRL^{tick}, where for each channel a, sa and ra represent sending and receiving over the channel, respectively, and $(sa, ra, ca) \in \mathcal{C}_\mathsf{T}$. It shows that the language is very suitable for specifying a system like this. The system leads to a state space of 940 states and 1,732 transitions without maximal progress, which can be generated in 0.9 seconds, and ... states and ... transitions including maximal progress, which shows the practicality of our approach. Maximal progress ensures that the specified delays are always performed in as few steps as possible, i.e. intervals in which no process can perform communication are jumped in a single transition.

$$
\begin{aligned}
\mathcal{P}_\mathsf{T} = \{&G(n : \mathbb{N}) = sa{\cdot}G(n+1) \lhd n < 14 \rhd \delta, \\
&W = ra{\cdot}tick(15){\cdot}(sb + sc){\cdot}W, \\
&D1 = rb{\cdot}tick(25){\cdot}sd{\cdot}D1, \\
&D2 = rc{\cdot}tick(25){\cdot}se{\cdot}D2, \\
&E = (rd + re){\cdot}E\}
\end{aligned}
$$

$$\mathcal{I}_\mathsf{T} = \partial_{\{sa,ra,sb,rb,sc,rc,sd,rd,se,re\}}(G(0)||W||D1||D2||E)$$

7 Conclusions and Extensions

We proposed an extension of μCRL, called μCRL^{tick}, which can practically be achieved, by transforming μCRL^{tick} specifications to μCRL specifications, such that the desired time properties still hold. We build on the work in earlier papers by emphasising ease of use and allowing time jumps of arbitrary length in a single transition. The toolset can directly be used for the verification of properties of μCRL^{tick} specifications, although an extra state space generation algorithm needs to be present to provide maximal progress. Such an algorithm can, however, be implemented straightforwardly.

The setting allows for several extensions, of which we name a few here. For instance, in [1, 2], a current time unit time-stop $\underline{\underline{\delta}}$ is mentioned, which, in contrast to the standard deadlock, does not allow the passage of time. This could be added to μCRL^{tick} by adding an extra action in $\mathcal{A}_\mathsf{T}$ and in the transformation ensure that no delays can be performed whenever this action, by then translated to deadlock, is enabled. The communication mechanism can be left as is. In [2] there is also mention of *relative time-out*, which places an upper-limit to the number of time units allowed to pass in the processes subjected to it. At least a system-wide version of it is achievable in our setting by including a process $R(c : \mathbb{T}) = \sum_{t \in \mathbb{T}} tock(t){\cdot}R(c-t) \lhd c > 0 \wedge 0 < t \leq c \rhd \delta$ in $\mathcal{P}$ (after transformation), and placing R with a given upper-limit in parallel with the system.

In this paper we chose a specific timing mechanism, most suitable for our transformation. There are, nevertheless, other mechanisms possible. Here, we briefly go into how other decisions would effect the transformation. For instance, instead of weak choice, we could

choose for strong choice. Then, it must be ensured in all processes that no delays are enabled whenever an urgent action is, which might result in very extensive conditions for the time actions. Another decision would be to interpret deadlock as a time-stop [1, 2] as opposed to a livelock. Then the deadlock relates to the time-stop as described earlier.

Future Work We want to implement the proposed transformation. We used it manually on a number of small cases, but want to use it on bigger ones in the future. In [14], an *LTL*-like temporal logic is introduced, which uses *tick* actions to encode time constraints, thereby allowing the usage of time in untimed temporal formulas. A similar approach can be taken for the μCRL^{tick} setting, but then also a transformation from timed formulas to untimed formulas should be provided, as for instance the property "action a is fired 3 time units after action b" in practice means, that any sequence of *tick* actions is allowed between a and b, as long as the sum of the delays is 3.

Acknowledgements We thank Wan Fokkink and the anonymous reviewers for their constructive comments.

References

[1] J.C.M. Baeten. Embedding untimed into timed process algebra: the case of explicit termination. *Math. Struct. in Comp. Science*, 13(4):589–618, 2003.

[2] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monograph. Springer, 2002.

[3] D.A. van Beek, K.L. Man, M.A. Reniers, J.E. Rooda, and R.R.H. Schiffelers. Syntax and Semantics of Timed Chi. CS-Report 05-09, Eindhoven University of Technology, 2005.

[4] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[5] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A Toolset for Analysing Algebraic Specifications. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 250–254, 2001.

