RWTH Aachen

Department of Computer Science
Technical Report

Introducing Timers to π-*Calculus*

Kamal Barakat

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2011-18

RWTH Aachen · Department of Computer Science · Feb. 2012 (revised version)

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Introducing Timers to π-*Calculus*

Kamal Barakat*

Lehrstuhl für Informatik 11
RWTH Aachen, Germany
Email: barakat@embedded.rwth-aachen.de

Abstract. We introduce a new concept of modeling timed behavior in pi-calculus by representing timed actions (or timers) as interactions between application processes and clock processes. This approach extends the original calculus in a manner such that bisimulation arrangements in pi-calculus remain untouched.

1 Introduction to *classic* π-*calculus*

π-*Calculus* is a communicating model that treats mobility in a native way. One could use this model for representing communication protocols, algorithms, programming languages and data structures. The syntax of π-*calculus* can be presented using the following expression

$$P ::= \sum_{i \in I} \pi_i \cdot P_i \mid P_1 | P_2 \mid \mathrm{new} aP \mid !P$$

where

$$\begin{aligned} \pi ::= \; & x(y) && \text{receive y along x} \\ & \overline{x}\langle y \rangle && \text{send y along x} \\ & \tau && \text{unobservable action} \end{aligned}$$

The summation construct represents nondeterminism in the model where a process has several action prefixes and hence several execution paths to choose from. The dot in any summation term is called the sequential composition operator which specifies the order of execution of action prefixes. The parallel composition operator "—" is used to compose a process from several subprocesses that are simultaneously executing in parallel. the restriction operator "new" limits the scope of its parameters to the current sequential expression. In the above example, the action prefix "a" is restricted (or bound) to the scope of "P". Names that appear in input actions are also bound. Finally the replication operator "!" causes the process in its scope to appear in endless copies in parallel. These expression constructs allow a structural comparison between two processes to detect structural congruence.

Definition ([10, Def 9.7]). **Structural Congruence** *Two process expressions P and Q in the* π-calculus *are structurally congruent, written $P \equiv Q$, if we can transform one into the other by using the following equations (in either direction):*

(1) Change of bound names (alpha-conversion)

* This work was funded by the DFG Cluster of Excellence on Ultra-high Speed Information and Communication (UMIC), German Research Foundation grant DFG EXC 89. Many thanks to AOR Priv.-Doz. Dr. Thomas Noll and Prof. Dr.-Ing. Stefan Kowalewski for their continuous support and supervision during this work.

(2) reordering of terms in a summation
(3) $P \mid 0 \equiv P,\ P \mid Q \equiv Q \mid P,\ P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
(4) $\mathrm{new}x(P \mid Q) \equiv P \mid \mathrm{new}xQ$ *if* $x \notin \mathrm{fn}(P)$, $\mathrm{new}x0 \equiv 0$, $\mathrm{new}xyP \equiv \mathrm{new}yxP$
(5) $!P \equiv P \mid !P$

Parrow [11] defines two additional structural congruence rules to cope with restrictions, that is:

- $(\mathrm{new}x)(\mathrm{if}u = v)P \equiv (\mathrm{if}u = v)(\mathrm{new}x)P$ if $x \neq u$ and $x \neq v$.
- $(\mathrm{new}x)(\mathrm{if}u \neq v)P \equiv (\mathrm{if}u \neq v)(\mathrm{new}x)P$ if $x \neq u$ and $x \neq v$.

We will be using these structural congruence rules later in our work for reducing sequences of timed actions (Lemma 38). In this work we consider that both [10] and [11] depict what we call *classic π-calculus*.

