Forgetting the Time in Timed Process Algebra

Anton Wijs

Forgetting the Time in Timed Process Algebra
Timeless Behaviour in a Timestamped World

Anton Wijs

Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands
A.J.Wijs@tue.nl

Abstract. In this paper, we propose the notion of partial time abstraction for timed process algebras, which introduces the possibility to abstract away parts of the timing of system behaviour. Adding this notion leads to so-called partially timed process algebras and partially timed labelled transition systems. We describe these notions, and generalise timed branching bisimilarity to partially timed branching bisimilarity, allowing the comparison of systems with partial timing. Finally, with several examples and a case study, we demonstrate how partial time abstraction can be a useful modelling technique for timed models, which can lead to rigorous minimisations of state spaces.

1 Introduction

For many systems, correct and relevant verification involves timing aspects. In order to specify such systems, timed versions of existing modelling paradigms such as process algebras, automata, and Petri nets have been developed. However, timing aspects can be very demanding; when comparing two timed systems, they are only deemed equal in behaviour if both can perform the same actions at the same time. Checking timed temporal properties can often only practically be done if we partition system behaviour into time regions [1, 2]. In order to do so, [17] defined several *time-abstracting* bisimilarities, and [12] defined strong and weak *time-abstracted* equivalences. Little research has been done to consider *relaxation* of the timing aspects by *partially* removing timing from timed systems where it is not relevant for either the verification of a temporal property, or a comparison with another system.

Where time is concerned, modelling languages either incorporate it completely, or they do not incorporate it at all. If time is present in the language, it means that either all actions are supplied with a time stamp, or that there are additional timing actions, which implement some relative notion of time. In process algebras, the inclusion of timing has naturally led to timed labelled transition systems, and subsequently to timed bisimilarities. The overall experience in this field is that these notions are very complex; for instance, in [10], it turned out that the definition of a specific notion of timed branching bisimilarity was not an equivalence when considering an absolute continuous (or dense) time domain. This was fixed by changing the definition. One of the conclusions,

however, was that the new notion was very demanding of the state spaces to be compared. In practice, this may result in too few equalities of state spaces.

Another complication was raised in [16]. There, it is argued that the abstraction of parts of a system specification does not lead to a sufficient decrease of the size of the resulting state space. Action abstraction is a powerful tool in explicit model checking, which can be used to hide action labels which are not interesting considering the system property to check. The main advantage is that this hiding of actions, which results in so-called 'silent steps', often leads to a reduced state space, which can be analysed more practically. The solution offered in [16] is to use untimed, instead of timed, silent steps. Surely, this means that the results of action abstraction in untimed and timed process algebras are indistinguishable, and therefore timed state spaces can be reduced more effectively.

However, this introduces complications. E.g. in an absolute time setting, implicit deadlocks due to ill-timedness may be removed, thereby introducing erroneous traces. Reniers and Van Weerdenburg [16] recognise this, but accept it. Another observation is that in general, there is no reason to inseparably connect action abstraction with time abstraction.

In this paper, we work out the notion of partial time abstraction, independently from action abstraction for an absolute time setting. Our choice for this setting, however, does not imply that the proposed notions cannot be applied to relative timing. We present a partially timed branching (PTB) bisimulation relation, which is general enough to minimise both internal behaviour and time-hidden behaviour.[1] It should be stressed that, whereas we stick to explicit state model checking in this paper, the described notions can be adapted to a symbolic model checking setting. In Section 2, we start by describing a basic timed process algebra, timed labelled transition systems, and timed branching bisimilarity, after which we move to a more general setting, giving rise to the notions of a partially timed labelled transition system and PTB bisimilarity in Section 3. In this Section, we also define several time abstraction operators for a timed process algebra. We illustrate the use of the new notions with examples, and Section 4 presents a case study. Alternative approaches and other time settings are considered in Section 5. Finally, Section 6 discusses related work, and Section 7 contains the conclusions.

2 Preliminaries

Timed Labelled Transition Systems Let $\mathcal{A}$ be a non-empty set of visible actions, and τ a special action to represent internal events, with $\tau \notin \mathcal{A}$. We use $\mathcal{A}_\tau$ to denote $\mathcal{A} \cup \{\tau\}$. The time domain $\mathbb{T}$ is a totally ordered set with a least element

[1] Throughout the text, we use both the terms *untimed* and *time-hidden*, the difference between them being that an untimed action is literally not timed, while a time-hidden action is an action where the timing cannot be observed, but it is still there. When reasoning about time-hidden behaviour, though, we often state that a time-hidden action can in principle be fired at any time, since we do not know its timing.

0. We say that $\mathbb{T}$ is *discrete* if for each pair $u, v \in \mathbb{T}$ there are only finitely many $w \in \mathbb{T}$ such that $u < w < v$.

We use the notion of timed labelled transition systems from [19], in which labelled transitions are provided with a time stamp. A transition (s, ℓ, u, s') expresses that state s evolves into state s' by the execution of action ℓ at (absolute) time u. Such a transition is presented as $s \xrightarrow{\ell}_u s'$. It is assumed that execution of transitions does not consume any time. To keep the definition of timed labelled transition systems clean, consecutive transitions are allowed to have decreasing time stamps (i.e. ill-timedness); in the semantics, decreasing time stamps simply give rise to an (immediate) deadlock (see Definitions 2 and 4). To express time deadlocks, the predicate $\mathcal{U}(s, u)$ denotes that state s can let time pass until time u. A special state $\surd$ represents successful termination, expressed by the predicate $\surd\!\downarrow$. For the remainder of this paper, variables u, v, etc. range over $\mathbb{T}$.

Definition 1 (Timed labelled transition system). *A* timed labelled transition system *(*Tlts*) [15] is a triple $(\mathcal{S}, \mathcal{T}, \mathcal{U})$, where:*

1. *$\mathcal{S}$ is a set of states, including a state $\surd$ to represent successful termination;*
2. *$\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathbb{T} \times \mathcal{S}$ is a set of transitions;*
3. *$\mathcal{U} \subseteq \mathcal{S} \times \mathbb{T}$ is a* delay relation*, which satisfies:*
   - *$\mathcal{U}(s, 0)$;*
   - *For $u > 0$, $\mathcal{U}(s, u)$ iff there exist $v \geq u \in \mathbb{T}, \ell \in \mathcal{A}, s' \in \mathcal{S}$ such that $\mathcal{T}(s, \ell, v, s')$.*

Timed Branching Bisimulation Van Glabbeek and Weijland [11] introduced the notion of a *branching bisimulation* relation for untimed Ltss. Intuitively, a τ-transition $s \xrightarrow{\tau} s'$ is invisible if it does not lose possible behaviour (i.e., if s and s' can be related by a branching bisimulation relation). Van der Zwaag [19] defined a timed version of branching bisimulation, adding time stamps of transitions and ultimate delays $\mathcal{U}(s, u)$. Fokkink, Pang, and Wijs [10] showed that it was not an equivalence in a dense time domain, and improved the definition.

For $u \in \mathbb{T}$, the reflexive transitive closure of $\xrightarrow{\tau}_u$ is $\Rightarrow_u$. With $s \Rightarrow_u s'$ we express that s' is reachable from s by a number of τ-transitions at time u.