[6] S.C.C. Blom, N. Ioustinova, and N. Sidorova. Timed verification with μCRL. In *Proc. PSI 2003*, volume 2890 of *LNCS*, pages 178–192, 2003.

[7] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP - a protocol validation and verification toolbox. In *Proc. CAV'96*, volume 1102 of *LNCS*, pages 437–440, 1996.

[8] W.J. Fokkink, J.F. Groote, J. Pang, B. Badban, and J.C. van de Pol. Verifying a Sliding Window Protocol in μCRL. In *Proc. AMAST 2004*, volume 3116 of *LNCS*, pages 148–163, 2004.

[9] W.J. Fokkink, J. Pang, and A.J. Wijs. Is Timed Branching Bisimilarity an Equivalence Indeed? In *Proc. FORMATS'05*, volume 3829 of *LNCS*, pages 258–272, 2005.

[10] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[11] J.F. Groote. The Syntax and Semantics of timed μCRL. Technical Report SEN-R9709, CWI, 1997.

[12] J.F. Groote, F. Monin, and J.C. van de Pol. Checking verifications of protocols and distributed systems by computer. In *Proc. CONCUR'98*, volume 1466 of *LNCS*, pages 629–655, 1998.

[13] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Proc. ACP'94*, Workshops in Computing Series, pages 26–62. Springer, 1995.

[14] N. Ioustinova. *Abstractions and Static Analysis for Verifying Reactive Systems*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[15] J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types*. Wiley-Teubner, Chichester, Stuttgart, 1996.

[16] S.P. Luttik. *Choice Quantification in Process Algebra*. PhD thesis, University of Amsterdam, 2002.

[17] X. Nicollin and J. Sifakis. An Overview and Synthesis on Timed Process Algebras. In *CAV'91*, volume 575 of *LNCS*, pages 376–398, 1991.

[18] S. Owre, J.M. Rushby, and N. Shankar. PVS: a Prototype Verification System. In *Proc. CADE'92*, volume 607 of *LNCS*, pages 748–752, 1992.

[19] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial Order Methods in Verification*, volume 29 of *DIMACS series in discrete mathematics and theoretical computer science*. AMS, 1996.

[20] I. Ulidowski and S. Yuen. Extending Process Languages with Time. In *Proc. AMAST'97*, volume 1349 of *LNCS*, pages 524–538, 1997.

[21] Y.S. Usenko. *Linearization in μCRL*. PhD thesis, Eindhoven University of Technology, 2002.

[22] A.J. Wijs and W.J. Fokkink. From $\chi_{\mathtt{t}}$ to μCRL: Combining Performance and Functional Analysis. In *Proc. ICECCS'05*, pages 184–193. IEEE Computer Society Press, 2005.

[23] A.G. Wouters. Manual for the μCRL tool set. Technical Report SEN-R0130, CWI, 2001.

[24] M.B. van der Zwaag. *Models and Logics for Process Algebra*. PhD thesis, University of Amsterdam, 2002.

A Properties of the Time Mechanism

In this section, we prove that the axioms and transition rules, as stated in Tables 1 and 2, are achieved for μCRL^{tick} specification $\mathcal{M}_{\mathsf{T}}$ after transformation to $\mathcal{M}$. We denote the transition rules as TR1,...,TR19. If $X = a(i)$, $Y(m) = a(m)$ and $m = i$, we say that $X = Y$. We start with an observation.

We begin by proving that an untimed system remains unchanged after transformation.

Proposition 1 (Semantics conservation [17]). *Given a μCRL^{tick} specification $\mathcal{M}_{\mathsf{T}}$ without delays, then $\mathcal{M}_{\mathsf{T}}$ behaves as its transformation $\mathcal{M}$.*

Proof. Follows from the form of the $TX \in \mathcal{P}$ and $\mathcal{I}$. Say that $\mathcal{M}_{\mathsf{T}}$ consists of a number of processes without delays. For each process P, in the transformed process TP, all actions, with their corresponding parameters, conditions and process calls are copied from P. Since $I_C^P = \emptyset$, furthermore, all other lines in TP are disabled, except for the *tock* alternatives. Since this holds for all processes in $\mathcal{M}_{\mathsf{T}}$, communications of time actions can only result in *tock* actions, which are in the end encapsulated in $\mathcal{I}$. □

Proposition 2 (DRT1). *Given a μCRL^{tick} process $X = tick(\mathtt{n})\cdot\checkmark$, with $\mathtt{n} < \mathtt{0}$. Then TX behaves as $T\delta$.*