Input actions are also called positive action prefixes, while output actions are negative. Each pair of positive and negative actions makes a redex if they have the same name. A redex allows these action prefix pairs to synchronize their execution (or to react). If these action prefixes appeared with parameters then their reaction means a transfer operation from the sender to the receiver. In this sense, the name of the action prefix represents the name of a communication channel between the processes that contain the redex parts. The action prefix τ in the classic *π-calculus* represents the occurrence of an redex internally. It obviously is unidirectional (neither positive nor negative) and has no parameters. All action prefixes of classic *π-calculus* can happen any time once they are enabled or unguarded. Figure 1 lists the reaction rules in *π-calculus*.

$$\mathrm{TAU}: \quad \tau.P + M \rightarrow P$$

$$\mathrm{REACT}: \quad (x(y).P + M) \mid (\overline{x}\langle z\rangle.Q + n) \rightarrow \{z/y\}\, P \mid Q$$

$$\mathrm{PAR}: \quad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

$$\mathrm{RES}: \quad \frac{P \rightarrow P'}{\mathrm{new}xP \rightarrow \mathrm{new}xP'}$$

$$\mathrm{STRUCT}: \quad \frac{P \rightarrow P'}{Q \rightarrow Q'} \qquad \textit{if}\, P \equiv Q \ \textit{and}\ P' \equiv Q'$$

Fig. 1. Reaction Rules

The execution of action prefixes can be seen as a transition if we look at *π-calculus* as a labeled transition system. To simplify the concept we consider these transitions without parameters. This allows us to look at the transition rules of CCS[1]. Figure 2 lists all transition rules from which inference trees are built for conclusions about system properties. This is generally done using induction on the depth on inference.

By integrating the notion of alpha-conversion due to message transfer between processes, the transition rules evolve in commitment rules. A commitment is simply a transition with parameters. However, car must be taken to the residues

[1] Calculus of Communicating Systems, the forefather of *π-calculus*.

$$\text{SUM}_t: \quad M + \alpha.P + N \xrightarrow{\alpha} P \qquad\qquad \text{REACT}_t: \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{L-PAR}_t: \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad\qquad \text{R-PAR}_t: \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$\text{RES}_t: \frac{P \xrightarrow{\alpha} P'}{\text{new}x P \xrightarrow{\alpha} \text{new}x P'} \; {\scriptstyle if\, \alpha \notin \{x, \overline{x}\}} \qquad\qquad \text{IDENT}_t: \frac{\{\boldsymbol{b}/\boldsymbol{a}\}\, P_A \xrightarrow{\alpha} P'}{A\langle \boldsymbol{b} \rangle \xrightarrow{\alpha} P'} \; {\scriptstyle if\, A(a) \overset{\text{def}}{=} P_A}$$

Fig. 2. Transition Rules

of transitions in π-*calculus*. In the case of output actions, the residue after the execution is concretion agent, while in the case of input actions it is an abstraction agent. Milner handled the issue of input actions that leave abstraction behind with special care, because the receiving a name over a channel causes the name in the input action prefix, say x, to be alpha-converted to received new name. The abstraction that results from this alpha conversion can have different behavior depending on the received name and the surrounding processes that could react to the resulting agent. However, if the agent is x-forgetful [10, Definition 10.7] it is immune to this behavioral *evolution* which makes it invariant to reaction because x does not appear later in this process. All commitment rules[2] are listed in Figure 3.

$$\text{SUM}_c: \quad M + \alpha.P + N \xrightarrow{\alpha} P \qquad\qquad \text{L-REACT}_c: \frac{P \xrightarrow{x} F \quad Q \xrightarrow{\overline{x}} C}{P \mid Q \xrightarrow{\tau} F@C}$$

$$\text{R-REACT}_c: \frac{P \xrightarrow{\overline{x}} F \quad Q \xrightarrow{x} C}{P \mid Q \xrightarrow{\tau} F@C} \qquad\qquad \text{L-PAR}_c: \frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}$$

$$\text{R-PAR}_c: \frac{Q \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid P} \qquad\qquad \text{RES}_c: \frac{P \xrightarrow{\alpha} A}{\text{new}x P \xrightarrow{\alpha} \text{new}x A} \; {\scriptstyle if\, \alpha \notin \{x, \overline{x}\}}$$