Definition 2 (Timed branching bisimulation). *Assume a* Tlts *$(\mathcal{S}, \mathcal{T}, \mathcal{U})$. A collection B of binary relations $B_u \subseteq \mathcal{S} \times \mathcal{S}$ for $u \in \mathbb{T}$ is a* timed branching bisimulation *[10] if $s\ B_u\ t$ implies:*

1. *if $s \xrightarrow{\ell}_u s'$, then*
   - *i. either $\ell = \tau$ and $s'\ B_u\ t$,*
   - *ii. or $t \Rightarrow_u \hat{t} \xrightarrow{\ell}_u t'$ with $s\ B_u\ \hat{t}$ and $s'\ B_u\ t'$;*
2. *if $t \xrightarrow{\ell}_u t'$, then vice versa;*
3. *if $s\!\downarrow$, then $t \Rightarrow_u t'\!\downarrow$ with $s\ B_u\ t'$;*
4. *if $t\!\downarrow$, then vice versa;*
5. *if $u \leq v$ and $\mathcal{U}(s, v)$, then for some $n \geq 0$ there are $t_0, \ldots, t_n \in \mathcal{S}$ with $t = t_0$ and $\mathcal{U}(t_n, v)$, and $u_0 < \cdots < u_n \in \mathbb{T}$ with $u = u_0$ and $v = u_n$, such that for $i < n$, $t_i \Rightarrow_{u_i} t_{i+1}$ and $s\ B_w\ t_{i+1}$ for $u_i \leq w \leq u_{i+1}$;*

6. *if* $u \leq v$ *and* $\mathcal{U}(t, v)$*, then vice versa;*

Two states s *and* t *are* timed branching bisimilar at u*, i.e.* $s \leftrightarrows^u_{tb} t$*, if there is a timed branching bisimulation* B *with* $s\ B_u\ t$*. States* s *and* t *are* timed branching bisimilar*, i.e.* $s \leftrightarrows_{tb} t$*, if they are timed branching bisimilar at all* $u \in \mathbb{T}$*.*

Transitions can be executed at the same time consecutively. By case 1 (and 2) in Definition 2, the behaviour of a state at some point in time is treated like untimed behaviour. Case 3 (and 4) deals with successful termination. By case 5 (and 6), time passing in a state s is matched by a related state t with a "τ-path" where all intermediate states are related to s at all times during the delay.[2] For a more detailed treatment of timed branching bisimilarity, see [10].

A Basic Process Algebra In the following, x, y are variables and s, t are process terms or $\surd$, with $\surd$ a special state representing successful termination.

We present a basic process algebra with alternative ('+'), sequential ('$\cdot$') and parallel ('$||$') composition, and action abstraction ('τ_I'), which we will use in this paper. It is based on the process algebras $\mathrm{BPA}_{\rho\delta U}$ [3] and timed μCRL [15]. With $a^{@u}$, we express that action a can be fired at time u. We consider the following transition rules, where the synchronisation of two actions $a, b \in \mathcal{A}_\tau$ resulting in an action c is denoted by $a \mid b = c$. The deadlock process is δ, which cannot fire any action, nor terminate successfully. If two actions a and b should never synchronise, we define that $a \mid b = \delta$. Usually, there are no synchronisations of τ actions defined, hence $\forall a.\tau \mid a = \delta \wedge a \mid \tau = \delta$. The process term $\tau_I(x)$ behaves as x with all action labels appearing in $I \subseteq \mathcal{A}$ rewritten to τ. Finally, the process term $u \gg x$ limits x to those alternatives with a first action timed at least at u.

$$\frac{}{\surd\!\downarrow} \qquad \frac{}{a^{@u} \xrightarrow{a}_u \surd} \qquad \frac{x \xrightarrow{a}_u x'}{x+y \xrightarrow{a}_u x' \quad y+x \xrightarrow{a}_u x'} \qquad \frac{x \xrightarrow{a}_u \surd}{x+y \xrightarrow{a}_u \surd \quad y+x \xrightarrow{a}_u \surd}$$

$$\frac{x \xrightarrow{a}_u x'}{x \cdot y \xrightarrow{a}_u x' \cdot y} \qquad \frac{x \xrightarrow{a}_u \surd}{x \cdot y \xrightarrow{a}_u u \gg y}$$

$$\frac{x \xrightarrow{a}_u \surd \quad v \leq u}{v \gg x \xrightarrow{a}_u \surd} \qquad \frac{x \xrightarrow{a}_u x' \quad v \leq u}{v \gg x \xrightarrow{a}_u x'}$$

$$\frac{x \xrightarrow{a}_u x' \qquad \mathcal{U}(y,u)}{x||y \xrightarrow{a}_u x'||y \quad y||x \xrightarrow{a}_u y||x'} \qquad \frac{x \xrightarrow{a}_u \surd \qquad \mathcal{U}(y,u)}{x||y \xrightarrow{a}_u y \quad y||x \xrightarrow{a}_u y}$$

$$\frac{x \xrightarrow{a}_u x' \quad y \xrightarrow{b}_u y' \quad a|b=c\neq\delta}{x||y \xrightarrow{c}_u x'||y'} \qquad \frac{x \xrightarrow{a}_u \surd \quad y \xrightarrow{b}_u y' \quad a|b=c\neq\delta}{x||y \xrightarrow{c}_u y' \quad y||x \xrightarrow{c}_u y'} \qquad \frac{x \xrightarrow{a}_u \surd \quad y \xrightarrow{b}_u \surd \quad a|b=c\neq\delta}{x||y \xrightarrow{c}_u \surd \quad y||x \xrightarrow{c}_u \surd}$$

$$\frac{x \xrightarrow{a}_u x' \qquad a \in I}{\tau_I(x) \xrightarrow{\tau}_u \tau_I(x')} \qquad \frac{x \xrightarrow{a}_u x' \qquad a \notin I}{\tau_I(x) \xrightarrow{a}_u \tau_I(x')} \qquad \frac{x \xrightarrow{a}_u \surd \qquad a \notin I}{\tau_I(x) \xrightarrow{a}_u \surd} \qquad \frac{x \xrightarrow{a}_u \surd \qquad a \in I}{\tau_I(x) \xrightarrow{\tau}_u \surd}$$

[2] Clearly, this last case is very demanding in a dense time domain, since the states need to be relatable at *all* times during the delay. One of the main reasons for introducing partial timing, as introduced later in this paper, is therefore to alleviate this requirement in specific situations (see Example 1).

$\mathcal{U}(\surd,0)$ $\mathcal{U}(a^{@u},v)$ *if* $v \leq u$ $\mathcal{U}(x+y,v) \Leftrightarrow \mathcal{U}(x,v) \vee \mathcal{U}(y,v)$ $\mathcal{U}(x \cdot y,v) \Leftrightarrow \mathcal{U}(x,v)$ $\mathcal{U}(v \gg x,v) \Leftrightarrow \mathcal{U}(x,v)$
$\mathcal{U}(\delta^{@u},v)$ *if* $v \leq u$ $\mathcal{U}(x||y,v) \Leftrightarrow \mathcal{U}(x,v) \wedge \mathcal{U}(y,v)$ $\mathcal{U}(\tau_I(x),v) \Leftrightarrow \mathcal{U}(x,v)$ $\mathcal{U}(u \gg x,v)$ *if* $v \leq u$

We note that with this process algebra and absolute timing, recursion is possible if we parameterise recursion variables with the current time (see Section 4).