Proof. If we transform X to TX, we get $TX = ring\cdot T\hat{X} \lhd n = 0 \rhd \delta + tick(\theta(\mathrm{T},n))\cdot TX'(\mathrm{T} \to n-t, n) \lhd \theta(\mathrm{T},n) \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(\mathrm{T} \to n-t, n) \lhd 0 < t < \theta(\mathrm{T},n) \rhd \delta$. Since $n \neq 0$, the *ring* option is disabled. Furthermore, since $\theta(\mathrm{T},n) = \mathtt{c}$, the *tick* option is also disabled. So, practically, $TX = \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(n-t) \lhd 0 < t < \mathtt{c} \rhd \delta$. Similarly, since $I_D^X = \emptyset$, we practically have $TX' = \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(n-t) \lhd 0 < t < \mathtt{c} \rhd \delta$. Clearly, we have $T\delta = \delta\cdot\checkmark + \sum_{t\in\mathbb{T}} tock(t)\cdot T\delta' \lhd 0 < t < \mathtt{c} \rhd \delta$, and $T\delta' = \delta\cdot\checkmark + \sum_{t\in\mathbb{T}} tock(t)\cdot T\delta' \lhd 0 < t < \mathtt{c} \rhd \delta$. Since the parameter of TX' does not have any behavioural effect and δ is never an option to fire, these two systems behave in the same way. □

Proposition 3 (DRT2). *Given two μCRL^{tick} processes $Z_0 = tick(\mathtt{n})\cdot X + tick(\mathtt{n})\cdot Y$ and $Z_1 = tick(\mathtt{n})\cdot(X+Y)$. Then, TZ_0 behaves as TZ_1.*

Proof. The transformation leads to the following:

$$
\begin{aligned}
TZ_0 &= ring\cdot T\hat{Z}_0 \lhd n = 0 \rhd \delta + \\
&\quad tick(\theta(\mathrm{T},n))\cdot TZ'_0(n-\theta(\mathrm{T},n), n-\theta(\mathrm{T},n)) \lhd \theta(\mathrm{T},n) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TZ'_0(n-t, n-t) \lhd 0 < t < \theta(\mathrm{T},n) \rhd \delta \\
T\hat{Z}_0 &= T(X[n=0] + Y[n=0]) \\
TZ'_0(ct_0 : \mathbb{T}, ct_1 : \mathbb{T}) &= ring\cdot T\hat{Z}'_0(ct_0, ct_1) \\
&\qquad \lhd ct_0 = 0 \vee ct_1 = 0 \rhd \delta + \\
&\quad tick(\theta(\mathrm{T},ct_0,\mathrm{T},ct_1))\cdot TZ'_0(ct_0 - \theta(\mathrm{T},ct_0,\mathrm{T},ct_1), \\
&\qquad ct_1 - \theta(\mathrm{T},ct_0,\mathrm{T},ct_1)) \lhd \theta(\mathrm{T},ct_0,\mathrm{T},ct_1) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TZ'_0(ct_0 - t, ct_1 - t) \\
&\qquad \lhd 0 < t < \theta(\mathrm{T},ct_0,\mathrm{T},ct_1) \rhd \delta \\
T\hat{Z}'_0(ct_0 : \mathbb{T}, ct_1 : \mathbb{T}) &= T(X[ct_0=0] + Y[ct_1=0]) \\
TZ_1 &= ring\cdot T\hat{Z}_1 \lhd n = 0 \rhd \delta + \\
&\quad tick(\theta(\mathrm{T},n))\cdot TZ'_1(n-\theta(\mathrm{T},n)) \lhd \theta(\mathrm{T},n) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TZ'_1(n-t) \lhd 0 < t < \theta(\mathrm{T},n) \rhd \delta \\
T\hat{Z}_1 &= T(X[n=0] + Y[n=0]) \\
TZ'_1(ct_0 : \mathbb{T}) &= ring\cdot T\hat{Z}'_1(ct_0) \lhd ct_0 = 0 \rhd \delta + \\
&\quad tick(\theta(\mathrm{T},ct_0))\cdot TZ'_1(ct_0 - \theta(\mathrm{T},ct_0)) \lhd ct_0 \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TZ'_1(ct_0 - t) \lhd 0 < t < ct_0 \rhd \delta \\
T\hat{Z}'_1(ct_0 : \mathbb{T}) &= T(X[ct_0=0] + Y[ct_0=0])
\end{aligned}
$$