$$\text{REC(c)}: \frac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A}$$

Fig. 3. Commitment Rules

Behavioral equivalence has two classes in π-*calculus*; strong & weak. Weak simulation behaviorally compares two differently implemented processes ignoring all occurrences of τ. All transitions (or commitments) that involve τ are called experiments and are referred to by "$\Rightarrow$". On the other hand, strong simulation does not have this relaxed property and requires exact match of the behavior of both studied processes. If both processes simulate each other then we call the relation a bisimulation. Structural congruence itself is considered a bisimulation.

[2] $A, B, \ldots$ stand for agents, $C, D, \ldots$ stand for concretion, $F, G, \ldots$ stand for abstractions and @ is the application. See [10, Definition 12.1 and 12.2]

2 Related work

We based our requirement analysis on the work of Baeten and Middelburg [1] where timed process algebras were nicely declared and sorted in classes, and the characteristics of each class were formally represented by axioms. We recognized four variations of these algebras from which we concluded the contribution that we need to provide to *π-calculus* in order to make it time-enabled. These variations are the absolute and relative timing schemes ($\mathrm{ACP}^{\mathrm{sat}}$ and $\mathrm{ACP}^{\mathrm{srt}}$) in their real and discrete formats ($\mathrm{ACP}^{\mathrm{dat}}$ and $\mathrm{ACP}^{\mathrm{drt}}$, respectively). In $\mathrm{ACP}^{\mathrm{sat}}$ three basic time operators were introduced; the absolute delay operator σ_{abs}, the absolute time-out operator ν_{abs} and the absolute initialization operator $\bar{\nu}_{abs}$. In [1, pp. 632], Baeten and Middelburg showed that ν_{abs} and $\bar{\nu}_{abs}$ can be reduced to σ_{abs} alone using sequential and parallel composition. We base our timed π-calculus extension on this result, which is especially convenient because *π-calculus* natively supports sequential and parallel composition.

Hennessy and Regan [6] developed a CCS based process algebra that is extended with a special action to represent the passage of one clock-tick of time. They specify operational semantics on the clock-tick level, and present their Temporal Process Logic (TPL) with a rigorous set of convergence and alternative characterization rules. However, this algebra does not address the case of uninterrupted sequences of the clock-tick operator, and hence does not specify particular points of time in the future at which specified actions can take place.

A recent contribution to timed *π-calculus* is the work of Jin et al. [7]. In the extended syntax there are new timed operators for expressing the passage of time and for modeling choice upon threshold crossing. Due to modeling time passing on the clock-tick level, all transition rules of *π-calculus* were revisited to provide delayed versions of each rule to represent how processes behave when time passes. This multiplies the total operational semantics of the calculus. We perceive a superfluity in introducing the threshold operator in the presence of the original choice operator "+" in *π-calculus*.

The approach in [12] provides a type system for specifying locations which restrict services that agents can use during communication. TDπ-calculus combines the execution of actions with the passage of time in dedicated time operators. Message exchange occurs at defined locations over channels at these locations if the timer was not exceeded, then the process proceeds as declared in the operator. If the time-out value is exceeded, the process takes an alternative execution path. Our calculus can provide the same timed behavior using the timer and choice operators. The timed semantics of both approaches are equivalent. However, our calculus retains the advantage of separating the passage of time from the execution of actions, and thus provides more flexibility in specifying system behavior, especially when no timing is required. In addition, our calculus allows state transitions based on pure time events.