3 Towards Partial Timing

The main idea of partial timing is the ability to ignore exact timing of parts of a system if only action orderings are important in those parts for a verification task or comparison of systems. It is, therefore, similar to action abstraction, by which it is possible to hide action labels not important for a verification task. We prefer the possibility to hide timing aspects independently from hiding action labels, unlike [16], because first of all, timed τ-steps may be useful in practice, and second of all, likewise, time-hidden labelled actions can be very practical.

Next, we define a *partially timed labelled transition system* (PTLTS), i.e. an LTS which can incorporate timing information in specific parts. One issue is what the delayability of a state should be if it has time-hidden outgoing transitions. Time-hidden transitions cannot introduce time deadlocks, since time is irrelevant for them. However, from a process term, we do not wish to derive ill-timed traces (see Example 3). For this reason, we define a process term ultimate delay relation $\mathcal{U}$ and a state ultimate delay relation $\bar{\mathcal{U}}$. $\mathcal{U}$ is defined at the end of this Section, and $\bar{\mathcal{U}}$ is defined here. $\mathcal{U}$ in Def. 1 is extended to $\bar{\mathcal{U}}$ by expressing that time-hidden transitions have no influence on the delayability of the source state. In the following, $\mathbb{B}$ is the boolean domain, with elements T (true) and F (false).

Definition 3 (Partially timed labelled transition system). *A* partially timed labelled transition system *(*PTLTS*) is a triple* $(\mathcal{S},\mathcal{T},\bar{\mathcal{U}})$*, where:*

1. $\mathcal{S}$ *is a set of states, including a state* $\surd$ *to represent successful termination;*
2. $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathbb{T} \times \mathbb{B} \times \mathcal{S}$ *is a set of transitions;*
3. $\bar{\mathcal{U}} \subseteq \mathcal{S} \times \mathbb{T}$ *is a* delay relation*, which satisfies:*
   - $\bar{\mathcal{U}}(s,0)$*;*
   - *For* $u > 0$*,* $\bar{\mathcal{U}}(s,u)$ *iff there exist* $v \geq u \in \mathbb{T}, \ell \in \mathcal{A}, s' \in \mathcal{S}$ *such that* $\mathcal{T}(s,\ell,v,\mathsf{T},s')$*.*

Transitions (s,ℓ,u,T,s') express that state s evolves into state s' by execution of action ℓ at (absolute) time u. Transitions (s,ℓ,u,F,s') express that state s evolves into state s' by the execution of action ℓ at some unobservable time.[3] It is assumed that the execution of transitions does not consume any time. A transition (s,ℓ,u,T,s') is denoted by $s \xrightarrow{\ell}_u s'$, (s,ℓ,u,F,s') is denoted by $s \xrightarrow{\ell}_{[u]} s'$. We write $s \xrightarrow{\ell} s'$ as a short-hand for $\exists v \in \mathbb{T}. s \xrightarrow{\ell}_{[v]} s'$ in case we are

[3] Although the exact time of execution is given in (s,ℓ,u,F,s'), it will not be observable by the partially timed branching bisimilarity relation given later. We still keep the time stamp though, because we reason the timing is in fact still there and should have an influence on the (interaction of) system behaviour.

not interested in the time stamp. With $\xrightarrow{\ell}_{(u)}$, we denote the possibility to either perform $\xrightarrow{\ell}_u$ or $\xrightarrow{\ell}$, and $\Rightarrow_{(u)}$ denotes the reflexive transitive closure of $\xrightarrow{\tau}_{(u)}$, i.e. time-hidden and timed τ-steps can succeed each other. Finally, $s \Rightarrow_{(u)}^{(v)} s'$ denotes that there exist $s_0, \ldots, s_n \in \mathcal{S}$, for some $n \in \mathbb{N}$, with $s_n = s'$ and $u_0 < \cdots < u_n \in \mathbb{T}$ with $u_0 = u$ and $u_n = v$, such that $s \Rightarrow_{(u_0)} s_0 \Rightarrow_{(u_1)} \cdots \Rightarrow_{(u_n)} s_n$. If $\bar{\mathcal{U}}(s, u)$, then state s can let time pass until time u.

Now, we define a PTB bisimulation, which intuitively combines untimed and timed branching bisimilarity.

Definition 4 (Partially timed branching bisimulation). *Assume a* PTLTS $(\mathcal{S}, \mathcal{T}, \bar{\mathcal{U}})$*. A collection B of binary relations $B_u \subseteq \mathcal{S} \times \mathcal{S}$ for $u \in \mathbb{T}$ is a* partially timed branching bisimulation *if $s\ B_u\ t$ implies:*

1. *if $s \xrightarrow{\ell}_u s'$, then*
   i. *either $\ell = \tau$ and $s'\ B_u\ t$,*
   ii. *or $t \Rightarrow_{(u)} \hat{t} \xrightarrow{\ell}_{(u)} t'$ with $s\ B_u\ \hat{t}$ and $s'\ B_u\ t'$;*
2. *if $t \xrightarrow{\ell}_u t'$, then vice versa;*
3. *if $s \xrightarrow{\ell} s'$, then*
   i. *either $\ell = \tau$ and $s'\ B_u\ t$,*
   ii. *or for some $v \geq u$, $t \Rightarrow_{(u)}^{(v)} \hat{t} \xrightarrow{\ell} t'$ with $s\ B_v\ \hat{t}$ and $s'\ B_v\ t'$;*
4. *if $t \xrightarrow{\ell} t'$, then vice versa;*
5. *if $s \downarrow$, then $t \Rightarrow_{(u)} t' \downarrow$ with $s\ B_u\ t'$;*
6. *if $t \downarrow$, then vice versa;*
7. *if $u \leq v$ and $\bar{\mathcal{U}}(s, v)$, then for some $n \geq 0$ there are $t_0, \ldots, t_n \in \mathcal{S}$ with $t = t_0$ and either $\bar{\mathcal{U}}(t_n, v)$ or $t_n \xrightarrow{\ell} t'$, and $u_0 < \cdots < u_n \in \mathbb{T}$ with $u = u_0$ and $v = u_n$, such that for $i < n$, $t_i \Rightarrow_{(u_i)} t_{i+1}$ with $s\ B_w\ t_{i+1}$ for $u_i \leq w \leq u_{i+1}$;*
8. *if $u \leq v$ and $\bar{\mathcal{U}}(t, v)$, then vice versa.*

Two states s and t are PTB bisimilar at u*, denoted by $s \leftrightarrows_{ptb}^u t$, if there is a* PTB *bisimulation B with $s\ B_u\ t$. States s and t are* PTB bisimilar*, denoted by $s \leftrightarrows_{ptb} t$, if they are* PTB *bisimilar at all $u \in \mathbb{T}$.*