Here, $P[b]$, with $b : \mathbb{B}$, refers to the process P, with all the conditions of its alternatives extended with '$\wedge b$'. First of all, clearly $T\hat{Z}_0 = T\hat{Z}_1$. Furthermore, $TZ_0 = TZ_1, TZ'_0 = TZ'_1$, and $T\hat{Z}'_0 = T\hat{Z}'_1$, considering that we can observe, that the two delays are initially of equal length, and furthermore, at all times $ct_0 = ct_1$. □

Proposition 4 (TR1,TR2). *For $X = tick(\mathtt{m})\cdot\checkmark$ with $\mathtt{m} > \mathtt{0}$, $\exists TX_i.TX \xrightarrow{tick(\mathtt{m})} TX_i$ with $TX_i = TY$, $Y = tick(0)$ and $\exists TX_j.TX \xrightarrow{tock(\mathtt{n})} TX_j$ with $\mathtt{0} < \mathtt{n} < \mathtt{m}$ and $TX_j = TZ$, $Z = tick(\mathtt{m}-\mathtt{n})$.*

Proof. Consider a process $X = tick(m)$ with $m > 0$. This transforms to $TX = ring\cdot T\hat{X} \lhd m = 0 \rhd \delta + tick(\theta(\mathrm{T},m))\cdot TX'(m - \theta(\mathrm{T},m)) \lhd \theta(\mathrm{T},m) \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(m-t) \lhd 0 < t < \theta(\mathrm{T},m) \rhd \delta$ with $TX'(ct_0 : \mathbb{T}) = ring\cdot T\hat{X}'(ct_0) \lhd ct_0 = 0 \rhd \delta + tick(\theta(\mathrm{T},ct_0))\cdot T\hat{X}'(ct_0 - \theta(\mathrm{T},ct_0)) \lhd \theta(\mathrm{T},ct_0) \neq \mathtt{c} \rhd$

$\delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(ct_0 - t) \lhd 0 < t < \theta(\mathrm{T}, ct_0) \rhd \delta$ and $T\hat{X}' = T\checkmark$. Since $\theta(\mathrm{T}, m) = m$, TX can fire $tick(m)$, leading to $TX'(m-m)$, which is clearly equal to TY. Another possibility for TX is to fire $tock(t)$, with $0 < t < m$, leading to process $TX'(m-t)$, which is clearly equal to TZ. □

Proposition 5 (**TR3**). *Given a μCRL^{tick} process $X = a\cdot\checkmark$, with $a \in \mathcal{A}_D$. Then $\exists TX_i.TX \xrightarrow{tock(\mathtt{m})} TX_i$, with $\mathtt{0} < \mathtt{m}$ and $TX_i = TX$.*

Proof. We have $TX = a\cdot\checkmark + tick(\theta)\cdot TX' \lhd \theta \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX' \lhd 0 < t < \theta \rhd \delta$. Since $\theta = \mathtt{c}$, $tick \notin en_{\mathcal{A}}(TX)$, but $tock \in en_{\mathcal{A}}(TX)$. We have $(TX, tock(t), TX') \in en(TX)$, with $0 < t < \mathtt{c}$, $\mathtt{c}$ being the reasonable upper-limit to the size of time steps. Furthermore, $TX' = a\cdot\checkmark + tick(\theta)\cdot TX' \lhd \theta \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX' \lhd 0 < t < \theta \rhd \delta$, so clearly $TX = TX'$. □

Proposition 6 (**TR4**). *Given a μCRL^{tick} process $X = a\cdot\checkmark$, with $a \in \mathcal{A}_U$. Then $\exists TX_i.TX \xrightarrow{tock(\mathtt{m})} TX_i$, with $\mathtt{0} < \mathtt{m}$ and $TX_i = T\delta$.*

Proof. We have $TX = a\cdot\checkmark + tick(\theta)\cdot TX' \lhd \theta \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX' \lhd 0 < t < \theta \rhd \delta$. Since $\theta = \mathtt{c}$, $tick \notin en_{\mathcal{A}}(TX)$, but $tock \in en_{\mathcal{A}}(TX)$. We have $(TX, tock(t), TX') \in en(TX)$, with $0 < t < \mathtt{c}$, $\mathtt{c}$ being the reasonable upper-limit to the size of time steps. Furthermore, $TX' = tick(\theta)\cdot TX' \lhd \theta \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX' \lhd 0 < t < \theta \rhd \delta$, which can only fire $tock$ steps. Clearly TX' behaves as $T\delta$ with $T\delta'$. □