Timed spi-calculus [5] approaches timing-out in a manner similar to [12]. The timed operational semantic is defined as a reduction relation on states with global time variable and a binder on all free names in these states. The operator allows the definition of a set of correspondences that can be terminated in the particular run, and also defines the process that remains to be executed. This reduction rule allows only process replication and output actions to survive a period (or epoch) to the next one. All other syntactic forms expire with a clock-tick and degenerate to the null-process. Timed spi-calculus combines the notion of time passing and action execution and conveniently tailors the behavior of particular syntactic components of the calculus for modeling security and cryptographic issues, e.g. the asymmetrical timed behavior of input and output actions. This makes the calculus, however, inflexible in treating issues outside that domain.

Berger and Yoshida [2] define types on channels which range in one dimension over being linear or affine and whether they are replicated or not. The other dimension depends on whether these names are input or output. From this type classification, they define constraints on names. The type system is then used to define message-loss and timers on process and network level. To model message-loss correctly, they solve the nondeterminism problem by using probabilistic automata. They proceed then by defining the impact of this nondeterminism on bisimulation by introducing timed approximate bisimulation. Nonetheless the analogy between [2] and [7] with respect to timed behavior becomes clear by considering their timeout and *timestepper* arrangements.

In the work of Lee and Žic [8] the time-out operator $P \rhd^t Q$ forces a process P to proceed as Q at time t if no enabled action of P was fired before t. Similar to previously mentioned approaches, a delay operator is defined. This operator takes place in the transition system of processes. Lee and Žic differentiate internal or invisible actions from those visible to the external observer by calling them uncontrollable and controllable respectively. In this work time is not allowed to pass during action execution and nondeterministic choices are only made by ordinary action prefixes. This means that the passage of time does not obscure the execution of an enabled action in a process. As in [7] the operational semantics explode due to different time passing properties of i/o actions and τ and their effect on combinatorial operators such as summation and parallel composition.

The concept of timed hybrid automata as proposed by Lynch et al. [9] defines trajectories as the evolution of a collection of variables over an interval of time, and they are the set of functions that map a closed interval to the set of valuation functions. Hybrid automata (HA) are comparable if they have the same external interface (i.e., external variables and actions). They consider a simulation relation between different HAs if one implements the other in the sense of inclusion of sets of execution traces. Then they declare the composition and hiding operations on HAs, which in turn, respect simulation. The distinction between input and output variables enables the definition of hybrid I/O automata (HIOA). The receptiveness property defined on HIOAs allows the time to always pass for any sequence of input actions. The presented structures in this work are interesting because they correspond to native capabilities in π-*calculus*. However, hybrid automata do not posses the hierarchical structuring capabilities and the dynamic channel setup of π-*calculus*.

In all the approaches we have presented, the designers tend to enrich π-*calculus* with time features. Although most approaches tend to use the same set of tools to model time properties, we see a tendency in each one of them to initiate special rules of time passage and their effects on processes in order to suite particular applications. In our approach we try to introduce the notion of time into π-*calculus* in the most native way possible for two reasons. We first would like to avoid making the calculus too complex so that it stays appealing for people of other domains who would like to use a formal model for representing timed communicating processes. The second reason is that by keeping our timed approach as generic as possible, we create a calculus that targets the widest possible domain of applications.

3 Real-Time in π^τ-*Calculus*

Although we use a universal notion of time in our model, we recognize the need to two types of clocks in modeling timed processes; a global clock which is visible to all processes and private clocks for those processes which need to execute internal timed actions that do not relate to actions in other processes. Processes that use the global clock to synchronize their actions make their timed behavior visible to the outside world. Taking the concept of universality of time into account, we can say that the formal presence of the global clock is not always explicitly needed. We need, however, the concept of *visible* timed actions to be able to check for behavioral equivalence. We refer to our extended version of the calculus by π^τ-*calculus*. We will start by defining the clock process, which is universal regardless of being a global of private clock, and then we will discuss the various properties of this definition. In the following all time clocks, variables and values range over $\mathbb{R}^{\geq 0}$.