Cases 1 and 5 (and 2 and 6) directly relate to cases 1 and 3 (and 2 and 4) of Def. 2, respectively, the only difference being that time-hidden τ-steps are taken into account. Note that in 1.*ii*, a timed ℓ-step can be matched with a sequence of τ-steps followed by a *time-hidden* ℓ-step. We reason that a time-hidden ℓ-step can simulate a timed ℓ-step, because the first is not observably restricted by timing, while the latter is. The reverse is not true for the same reason. Besides, if we defined a timed ℓ-step and a time-hidden ℓ-step to be PTB bisimilar, then PTB bisimilarity would definitely not be an equivalence, which is undesired, since it would not be transitive. Consider $q \xrightarrow{\ell}_0 q'$, $r \xrightarrow{\ell}_1 r'$, and $s \xrightarrow{\ell} s'$ with $\ell \neq \tau$; clearly $q \not\leftrightarrows_{ptb} r$, and therefore we either want $q \not\leftrightarrows_{ptb} s$ or $r \not\leftrightarrows_{ptb} s$. Case 3.*i* (4.*i*) is similar to 1.*i* (2.*i*), but dealing with a time-hidden τ-step. Case 3.*ii* (4.*ii*) states that a time-hidden ℓ-step can be matched by a sequence of τ-steps in

which time may pass, leading to a state where a time-hidden ℓ-step is enabled. Case 7 (8) differs from case 5 (6) of Def. 2 in taking time-hidden τ-steps into account. Also, besides the possibility to match a delay by a τ-sequence to a state which can delay to the same moment in time, this latter state may just have the possibility to perform a time-hidden step (see Example 2). The idea behind cases 3.*ii* (4.*ii*) and 7 (8) is that time-hidden steps are not observably subject to delays, i.e. the progress of time does not disable a time-hidden step. Consider the PTLTSs $s \xrightarrow{\tau}_2 s' \xrightarrow{a} s''$ and $t \xrightarrow{a} t'$. Here, $s \leftrightarrows^0_{ptb} t$, since a delay of s to time 2 can be matched by t since it can perform a time-hidden step (case 7 (8) of Def. 4). Note that case 3 (4) is less demanding than 7 (8) concerning time; the first demands that states are related at the transition times, while the latter demands relations at all times. This is because in 3, a time-hidden step is fired, while in 7, s performs a delay, and is therefore time constrained.

Note that the union of PTB bisimulations is again a PTB bisimulation.

Theorem 1. *Timed branching bisimilar process terms P and Q are also* PTB *bisimilar, i.e. $\leftrightarrows_{tb} \subset \leftrightarrows_{ptb}$.*

Proof. A timed branching bisimulation relation B_u is a PTB bisimulation relation B'_u, since case 1.*i* (2.*i*) for B_u matches 1.*i* (2.*i*) for B'_u, 1.*ii* (2.*ii*) for B_u matches 1.*ii* (2.*ii*) for B'_u since $t \Rightarrow_{(u)} \hat{t} \xrightarrow{\ell}_{(u)} t'$ may consist of only timed steps. Cases 3 and 4 for B'_u are not applicable, since they concern time-hidden transitions. Case 3 (4) for B_u matches 5 (6) for B'_u since $t \Rightarrow_{(u)} t'$ may consist of only timed τ-steps. Finally, case 5 (6) for B_u matches 7 (8) for B'_u since $\bar{\mathcal{U}}(s,v) \Leftrightarrow \mathcal{U}(s,v)$ in the absence of time-hidden steps, and $\bar{\mathcal{U}}(t_n, v)$ is the only applicable condition for t_n. Furthermore, $t_i \Rightarrow_{(u_i)} t_{i+1}$ may only concern timed τ-steps. □

Similarly, we can prove that branching bisimilar terms are also PTB bisimilar. Furthermore, in [18], we prove that PTB bisimilarity is an equivalence relation.

In the following examples, $\mathbb{Q}_{\geq 0} \subseteq \mathbb{T}$.

Example 1. Consider the LTSs $s_0 \xrightarrow{a}_{\frac{1}{3}} s_1 \xrightarrow{b}_1 s_2$ and $t_0 \xrightarrow{a}_{\frac{1}{3}} t_1 \xrightarrow{\tau}_{\frac{1}{2}} t_2 \xrightarrow{b}_1 t_3$. According to Def. 2, clearly $s_0\ B_u\ t_0$ for $u \geq 0$, and we need to determine that $s_1\ B_{\frac{1}{3}}\ t_1$. Since $\mathcal{U}(s_1, 1)$, by case 5 of Def. 2, we must check that we can reach a state t_i from t_1 via τ-steps, such that $\mathcal{U}(t_i, 1)$ and $s_1\ B_u\ t_i$ for some $u \geq \frac{1}{3}$. This means explicitly establishing that $s_1\ B_u\ t_1$ for all $\frac{1}{3} \leq u \leq \frac{1}{2}$ and $s_1\ B_u\ t_2$ for all $\frac{1}{2} \leq u \leq 1$. Moving to PTLTSs, $\mathcal{U}$ from Def. 1 and $\bar{\mathcal{U}}$ coincide, and case 7 of Def. 4 is similarly applicable. However, if we time-hide b, then by definition, $\bar{\mathcal{U}}(s_1, u)$ only for $u = 0$, hence case 7 is no longer a requirement. Note that if we only time-hide $t_2 \xrightarrow{b}_1 t_3$, the second PTLTS still simulates the first one.

Example 2. Consider the PTLTSs $s_0 \xrightarrow{a}_0 s_1 \xrightarrow{b}_{[3]} s_2$ and $t_0 \xrightarrow{a}_0 t_1 \xrightarrow{\tau}_1 t_2 \xrightarrow{b}_{[2]} t_3$. We have $s_0 \leftrightarrows_{ptb} t_0$, since $s_0\ B_u\ t_0$ for $u \geq 0$, $s_1\ B_u\ t_1$ for $u \leq 1$, $s_1\ B_u\ t_2$ for $u \geq 0$, and $s_2\ B_u\ t_3$ for $u \geq 0$ is a PTB bisimulation.

In Example 2, the time-hidden b-step from s_1 can be matched with the delay $\bar{\mathcal{U}}(t_1, 1)$ (note in case 8 (7) of Def. 4 that the delay can simply be matched by s_1

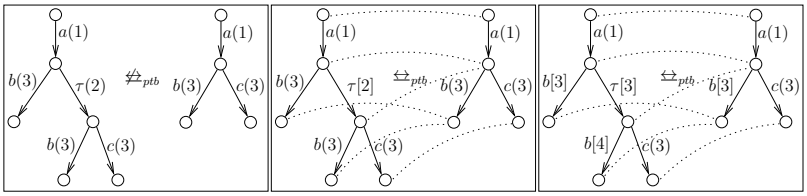

Fig. 1. Minimisation due to time abstraction

by the fact that it can fire a time-hidden step), followed by the timed silent step from t_1 (by case 2.i (1.i)) and the time-hidden b-step from t_2 (by case 4 (3)). For time-hidden steps, there are no delay requirements, which greatly alleviates the demands of bisimilarity of timed systems when time-hiding is applied to them.