Proposition 7 (**TR5**). *For μCRL^{tick} process $\checkmark$, we have $(T\checkmark, tock(m), T\checkmark) \in en(T\checkmark)$.*

Proof. We have $T\checkmark = \sum_{t\in\mathbb{T}} tock(t)\cdot T\checkmark \lhd 0 < t < \mathtt{c} \rhd \delta$. Clearly $(T\checkmark, tock(t), T\checkmark) \in en(T\checkmark)$, with $0 < t < \mathtt{c}$, $\mathtt{c}$ being the reasonable upper-limit to the size of time steps. □

Proposition 8 (**TR6**). *Given a μCRL^{tick} process $X = tick(\mathtt{0})\cdot\checkmark$. Then $\exists TX_i.TX \xrightarrow{ring} TX_i$, with $TX_i = T\checkmark$.*

Proof. We have $TX = ring\cdot T\hat{X} \lhd 0 = 0 \rhd \delta + tick(\theta(\mathrm{T}, 0))\cdot TX'(0 - \theta(\mathrm{T}, 0)) \lhd \theta(\mathrm{T}, 0) \neq \mathtt{c} \rhd \delta + \sum_{t\in\mathbb{T}} tock(t)\cdot TX'(0 - \theta(\mathrm{T}, 0)) \lhd 0 < t < \theta(\mathrm{T}, 0) \rhd \delta$. Since $\theta(\mathrm{T}, 0) = \mathtt{c}$, $tick \notin en_{\mathcal{A}}(TX)$, but $ring \in en_{\mathcal{A}}(TX)$. It follows that $T\hat{X} = T\checkmark$. □

Proposition 9 (**TR7,TR8**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{tick(\mathtt{m})} TP_i$ and $\exists TQ_j.TQ \xrightarrow{tick(\mathtt{m})\vee tock(\mathtt{m})} TQ_j$, then (1) $T(P + Q) \xrightarrow{tick(\mathtt{m})} T(P_i + Q_j)$ and (2) $T(Q + P) \xrightarrow{tick(\mathtt{m})} T(Q_j + P_i)$.*

Proof. We only prove (1) here. The proof for (2) is similar, due to the commutativity of $+$. First of all, let $R(d : \mathbb{D}) = P + Q$ be of the following form:

$$R(d:\mathbb{D}) = \sum_{i\in I^P} \sum_{e_i^P:\mathbb{D}_i^P} a_i^P(\boldsymbol{f_i^P})\cdot X_i^P(\boldsymbol{g_i^P}) \lhd \boldsymbol{h_i^P} \rhd \delta + \sum_{i\in I^Q} \sum_{e_i^Q:\mathbb{D}_i^Q} a_i^Q(\boldsymbol{f_i^Q})\cdot X_i^Q(\boldsymbol{g_i^Q}) \lhd \boldsymbol{h_i^Q} \rhd \delta$$

Now we have $TR(d : \mathbb{D})$ as follows:

$$\begin{aligned}
&TR(d:\mathbb{D}) = \\
&\quad \sum_{i\in I^P\setminus I_C^P} \sum_{e_i^P:\mathbb{D}_i^P} a_i^P(\boldsymbol{f_i^P})\cdot TX_i^P(\boldsymbol{g_i^P}) \lhd \boldsymbol{h_i^P} \rhd \delta + \\
&\quad \sum_{i\in I^Q\setminus I_C^Q} \sum_{e_i^Q:\mathbb{D}_i^Q} a_i^Q(\boldsymbol{f_i^Q})\cdot TX_i^Q(\boldsymbol{g_i^Q}) \lhd \boldsymbol{h_i^Q} \rhd \delta + \\
&\quad ring\cdot T\hat{R}(d) \lhd \mathrm{F} \vee \bigvee_{i\in I_C^P} (\boldsymbol{f_i^P} = 0 \wedge \boldsymbol{h_i^P}) \vee \\
&\qquad \bigvee_{i\in I_C^Q} (\boldsymbol{f_i^Q} = 0 \wedge \boldsymbol{h_i^Q}) \rhd \delta + \\
&\quad tick(\theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}))\cdot TR'(d, \\
&\qquad \overrightarrow{\boldsymbol{h}_{i_c^P} \to \boldsymbol{f}_{i_c^P} - \theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}), \boldsymbol{f}_{i_c^P}}, \\
&\qquad \overrightarrow{\boldsymbol{h}_{i_c^Q} \to \boldsymbol{f}_{i_c^Q} - \theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}), \boldsymbol{f}_{i_c^Q}}) \\
&\qquad \lhd \theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) \neq \mathtt{c} \rhd \delta + \\
&\quad \sum_{t\in\mathbb{T}} tock(t)\cdot TR'(d, \overrightarrow{\boldsymbol{h}_{i_c^P} \to \boldsymbol{f}_{i_c^P} - t, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q} \to \boldsymbol{f}_{i_c^Q} - t, \boldsymbol{f}_{i_c^Q}}) \\
&\qquad \lhd 0 < t < \theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) \rhd \delta
\end{aligned}$$