3.1 Modeling clocks in π^τ-*Calculus*

We assume that each process P that is capable of performing internal (*unobservable*) timed actions contains a subprocess $C(c)$ that represents its local (or private) clock. To achieve absolute and relative timing and to be able to save the current time, $C(c)$ needs to provide three operations:

$$\begin{aligned} C(c) ::= &\ (\text{if } \dot{t} = \alpha + \beta)\ c(\alpha, \beta) \cdot C\langle c \rangle + \\ &\ (\text{if } \dot{t} = \alpha)\ c(\alpha) \cdot C\langle c \rangle + \\ &\ \overline{c}\langle \dot{t} \rangle \cdot C\langle c \rangle \end{aligned} \tag{1}$$

The clock subprocess $C(c)$ will react through its action prefixes $c(\alpha, \beta)$ and $\overline{c}\langle \dot{t} \rangle$ with any other subprocess of P, say P_r, when it tries to execute $\overline{c}\langle q, p \rangle$ (if the current absolute time $\dot{t}$ is $q + p$) and $c(s)$ respectively. If the current time is less than $q + p$, $\overline{c}\langle q, p \rangle$ will block until the time reaches $q + p$. If $\dot{t}$, however, is greater than $q + p$, $\overline{c}\langle q, p \rangle$ will deadlock forever which means that the reaction will not complete. Time values act here as names and the matching between the parameters of action c and the current time occurs in a similar fashion to the classic name matching by Parrow. However, names that act as time variables or values need to be valuated to their respective values in $\mathbb{R}^{\geq 0}$ so that comparisons against $\dot{t}$ and other values and variables can be conducted. We define this function formally as follows:

Definition 31. *The "valuation" function $v : A \rightarrow \mathbb{R}^{\geq 0}$ delivers the time values of names in* π^τ-calculus *that act as time variables or values.*

The second operation $C(c)$ offers, is analog to the first one; $c(\alpha)$ accepts synchronization requests with one absolute parameter instead of two parameters of which the second is interpreted relative to the first. The same semantics apply here; if the current time $\dot{t}$ is equal to the passed value α the synchronization proceeds, otherwise the complementary action $\overline{c}\langle p\rangle$ will block or deadlock as explained above. This *absolute* time operator can of course be represented using the *relative* operator by passing zero in one of the parameters of $\overline{c}\langle q, p\rangle$. We provide it here for convention.

Finally, the clock process sends its current time by enabling the action $\overline{c}\langle\dot{t}\rangle$ to react with any subprocess P_r that tries to know the current time by executing $c(s)$. This action obviously does not deadlock because no conditional matching takes place.

In the case of a process P that possesses a private clock, the process definition looks like the following: $P ::= new\ c(P_r \mid C(c))$. Obviously, without the restriction of c P_r and $C(c)$ could proceed using the $L\text{-}PAR_c$ or $R\text{-}PAR_c$ commitment rules, rather than $L\text{-}REACT_c$ or $R\text{-}REACT_c$. This also prevents external processes from $REACT$ing with the private clock $C(c)$. On the other hand, we should not restrict the action prefix d of the global clock process $C(d)$, so that it is publicly available to all other processes.

A design principle of our calculus becomes clear at this stage. Similar to the approach in [1] each clock increases its value internally and reacts to timed requests of other processes accordingly. By abstracting the passage of time as an internal activity of clock process, we save the users of our calculus the formalities that arise from explicitly defining it (e.g. $\triangleright^t$ in [2]). This abstraction results in a considerable reduction in the overhead of time specification and helps us achieve our design objectives.