Like action abstraction, time abstraction may lead to the minimisation of LTSs, particularly when delayable actions are time-hidden. To illustrate this, in an untimed setting, $a{\cdot}(\tau{\cdot}(x+y)+x) = a{\cdot}(x+y)$ is an important axiom by which LTSs can be minimised using branching bisimilarity. Reniers and Van Weerdenburg [16] note that for a timed branching bisimilarity, a timed version of the axiom does not hold, and using untimed τ-steps reintroduces it. We show that in a partially timed setting, even a more general version of the axiom holds.

On the left of Figure 1, in which transitions $\xrightarrow{\ell}_u$ are written as $\xrightarrow{\ell(u)}$, the two PTLTSs are not PTB bisimilar, since concerning the states reached by taking the a-step, in the left PTLTS, at time 2 the decision can be made to take the b-step at time 3 by ignoring the τ-step, where in the right one, this cannot be done. In the middle of Figure 1, the τ-step is time-hidden, hence time does not influence the possibility to take the step, thus the two PTLTSs are PTB bisimilar, relatable states being connected with dashed lines. So far, this is similar to [16]. On the right of Figure 1, we have a PTLTS with a delayable action b; depending on whether the τ-step is performed or not, b may be executed at time 3 or 4. Such a situation is common in real system models. If we time-hide b, further reduction is possible. Note that it is not required to time-hide the silent step.

Note that time-hiding can alleviate the delay requirements of a (partially) timed branching bisimilarity, since time-hidden transitions are enabled immediately. E.g. at time u, we may fire a transition $s \xrightarrow{\ell}_v s'$ at $v \geq u$ after a delay to time v, while a transition $s \xrightarrow{\ell}_{[v]} s'$ can be fired immediately.

A Partially Timed Process Algebra Next, we extend the process algebra of Section 2. With $a^{[@u]}$, we express that a can be fired at some hidden time. Keeping the exact timing when time-hiding is essential to ensure that additional system behaviour is not introduced. Consider the term $a^{@2}{\cdot}b^{@1}$. Surely b cannot be executed, since the time has already progressed too far. If we time-hide b, we do not want to lose this, since an observer might be unable to observe the timing of b, but in our opinion this does not mean that the timing is not there. The same argument holds for e.g. interleavings resulting from parallel compositions.

First, we define two versions of time abstraction, one for hiding timing information of specific action labels, the other for hiding a certain time interval.

Action-based Time Abstraction Transition Rules In action-based time abstraction, timing information of all transitions with an action label in a given set $J \subseteq \mathcal{A}_\tau$ is hidden. The process term $\iota_J(P)$ hides timing of all actions of P in J.

$$\frac{x \xrightarrow{a}_u x' \quad a \in J}{\iota_J(x) \xrightarrow{a}_{[u]} \iota_J(x')} \qquad \frac{x \xrightarrow{a}_u \surd \quad a \in J}{\iota_J(x) \xrightarrow{a}_{[u]} \surd} \qquad \frac{x \xrightarrow{a}_u x' \quad a \notin J}{\iota_J(x) \xrightarrow{a}_u \iota_J(x')}$$

$$\frac{x \xrightarrow{a}_u \surd \quad a \notin J}{\iota_J(x) \xrightarrow{a}_u \surd} \qquad \frac{x \xrightarrow{a}_{[u]} x'}{\iota_J(x) \xrightarrow{a}_{[u]} \iota_J(x')} \qquad \frac{x \xrightarrow{a}_{[u]} \surd}{\iota_J(x) \xrightarrow{a}_{[u]} \surd}$$

Time-based Time Abstraction Transition Rules All timing information between times u_1 and u_2 ($u_1, u_2 \in \mathbb{T}, u_2 \geq u_1$) are hidden. Such a time abstraction is useful if you want to hide the timing specifics of a certain time interval. The process term $\bar{\iota}_{u_1,u_2}(P)$ hides timing of all actions of P if $u_1 \leq u \leq u_2$.

$$\frac{x \xrightarrow{a}_u x' \quad u_1 \leq u \leq u_2}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_{[u]} \bar{\iota}_{u_1,u_2}(x')} \qquad \frac{x \xrightarrow{a}_u \surd \quad u_1 \leq u \leq u_2}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_{[u]} \surd} \qquad \frac{x \xrightarrow{a}_u x' \quad u_1 > u \vee u_2 < u}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_u \bar{\iota}_{u_1,u_2}(x')}$$

$$\frac{x \xrightarrow{a}_u \surd \quad u_1 > u \vee u_2 < u}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_u \surd} \qquad \frac{x \xrightarrow{a}_{[u]} x'}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_{[u]} \bar{\iota}_{u_1,u_2}(x')} \qquad \frac{x \xrightarrow{a}_{[u]} \surd}{\bar{\iota}_{u_1,u_2}(x) \xrightarrow{a}_{[u]} \surd}$$

Additional Rules for Existing Operators Looking back at the basic process algebra of Section 2, all rules except those for parallel composition can also straightforwardly be interpreted with $@u$ and $\xrightarrow{\ell}_u$ substituted by $[@u]$ and $\xrightarrow{\ell}_{[u]}$, respectively. Action $a^{[@u]}$ can then be seen as the time-hidden action a.

We define the following additional transition rules:

$$\frac{x \xrightarrow{a}_{[u]} x' \quad \mathcal{U}(y,u)}{\begin{array}{c} x||y \xrightarrow{a}_{[u]} x'||y \\ y||x \xrightarrow{a}_{[u]} y||x' \end{array}} \qquad \frac{x \xrightarrow{a}_{[u]} \surd \quad \mathcal{U}(y,u)}{\begin{array}{c} x||y \xrightarrow{a}_{[u]} y \\ y||x \xrightarrow{a}_{[u]} y \end{array}} \qquad \frac{x \xrightarrow{a}_{[u]} x' \quad y \xrightarrow{b}_u y' \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_u x'||y' \\ y||x \xrightarrow{c}_u y'||x' \end{array}}$$

$$\frac{x \xrightarrow{a}_{[u]} \surd \quad y \xrightarrow{b}_u y' \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_u y' \\ y||x \xrightarrow{c}_u y' \end{array}} \qquad \frac{x \xrightarrow{a}_{[u]} x' \quad y \xrightarrow{b}_u \surd \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_u x' \\ y||x \xrightarrow{c}_u x' \end{array}}$$

$$\frac{x \xrightarrow{a}_{[u]} \surd \quad y \xrightarrow{b}_u \surd \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_u \surd \\ y||x \xrightarrow{c}_u \surd \end{array}} \qquad \frac{x \xrightarrow{a}_{[u]} x' \quad y \xrightarrow{b}_{[u]} y' \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_{[u]} x'||y' \\ y||x \xrightarrow{c}_{[u]} y'||x' \end{array}}$$

$$\frac{x \xrightarrow{a}_{[u]} \surd \quad y \xrightarrow{b}_{[u]} y' \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_{[u]} y' \\ y||x \xrightarrow{c}_{[u]} y' \end{array}} \qquad \frac{x \xrightarrow{a}_{[u]} \surd \quad y \xrightarrow{b}_{[u]} \surd \quad a|b=c \neq \delta}{\begin{array}{c} x||y \xrightarrow{c}_{[u]} \surd \\ y||x \xrightarrow{c}_{[u]} \surd \end{array}}$$

To ensure that the parallel composition operator yields well-timed interleavings, the (process algebra related) ultimate delay relation $\mathcal{U}$ ignores time abstraction. Therefore, its definition from Section 2 is extended with $\mathcal{U}(a^{[@u]}, v)$ if $v \leq u$, $\mathcal{U}(\delta^{[@u]}, v)$ if $v \leq u$, $\mathcal{U}(\iota_J(x), v) \Leftrightarrow \mathcal{U}(x, v)$ and $\mathcal{U}(\bar{\iota}_{u_1,u_2}(x), v) \Leftrightarrow \mathcal{U}(x, v)$.