Since $\exists TP_i.TP \xrightarrow{tick(m)} TP_i$, by the form of TP, $\theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}) = m$ and $TP_i = TP'$. We can distinguish two cases:

1. $\exists TQ_j.TQ \xrightarrow{tick(\mathtt{m})} TQ_j$. Then, by the form of TQ, $\theta(\overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) = m$ and $TQ_j = TQ'$. By the definition of θ, $\theta(\overrightarrow{b_0, n_0}, \overrightarrow{b_1, n_1}) = min(\theta(\overrightarrow{b_0, n_0}), \theta(\overrightarrow{b_1, n_1}))$, therefore $\theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) = m$. So $TR(d) \xrightarrow{tick(m)} TR'(d, \ldots)$.
2. $\exists TQ_j.TQ \xrightarrow{tock(\mathtt{m})} TQ_j$. Then, by the form of TQ, $\theta(\overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) > m$ and $TQ_j = TQ'$. By the definition of θ, $\theta(\overrightarrow{b_0, n_0}, \overrightarrow{b_1, n_1}) = min(\theta(\overrightarrow{b_0, n_0}), \theta(\overrightarrow{b_1, n_1}))$, therefore $\theta(\overrightarrow{\boldsymbol{h}_{i_c^P}, \boldsymbol{f}_{i_c^P}}, \overrightarrow{\boldsymbol{h}_{i_c^Q}, \boldsymbol{f}_{i_c^Q}}) = m$. So $TR(d) \xrightarrow{tick(m)} TR'(d, \ldots)$.

Finally, we show that $TR'(d, \overrightarrow{ct_{i_c^P} : \mathbb{T}}, \overrightarrow{ct_{i_c^Q} : \mathbb{T}}) = T(P' + Q')$. We have $TR'(d, \overrightarrow{ct_{i_c^P} : \mathbb{T}}, \overrightarrow{ct_{i_c^Q} : \mathbb{T}}) = T(P + Q)'$. It follows directly that $T(P + Q)' = T(P' + Q')$. A similar argument holds for $T\hat{R}$ and $T\hat{R}'$. □

Proposition 10 (**TR9**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{tock(\mathtt{m})} TP_i$ and $\exists TQ_j.TQ \xrightarrow{tock(\mathtt{m})} TQ_j$, then $T(P+Q) \xrightarrow{tock(\mathtt{m})} T(P_i + Q_j)$.*

Proof. First of all, let $R(d:\mathbb{D}) = P+Q$ and $TR(d:\mathbb{D})$ be of the forms as presented in Proposition 9.

Since $\exists TP_i.TP \xrightarrow{tock(m)} TP_i$, by the form of TP, $\theta(\overrightarrow{\boldsymbol{h}_{\boldsymbol{i_c^P}}}, \overrightarrow{\boldsymbol{f}_{\boldsymbol{i_c^P}}}) > m$ and $TP_i = TP'$. Since $\exists TQ_j.TQ \xrightarrow{tock(\mathtt{m})} TQ_j$, by the form of TQ, $\theta(\overrightarrow{\boldsymbol{h}_{\boldsymbol{i_c^Q}}}, \overrightarrow{\boldsymbol{f}_{\boldsymbol{i_c^Q}}}) > m$ and $TQ_j = TQ'$. By the definition of θ, $\theta(\overrightarrow{b_0, n_0}, \overrightarrow{b_1, n_1}) = min(\theta(\overrightarrow{b_0, n_0}), \theta(\overrightarrow{b_1, n_1}))$, therefore $\theta(\overrightarrow{\boldsymbol{h}_{\boldsymbol{i_c^P}}}, \overrightarrow{\boldsymbol{f}_{\boldsymbol{i_c^P}}}, \overrightarrow{\boldsymbol{h}_{\boldsymbol{i_c^Q}}}, \overrightarrow{\boldsymbol{f}_{\boldsymbol{i_c^Q}}}) > m$. So $TR(d) \xrightarrow{tock(m)} TR'(d,\ldots)$.