3.2 Internal (or unobservable) timers

The inference trees for reactions between a clock process and another process are listed in Figure 4. We assume a process that consists of a subprocess P_r and an internal clock subprocess $C(c)$. We also assume that P_r has the form $P_r ::= Q + \overline{c}\langle q, p\rangle \cdot P'_r + R$. We use the notational convention to mark these tau actions by their time parameters to distinguish them from the original silent action τ. Unlike the proposed handling in [1] we do not distinguish absolute and relative timing in two separate calculi. We only consider these two concepts superficially by taking zero as the time reference and omitting it notationally in the former case (e.g. τ^p) and by taking a non-zero value or identifier in the latter. The implementation of $\tau_s^{q,p}$ is a sequential composition of $\tau^{q,p}$ and τ_s. The same applies for τ_s^p. This concept works because we assume that timed actions do not consume execution time, and because we assume higher execution priority for them over non-timed action prefixes. This notion becomes clear in Section 3.7.

The timed τ action prefix can thus be said to execute at specified points of time consuming itself no execution time, but allowing other actions in the sequential process expression to align themselves to it and hence acquire time properties. The timed τ remains invisible (or silent) but will be decorated with time information (relative or absolute and with or without bound names) to specify its occurrence point of time and to specify its impact on other timed actions ans variables. This is useful when we discuss timed bisimulation (see Example 311). Figure 5 shows the relation between the superscript and subscript of timed τ.

$$\tau^{q,p}\left\{\dfrac{\dfrac{\dfrac{\dfrac{}{M + c(\alpha,\beta)\cdot C(c) + N \overset{c(\alpha,\beta)}{\rightarrow} C(c)}\,\mathrm{SUM_c}}{C(c) \overset{c(\alpha,\beta)}{\rightarrow} C(c)}\,\mathrm{IDENT_t} \qquad \dfrac{\dfrac{}{Q + \overline{c}\langle q,p\rangle \cdot P'_r + R \overset{\overline{c}\langle q,p\rangle}{\rightarrow} P'_r}\,\mathrm{SUM_c}}{P_r \overset{\overline{c}\langle q,p\rangle}{\rightarrow} P'_r}\,\mathrm{IDENT_t}}{C(c) \mid P_r \overset{\tau^{q,p}}{\rightarrow} C(c) \mid P'_r}\,\mathrm{L\text{-}REACT_c}}{new\ c\ (C(c) \mid P_r) \overset{\tau^{q,p}}{\rightarrow} new\ c\ (C(c) \mid P'_r)}\,\mathrm{RES_c}\right.$$

$$\tau^{p}\left\{\dfrac{\dfrac{\dfrac{\dfrac{}{M + c(\alpha)\cdot C(c) + N \overset{c(\alpha)}{\rightarrow} C(c)}\,\mathrm{SUM_c}}{C(c) \overset{c(\alpha)}{\rightarrow} C(c)}\,\mathrm{IDENT_t} \qquad \dfrac{\dfrac{}{Q + \overline{c}\langle p\rangle \cdot P'_r + R \overset{\overline{c}\langle p\rangle}{\rightarrow} P'_r}\,\mathrm{SUM_c}}{P_r \overset{\overline{c}\langle p\rangle}{\rightarrow} P'_r}\,\mathrm{IDENT_t}}{C(c) \mid P_r \overset{\tau^{p}}{\rightarrow} C(c) \mid P'_r}\,\mathrm{L\text{-}REACT_c}}{new\ c\ (C(c) \mid P_r) \overset{\tau^{p}}{\rightarrow} new\ c\ (C(c) \mid P'_r)}\,\mathrm{RES_c}\right.$$

$$\tau_{s}\left\{\dfrac{\dfrac{\dfrac{\dfrac{}{M + \overline{c}\langle \dot{t}\rangle\cdot C(c) + N \overset{\overline{c}\langle \dot{t}\rangle}{\rightarrow} C(c)}\,\mathrm{SUM_c}}{C(c) \overset{\overline{c}\langle \dot{t}\rangle}{\rightarrow} C(c)}\,\mathrm{IDENT_t} \qquad \dfrac{\dfrac{}{Q + c(p) \cdot P'_r + R \overset{c(s)}{\rightarrow} P'_r}\,\mathrm{SUM_c}}{P_r \overset{c(s)}{\rightarrow} P'_r}\,\mathrm{IDENT_t}}{C(c) \mid P_r \overset{\tau_s}{\rightarrow} C(c) \mid P'_r}\,\mathrm{R\text{-}REACT_c}}{new\ c\ (C(c) \mid P_r) \overset{\tau_s}{\rightarrow} new\ c\ (C(c) \mid P'_r)}\,\mathrm{RES_c}\right.$$