Example 3. Consider process terms $a^{[@1]} \parallel b^{@3}$ and $a^{[@2]}{\cdot}b^{@3}$. By definition, $\mathcal{U}(a^{[@1]}, u)$ for $u \leq 1$. This ensures that only $s_0 \xrightarrow{a}_{[1]} s_1 \xrightarrow{b}_3 s_2$ can be yielded from $a^{[@1]} \parallel b^{@3}$, i.e. the timing of a is not observable, but it is still there. On the other hand, $\bar{\mathcal{U}}$ ensures that $s_0 \xrightarrow{a}_{[1]} s_1 \xrightarrow{b}_3 s_2$ and $t_0 \xrightarrow{a}_{[2]} t_1 \xrightarrow{b}_3 t_2$ (yielded from $a^{[@2]}{\cdot}b^{@3}$) are PTB bisimilar, since both $\bar{\mathcal{U}}(s_0, u)$ and $\bar{\mathcal{U}}(t_0, u)$ only for $u = 0$.

We use $\mathcal{U}$ and $\bar{\mathcal{U}}$ for process terms and PTLTS*s*, respectively, since on the one hand, we do not want to enable extra orderings of actions when time-hiding, but on the other hand, we want to formalise that when comparing behaviour, delays do not apply for states from which only time-hidden transitions can be fired.

4 A Case Study: The Sliding Window Protocol

Now, we show the practical use of the techniques with a case study. For this, we extend the basic timed process algebra with guards, recursion, and data. Given a boolean condition b and process terms P, Q, a process term $P \lhd b \rhd Q$ behaves as P iff $b = \mathsf{T}$, and as Q otherwise. Constructs can be parameterised with data, range over data domains, and incorporate recursion. For example, a process $P_t(d : \mathbb{D}) = \sum_{e:\mathbb{D}} a^{@t+t'}(e){\cdot}P_{t+t'}(e)$, with d a variable of type $\mathbb{D}$, can execute actions a parameterised with any value of type $\mathbb{D}$, expressed using choice quantification [13, 15], continuously, each time advancing the time by t' time units, and changing the state of P to the chosen value. Note that if $\mathbb{D}$ is an infinite data domain, this implies an impractical infinite number of alternatives. However, such a construct often represents receiving a message, and enforced synchronisation with a send action b then leads to a finite number of alternatives. Enforced synchronisation can be achieved with the *encapsulation* operator ∂_H, with $H \subseteq \mathcal{A}$. It disables firing the actions in the set H individually; only their synchronisation result can thus be fired. E.g. from $\partial_{\{a,b\}}(\sum_{n:\mathbb{N}} a(n) \parallel b(5))$, with $a \mid b = c$, we can only derive a transition labelled $c(5)$, since all options $a(n)$ with $n \neq 5$ cannot synchronise with $b(5)$ and are therefore blocked. Finally, $P_t = p{\cdot}P'_{t'}$ is a process which executes the process described by process term p, built using the operators of the timed process algebra, ending up in process $P'_{t'}$.

We do not yet have a real implementation of time abstraction and PTB bisimilarity, but we can obtain similar results for particular cases using the untimed μCRL [7] and LTSmin [8] toolsets. We stress, though, that this is far from ideal, and a real implementation is highly preferable. We model t as a data parameter of the process equations and actions (with $\mathbb{T} = \mathbb{Z}_{\geq 0}$), e.g. we model an action $a^{@t}(d)$, with d some data parameter, as $a(t, d)$. We have extended the μCRL LTS generator such that these modelled time stamps are correctly interpreted in parallel compositions (i.e. ensuring that e.g. $a(1, d) \parallel b(2, d')$ cannot yield a trace $s_0 \xrightarrow{b(2,d')} s_1 \xrightarrow{a(1,d)} s_2$), and developed a tool to time-hide a given set of actions in an LTS, i.e. to remove the time parameter from all occurrences of these actions, e.g. $\xrightarrow{a(t,d)}$ is rewritten to $\xrightarrow{a(d)}$ if a should be time-hidden. Furthermore, we use implementations of untimed strong and branching bisimilarity to minimise the LTS*s*. Please note that this approach has a number of limitations, among which:

1. We cannot use timed τ-steps, because the untimed bisimilarities cannot handle them. Time stamps on other steps are fine; since the time stamps are incorporated in the data parameters, they will be part of the comparison;
2. Time-hiding is applied on LTS*s*, hence globally. We cannot use time-hiding on e.g. specific occurrences of an action a. Furthermore, in practice, having to generate the full LTS before time-hiding can be applied can be a bottle-neck;
3. The progress of time should be ensured in the specification; time-deadlocks are not detected by the tools. This makes modelling quite hard;
4. We cannot use continuous time; in this Section, $\mathbb{T} = \mathbb{Z}_{\geq 0}$.

Next, we consider a timed *Sliding Window* (SW) protocol based on [9], which is a nice example of a complex, timed system. We use it to demonstrate how a PTLTS can be minimised for specific verification tasks. For a detailed explanation of the protocol, see [9]. A sender S needs to send a stream of data packets to a receiver R over an unreliable channel which may reorder, lose, and duplicate packets. The packets are labelled with sequence numbers modulo a given K, and they are processed in batches (windows) of size N. S may send new packets in its current window in the correct order, and resend any old packet in the window at any time. When R receives a packet for the first time, it is placed in the correct position in a buffer, and R sends an acknowledgement to S over an equally unreliable channel. R sorts the received packets on their sequence numbers, and, whenever possible, delivers some of the packets in the correct order on a channel as output. When S receives an acknowledgement, he moves his window forward beyond the acknowledged packet. An important correctness requirement is that $K \geq 2 * N$, and to ensure that sequence numbers are not reused too quickly, timing constraints are set; when S receives an acknowledgement for packet $K-1$, he waits $Lmax$ time units before sending a new packet 0 (constraint **CS**). R accepts packets with new numbers $Lmax$ time units after delivering a packet $K-1$ to the output (constraint **CR**) [9]. The channels nondeterministically delay the packets sent through them, and this aspect allows us to obtain smaller PTLTS*s* if we choose to abstract away the related timing. We illustrate this by providing part of the specification of S, and the entire specification of Fch, the channel between S and R, with $sa \mid ra = ca$:

$$
\begin{aligned}
&S(t:\mathbb{T}, first:\mathbb{N}, ftsend:\mathbb{N}, tackmax:\mathbb{N}, nrm:\mathbb{N}) = \\
&\quad sa(t+t', ftsend)\cdot S(t+t', first, (ftsend+1)\ \mathbf{mod}\ K, tackmax, nrm-1) \\
&\qquad\qquad \lhd\ (ftsend = 0\ \Longrightarrow\ ((t+t') > (tackmax + Lmax) \wedge (first = ftsend))) \wedge \\
&\qquad\qquad \mathbf{inmod}(ftsend, first, first+N, K) \wedge nrm > 0 \rhd \delta + \\
&\quad \sum_{i:\mathbb{N}<K} sa(t+t', i)\cdot S(t+t', first, ftsend, tackmax, nrm) \\
&\qquad\qquad \lhd\ \mathbf{inmod}(i, first, ftsend, K) \wedge t < 30 \rhd \delta + \ldots
\end{aligned}
$$

$$
\begin{aligned}
&Fch(t:\mathbb{T}, L:(\mathbb{N},\mathbb{T})\mathbf{list}, rbnd:\mathbb{N}, sbnd:\mathbb{N}) = \\
&\quad \sum_{x:\mathbb{N}} \sum_{t_1:\mathbb{T}\geq t} ra(t_1, x)\cdot Fch(t_1, \mathbf{ins}((x, t_1 + (Lmax/2)), L), 3, sbnd) + \\
&\quad \sum_{x:\mathbb{N}} \sum_{t_1:\mathbb{T}\geq t} ra(t_1, x)\cdot Fch(t_1, \mathbf{ins}((x, t_1 + Lmax), L), 3, sbnd) +
\end{aligned}
$$

$$\sum_{x:\mathbb{N}} \sum_{t_1:\mathbb{T}\geq t} ra(t_1,x)\cdot Fch(t_1,L,rbnd-1,sbnd) \lhd\ rbnd > 0 \rhd \delta +$$
$$sb(\mathbf{headtime}(L),\mathbf{headnr}(L))\cdot Fch(\mathbf{headtime}(L),\mathbf{tail}(L),rbnd,3)$$
$$\lhd\ \mathbf{nonempty}(L) \wedge t \leq \mathbf{headtime}(L) \rhd \delta +$$
$$sb(\mathbf{headtime}(L),\mathbf{headnr}(L))\cdot Fch(\mathbf{headtime}(L),L,rbnd,sbnd-1)$$
$$\lhd\ \mathbf{nonempty}(L) \wedge t \leq \mathbf{headtime}(L) \wedge sbnd > 0 \rhd \delta$$

In S, t is the current time, *first* is the first packet in the current window, and *ftsend* is the next packet to send; *tackmax* equals the last time an acknowledgement for packet $K-1$ has been received, and *nrm* is the maximum number of new packets we want S to send. Special functions are written in boldface: **mod** is as usual, and **inmod** returns whether or not the first value lies between the second and the third value modulo the fourth value. The two options of S express sending new packets and resending old packets, respectively, where t' is a discrete jump forward in time (in our experiments, $t'=1$) and i ranges over the old packets. Since $sa \mid ra = ca$, packets can be sent to the channel *Fch* if matching *sa* and *ra* are enabled. In *Fch*, besides the current time, we have a special list of tuples $(x:\mathbb{N}, t:\mathbb{T})$, with functions **ins**, which inserts a new tuple in the list ordered by increasing time t, and **headtime** and **headnr**, which return t and x of the head of the list, respectively; finally, **nonempty** determines whether the list is empty or not. Note that we use two bounds for receiving and sending packets, *rbnd* and *sbnd*, to ensure that packets are not continuously ignored by the channel; we reset these bounds to 3 after each successful receive and send.

The key observation can be done in the receive options of *Fch*: a packet can be stored in L with two time values: $t_1 + Lmax/2$ and $t_1 + Lmax$ (with t_1 the time of receiving the packet, and $Lmax$ the maximum packet lifetime), and it can be ignored.[4] The time chosen here determines when the packet is sent with *sb* to R, which can fire action *rb* to receive packets. If we time-hide *sb* in the given specification part,[5] then traces can be 'folded' together if they concern the same packet numbers in the same order, i.e. we only care about the order in which the packets are sent, not at which moments in time. This can lead to rigorous PTLTS reductions; consider two traces $s_0 \xrightarrow{ca(0)}_0 s_1 \xrightarrow{ca(1)}_1 s_2 \xrightarrow{sb(0)}_{Lmax/2} s_3 \xrightarrow{sb(1)}_{(Lmax+1)/2} s_4$ and $t_0 \xrightarrow{ca(0)}_0 t_1 \xrightarrow{ca(1)}_1 t_2 \xrightarrow{sb(0)}_{Lmax} t_3 \xrightarrow{sb(1)}_{Lmax+1} t_4$. If we time-hide the *sb*-steps, these traces are PTB bisimilar (since $s_i \leftrightarrows^u_{ptb} t_i$ for $0 \leq i \leq 4$, $u \geq 0$), and hence can be reduced to one trace. The benefit of time-hiding here, is that even if we hide time entirely, time restrictions still apply; e.g. **CS** and **CR** still limit the potential behaviour of S. Note that as we allow more freedom in the packet delays, i.e. allow more than only two durations, the achieved reduction will increase.

Table 1 presents the size of an LTS generated using the μCRL toolset on 30 workstations in a distributed fashion, where each workstation had a quad-core INTEL XEON processor E5520 2.27 GHz, 24 GB RAM, and was running DEBIAN 2.6.26-19. We used the distributed reduction tool of LTSmin for the

[4] In the real protocol, any time between t_1 and $Lmax$ is possible.

[5] In the full specification, we actually time-hide the *cb*-transitions, where $sb \mid rb = cb$, and *sb* and *rb* are encapsulated.

Table 1. Sizes of LTSs describing the behaviour of a timed SW protocol

Description	# States	# Transitions
SW	335,236,894	1,940,674,714
$\leftrightarrows_s$ red. of ι_J(SW), $J = \mathcal{A} \setminus \{ca,\ cack2\}$	243,912,294	1,371,275,560
$\leftrightarrows_b$ red. of $\tau_J(\iota_J$(SW)), $J = \mathcal{A} \setminus \{ca,\ cack2\}$	7,231,576	48,686,049
$\leftrightarrows_b$ red. of $\tau_I(\iota_{\mathcal{A}}$(SW)), $I = \{ca\}$	234,398,155	1,310,272,020
$\leftrightarrows_b$ red. of $\iota_J(\tau_I(SW))$, $I = \mathcal{A} \setminus \{ca, deliver\}$, $J = \{\tau, ca\}$	7,450,689	49,944,368

strong ($\leftrightarrows_s$) and branching ($\leftrightarrows_b$) bisimulation reductions. The different rows in Table 1 display the results of reductions taylored for checking specific properties: we get a full finite LTS using specification SW with values $N = 2, K = 5, nrm = 7$ and $Lmax = 2$, and bounding the overall time progress to 30 time units. For verification of constraint **CS** (row 2), we can time-hide all steps except the ones labelled *ca* and *cack2* (which represents S receiving an acknowledgement of a packet with number $K - 1$). Then, we can also still analyse the action ordering; if this is not needed, we can action-hide these actions, and get a better reduction (row 3). Similar reductions can be done for constraint **CR**. Row 4 represents a reduction useful for verifying action orderings other than *ca*; the latter label is completely hidden, all others are only time-hidden. Finally, hiding all except *ca* and *deliver* (the label denoting that R delivers a packet as output) allows us to check that the packets are delivered by R in the correct order, which is the most important property to check. We can also still analyse the timing of packet delivery. The full specification plus experiment instructions can be found at [18].