Finally, we show that $TR'(d, \overrightarrow{ct_{i_c^P}:\mathbb{T}}, \overrightarrow{ct_{i_c^Q}:\mathbb{T}}) = T(P'+Q')$. We have $TR'(d, \overrightarrow{ct_{i_c^P}:\mathbb{T}}, \overrightarrow{ct_{i_c^Q}:\mathbb{T}}) = T(P+Q)'$. It follows directly that $T(P+Q)' = T(P'+Q')$. A similar argument holds for $T\hat{R}$ and $T\hat{R}'$. □

Proposition 11 (**TR10**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{ring} TP_i$ and $\exists TQ_j.TQ \xrightarrow{ring} TQ_j$, then $T(P+Q) \xrightarrow{ring} T(P_i + Q_j)$.*

Proof. First of all, let $R(d:\mathbb{D}) = P+Q$ and $TR(d:\mathbb{D})$ be of the forms as presented in Proposition 9.

Since $\exists TP_i.TP \xrightarrow{ring} TP_i$, by the form of TP, $\exists k \in I_C^P.\boldsymbol{f_k^P} = 0 \wedge \boldsymbol{h_k^P}$ and $TP_k = T\hat{P}$. Since $\exists TQ_j.TQ \xrightarrow{ring} TQ_j$, by the form of TQ, $\exists l \in I_C^Q.\boldsymbol{f_l^Q} = 0 \wedge \boldsymbol{h_l^Q}$ and $TQ_l = T\hat{Q}$. By the form of $TR(d)$, it follows that $TR(d) \xrightarrow{ring} T\hat{R}(d)$.

Finally, we show that $T\hat{R}(d) = T(\hat{P}+\hat{Q})$. We do that by comparing $\hat{R}(d)$ with $\hat{P}+\hat{Q}$. For $\hat{R}(d)$, we have:

$$\begin{aligned}\hat{R}(d:\mathbb{D}) = &\sum_{i\in I_C^P}\sum_{j\in I^{X_i^P}}\sum_{e_j^{i,P}:\mathbb{D}_j^{i,P}} a_j^{i,P}(f_j^{i,P}(\boldsymbol{g_i^P}, e_j^{i,P}))\cdot\\ &X_j^{i,P}(g_j^{i,P}(\boldsymbol{g_i^P}, e_j^{i,P})) \lhd\ h_j^{i,P}(\boldsymbol{g_i^P}, e_j^{i,P}) \wedge \boldsymbol{f_i^P} = 0 \wedge \boldsymbol{h_i^P} \rhd \delta +\\ &\sum_{i\in I_C^Q}\sum_{j\in I^{X_i^Q}}\sum_{e_j^{i,Q}:\mathbb{D}_j^{i,Q}} a_j^{i,Q}(f_j^{i,Q}(\boldsymbol{g_i^Q}, e_j^{i,Q}))\cdot\\ &X_j^{i,Q}(g_j^{i,Q}(\boldsymbol{g_i^Q}, e_j^{i,Q})) \lhd\ h_j^{i,Q}(\boldsymbol{g_i^Q}, e_j^{i,Q}) \wedge \boldsymbol{f_i^Q} = 0 \wedge \boldsymbol{h_i^Q} \rhd \delta\end{aligned}$$

Clearly, this is equal to $\hat{P}+\hat{Q}$.

□

Proposition 12 (**TR11,TR12**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{ring} TP_i$ and $TQ \not\xrightarrow{ring}$, then (1) $T(P+Q) \xrightarrow{ring} TP_i$ and (2) $T(Q+P) \xrightarrow{ring} TP_i$.*

Proof. We only prove (1) here. The proof for (2) is similar, due to the commutativity of $+$. First of all, let $R(d:\mathbb{D}) = P+Q$ and $TR(d:\mathbb{D})$ be of the forms as presented in Proposition 9.

Since $\exists TP_i.TP \xrightarrow{ring} TP_i$, by the form of TP, $\exists k \in I_C^P.\boldsymbol{f_k^P} = 0 \wedge \boldsymbol{h_k^P}$ and $TP_k = T\hat{P}$. Since $TQ \not\xrightarrow{ring}$, by the form of TQ, $\neg\exists l \in I_C^Q.\boldsymbol{f_l^Q} = 0 \wedge \boldsymbol{h_l^Q}$. By the form of $TR(d)$, it follows that $TR(d) \xrightarrow{ring} T\hat{R}(d)$.