Fig. 4. Inference Trees for Timed Tau

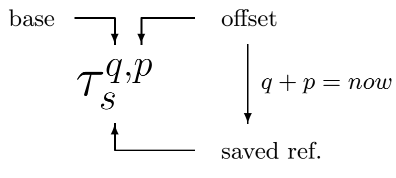

Fig. 5. Timed τ

The optional subscript specifies an identifier that holds the value of the point in time at which the execution of τ has occurred. If the superscript was omitted, we call this operator *passive* and the names in the subscript will be alpha-converted to the value of the current time and τ will be directly executed when it becomes enabled (or unguarded). If more than one name were present in the subscript, e.g. $\tau_{a,b}$, this will be equivalent to a series of passive timed tau action prefixes for each name. The expression $\tau_a\tau_b$ causes both a and b to be alpha-converted to the same time value because (as explained above) we assume that timed action prefixes do not consume execution time on their own, and because all timed action prefixes have higher priority over the rest of action prefixes in π^τ*-calculus.*

The superscript of τ specifies the execution point of time and its reference. In this case we call the time operator *active*. The reference can be a time value or an identifier that holds a value of that sort. The component p of the constraint specifies the point in time at which timed τ is executed in reference to q. Again, p can be any time value. In addition, q and p can be arithmetic expressions that specify single or multiple points in time at which τ can be executed. The actual execution point of time is calculated by adding the values of p and q. An active time operator with two parameters is called *relative*, while operators that reference the starting point of time of the system and thus have one parameter are called *absolute*. Examples are found in Figure 6. We have to keep in mind, that time identifiers that are not explicitly initialized in a process parameter or in the subscript of τ are automatically initialized to zero.

Absolute Timing	
$\tau^3 \cdot a$	Action a executes once after 3 time units relative to 0.
$\tau_p^{p+3.1} \cdot a$	Action a repeats executes at 3.1 time units relative to 0 and saves the current time value in p. If this expressions appears in a recursive expression, it is capable of executing repeatedly each 3.1 time units.
Relative Timing	
$\tau^{3.3,1.2} \cdot a$	Action a executes once at time 4.5.
$\tau_s \cdot \tau^{s,1.2} \cdot a$	Action a is invoked once after 1.2 time units relative to s where s is initialized in a previous τ.
$\tau_s^{s,2} \cdot a$	Action a executes after two time units from the current value of s and saves the current execution time in s. If s is not previously initialized, it is valuated to zero. If this expression is invoked recursively, it repeats each two time units.

Fig. 6. Examples for timed τ

3.3 External (or observable) timers

As we previously indicated, if a process P synchronizes its timed actions with the global clock process $C(d)$ then these actions must be visible to the outside world. This means that the external observer does not see timed τ actions here but the time action prefixes $\overline{d}\langle q,p\rangle$, $\overline{d}\langle p\rangle$ and $d(s)$ on the part of the client processes in addition to $d(q,p)$, $d(p)$ and $\overline{d}\langle\dot{t}\rangle$ of the global clock. As with timed τ, we use the notational convention $d_s^{q,p}$ to stand for these actions on the part of the client processes. The *bidirectional* case is easy to identify by means of the superscript and subscript both appearing next to the timer's name. In this case, the same operator obviously stands simultaneously for input and output actions (active and passive).

Since the passive timed action prefix as any other input action is binding over its parameters, it is important to notice that $\mathrm{bn}(d_s^{q,p}) = \{s\}$ whereas $\mathrm{fn}(d_s^{q,p}) = \{d, q, p\}$.