5 Some Timing Considerations

In this paper, we focus on absolute time. Here, we remark on applying partial timing with relative time. Consider the relative time process term $a^{@1} \cdot b^{@2}$. If we time-hide a, we lose the information when b actually starts executing, and the overall execution time. One course of action ([16]) is to maintain the overall duration, 'shifting' the timing of time-hidden actions towards the timed actions. If we follow this, then in our example, we get $a \cdot b^{@3}$. We advocate instead to keep the timing information as in our approach with absolute time, e.g. $a^{[@1]} \cdot b^{@2}$. Time shifting first of all does not guarantee that the overall duration is kept, e.g. if we time-hide b, this is lost anyway, and second of all, it does not seem entirely correct that time-hiding a has an influence on the duration of the execution of b. The way in which time progress is described in relative time should be reflected in PTB bisimilarity and the time-hiding operators. E.g. a sequence of τ-steps at time u in absolute time should be expressed as a sequence of τ-steps with time label 0 in relative time. Time-based time abstraction should be defined such that the current time is kept track of, to detect which actions should be time-hidden.

In the *two-phase* model, in which time progress is expressed using additional delay actions and transitions, partial timing seems to be achievable by action-

hiding the delay actions; then, a branching bisimulation reduction results in their removal, in the absence of time non-determinism. This approach is straightforward when removing *all* timing; partial removal remains to be investigated.

Our next step is to investigate under which conditions a *rooted* [5, 14] PTB bisimilarity is a congruence over our process algebra with time abstraction. In general this is not the case, because PTB bisimilar terms may 'interact' differently with other terms, since their timing may be partially ignored, but it is still there.

6 Related Work

In [4], a time free projection operator π_{tf} is used in a relative discrete time setting. A process $\pi_{tf}(P)$ is the process P made time free. In contrast to our work, they do not consider the possibility to only make certain action labels time free, or abstract away a specific period of time.

[16] considers untimed τ-steps, which allows better timed rooted branching bisimulation minimisation. There, ill-timedness of processes can 'disappear' when hiding actions. In our system, a time-hidden τ-step can be obtained from any action $a^{@u}$, by hiding both action label and timing, e.g. $\iota_{\{\tau\}}(\tau_{\{a\}}(a^{@u}))$. They do not consider the use of timed τ-steps and time-hidden labelled steps.

Approaches using *regions* and *zones* for timed automata [2] and *state classes* for Time Petri Nets [6] certainly have something in common with our work. There, the abstractions work system-wide, using a bisimilarity which fully ignores timing. We, however, abstract away some of the timing in the specification itself, and use a bisimilarity which does *not* ignore timing, but can handle time-hidden behaviour as well. [2] uses a function *Untime*, which removes the timing of system behaviour without introducing new action orderings. This is applied on the state space, though; no definition is given to abstract timing away from (parts of) the specification. In [17], several time-abstracting bisimilarities are defined for a relative time setting with a two-phase model. Like PTB bisimilarity, their strong time-abstracting bisimilarity preserves branching-time properties. Interestingly, it seems that, as suggested in Section 5, this coincides with untimed branching bisimilarity if we action-hide all the delay actions and no other actions, assuming there is no time non-determinism. They do not consider timed branching bisimilarity with time abstraction, nor hiding parts of the timing.

7 Conclusions

We proposed a generalisation of timed process algebras, TLTSs, and timed branching bisimilarity, by introducing time abstraction, by which we can hide the timing of system parts, and PTB bisimilarity, which is proven to be an equivalence relation. This allows to consider two timed systems equal in functionality, even if they are (partially) different in their timings. By hiding timing, no functionality is removed or added, i.e. no new orderings of actions are introduced. We have discussed how time abstraction, like action abstraction, can lead to minimised LTSs. The practical use of time-hiding has been demonstrated with a case study.

Future Work We wish to fully implement the generalisations in a toolset. PTB bisimilarity can be analysed to see under which timing conditions it constitutes a congruence, and we wish to investigate a complete axiomatisation of the process algebra and PTB bisimilarity, plus decidability of the latter.

Acknowledgements We thank Wan Fokkink for his constructive comments.

References

1. R. Alur and D. Dill. Automata for Modeling Real-Time Systems. In *Proc. of ICALP 1990*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.
2. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.
4. J.C.M. Baeten, C.A. Middelburg, and M. Reniers. A New Equivalence for Processes with Timing – With an Application to Protocol Verification. CSR 02-10, Eindhoven University of Technology, 2002.
5. J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.
6. B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on Softw. Engin.*, 17(3):259–273, 1991.
7. 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 *Proceedings of CAV 2001*, volume 2102 of *LNCS*, pages 250–254. Springer, 2001.
8. S.C.C. Blom, J.C. van de Pol, and M. Weber. Bridging the Gap between Enumerative and Symbolic Model Checkers. CTIT Technical Report TR-CTIT-09-30, University of Twente, 2009.
9. D. Chkliaev, J. Hooman, and E. de Vink. Verification and Improvement of the Sliding Window Protocol. In *Proceedings of TACAS 2003*, volume 2619 of *LNCS*, pages 113–127. Springer, 2003.
10. W.J. Fokkink, J. Pang, and A.J. Wijs. Is Timed Branching Bisimilarity a Congruence Indeed? *Fundamenta Informaticae*, 87(3/4):287–311, 2008.
11. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.
12. K. Larsen and Y. Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1997.
13. S.P. Luttik. *Choice Quantification in Process Algebra*. PhD thesis, University of Amsterdam, 2002.
14. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
15. M.A. Reniers, J.F. Groote, M.B. van der Zwaag, and J. van Wamel. Completeness of Timed μCRL. *Fundamenta Informaticae*, 50(3-4):361–402, 2002.
16. M.A. Reniers and M. van Weerdenburg. Action Abstraction in Timed Process Algebra: The Case for an Untimed Silent Step. In *FSEN*, 2007.
17. S. Tripakis and S. Yovine. Analysis of Timed Systems using Time-Abstracting Bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
18. A.J. Wijs. Forgetting the Time in Timed Process Algebra - Appendix. `http://www.win.tue.nl/~awijs/timeabstraction/timeabs.html`, 2010.
19. M.B. van der Zwaag. The cones and foci proof technique for timed transition systems. *Information Processing Letters*, 80(1):33–40, 2001.