Finally, we show that $T\hat{R}(d) = T\hat{P}$. We do that by comparing $\hat{R}(d)$ with $T\hat{P}$. $\hat{R}(d)$ is of the form as presented in Proposition 11. Since we know that $\neg\exists l \in I_C^Q.\boldsymbol{f_l^Q} = 0 \wedge \boldsymbol{h_l^Q}$, effectively, $TR(d) = \hat{P}$. □

Proposition 13 (**TR13,TR14,TR15**). *Given a μCRL^{tick} process P, such that $\exists TP_i.TP \xrightarrow{ring} TP_i$, then, for any other μCRL^{tick} process Q, $TP|^{\intercal}|TQ \xrightarrow{ring} TP_i|^{\intercal}|TQ$, $TQ|^{\intercal}|TP \xrightarrow{ring} TQ|^{\intercal}|TP_i$ and $TP{\cdot}TQ \xrightarrow{ring} TP_i{\cdot}TQ$.*

Proof. Follows from the transition rules for $||$ and $\cdot$ in μCRL, since in the transformed $\mathcal{I}_{\intercal}$, *ring* actions are not treated in a special way, i.e. are not encapsulated or renamed. □

Proposition 14 (**TR16,TR17**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{tick(\mathtt{m})} TP_i$ and $\exists TQ_j.TQ \xrightarrow{tick(\mathtt{m})\vee tock(\mathtt{m})} TQ_j$, then (1) $TP|^{\intercal}|TQ \xrightarrow{tick(\mathtt{m})} TP_i|^{\intercal}|TQ_j$ and (2) $TQ_j|^{\intercal}|TP_i \xrightarrow{tick(\mathtt{m})} TQ_j|^{\intercal}|TP_i$.*

Proof. We only prove (1) here. The proof for (2) is similar, due to the commutativity of $|^{\intercal}|$. First of all, we have $TP|^{\intercal}|TQ \triangleq \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP||TQ))$. We can distinguish two cases:

1. $\exists TQ_j.TQ \xrightarrow{tick(\mathtt{m})} TQ_j$. Since $(tick, tick, tick') \in \mathcal{C}$, $\partial_{\{tick,tock\}}(TP||TQ) \xrightarrow{tick'(m)} \partial_{\{tick,tock\}}(TP_i||TQ_j)$, therefore $\rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP||TQ)) \xrightarrow{tick(m)} \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP_i||TQ_j))$.

2. $\exists TQ_j.TQ \xrightarrow{tock(\mathtt{m})} TQ_j$. Since $(tick, tock, tick') \in \mathcal{C}$, $\partial_{\{tick,tock\}}(TP||TQ) \xrightarrow{tick'(m)} \partial_{\{tick,tock\}}(TP_i||TQ_j)$, therefore $\rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP||TQ)) \xrightarrow{tick(m)} \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP_i||TQ_j))$.

□

Proposition 15 (**TR18**). *Given two μCRL^{tick} processes P and Q, such that $\exists TP_i.TP \xrightarrow{tock(\mathtt{m})} TP_i$ and $\exists TQ_j.TQ \xrightarrow{tock(\mathtt{m})} TQ_j$, then $TP|^{\top}|TQ \xrightarrow{tock(\mathtt{m})} TP_i|^{\top}|TQ_j$.*

Proof. First of all, we have $TP|^{\top}|TQ \triangleq \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP||TQ))$. Since $(tock, tock, tock') \in \mathcal{C}$, $\partial_{\{tick,tock\}}(TP||TQ) \xrightarrow{tock'(m)} \partial_{\{tick,tock\}}(TP_i||TQ_j)$, therefore $\rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP||TQ)) \xrightarrow{tock(m)} \rho_{\{tick'\to tick, tock'\to tock\}}(\partial_{\{tick,tock\}}(TP_i||TQ_j))$. □

Proposition 16 (**TR19**). *Given a μCRL^{tick} process P, such that (1) $\exists TP_i.TP \xrightarrow{tick(\mathtt{m})} TP_i$, then for any μCRL^{tick} process Q, $TP{\cdot}TQ \xrightarrow{tick(\mathtt{m})} TP_i{\cdot}TQ$, and (2) $\exists TP_i.TP \xrightarrow{tock(\mathtt{m})} TP_i$, then for any μCRL^{tick} process Q, $TP{\cdot}TQ \xrightarrow{tock(\mathtt{m})} TP_i{\cdot}TQ$.*

Proof. Both cases follow directly from the form of TP. □