3.4 Time constructs

As suggested in [1], we need particular time operators and constants to be able to describe time properties of concurrent processes. In π^τ-*calculus* we obtain these time properties by injecting $c_s^{q,p}$ or $d_s^{q,p}$ in sequential process expressions. In the following sections we use the internal timed action $c_s^{q,p}$ to present the operators of π^τ-*calculus*. This, however, applies analogously to the observable timed action $d_s^{q,p}$. The semantic properties, however, in regard to bisimulation differ largely. We discuss these properties later in this report.

Constants In the classic π-*calculus*, the null process "0" represents terminated processes. In addition, it is used to represent deadlocked processes [10, Exercise 5.26]. We use this concept as is without further modification.

Delay The delay operator $\sigma^p_{abs}(a)$ in $\mathrm{BPA}^{\mathrm{sat}}$ as proposed by [1] can be represented in π^τ-*calculus* as follows:

$$c^{0,p} \cdot a$$

$c^{0,p}$ (or c^p) consumes itself no execution time, and terminates at time p. when $c^{0,p}$ terminates, "a" becomes unguarded, which makes it available for execution starting from time p.

Absolute Time-out The operator for absolute time-out in $\mathrm{BPA}^{\mathrm{sat}}$ is denoted as $\nu^p_{abs}(a)$. In π^τ-*calculus* we substitute this operator by writing:

$$c^{0,p} \cdot 0 + a$$

Since "a" is unguarded, it can start to execute anytime up till (but not including) p. If "a" does not execute $c^{0,p}$ will fire at time p and the process will take the alternative path. In this example we have attached the empty sum 0 (or process null) to c suggesting to terminate the process and force possible complementing action prefixes (or redexes) of "a" to deadlock. However, this must not be the case. The system designer can choose to point the system to another state instead of deadlocking, which could - in this case - be an advantage over $\mathrm{BPA}^{\mathrm{sat}}$ where deadlocking is a must if "a" fails to start before p.

Absolute initialization The operator $\overline{\nu}^{p}_{abs}(a)$ indicates that "a" can start to execute from point p and beyond. The equivalent expression in π^τ*-calculus* is identical to the absolute delay above:

$$c^{0,p} \cdot a$$

The similarity between the absolute initialization and the delay technique (section 3.4) in our calculus might be easier to understand if we consider that using the axioms of BPA$^{\text{sat}}$ one could express all ν_{abs} and $\overline{\nu}_{abs}$ processes by means of the absolute delay, alternative composition and sequential composition. We use this result to keep our design minimal and limit our time operators to only $c_s^{q,p}$ (or $d_s^{q,p}$ according to the scope of the clock).

Time constraints A common way of managing constraints in flavored π*-calculus* [4,3] is the definition of a store in which statements entail particular agents or not. These stores can be manipulated with *tell* and *ask* operations to add statements to the store and to probe whether the store contains an entailing statement respectively. Agents that are prefixed with entailing statements are enabled.

We have simple time constraints in π^τ*-calculus*. If the current local time $\dot{t} = q + p$ then the threshold is satisfied and the timer will fire. In addition, we can extend the classic conditional prefixing of sequential process expressions as suggested by Parrow [11]. The standard *Match* and *Mismatch* constructs use only the equality or inequality notions because the original application was on names only. However, in our extended calculus we have time variables and values in addition to names. Therefore, we can safely extend these constructs to be able to do simple arithmetic comparisons on these values.

Definition 32. Time constraints *let* $a, b, \ldots \in A^\tau$ *where* A^τ *is the set of all variable names in* π^τ-calculus*,* $x \in \mathbb{R}^{\geq 0}$*, and let* $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$*. Consider that* $\delta = a - b \mid a - x \mid x - a \mid a$*. A time constraint is an expression* C *such as* $C = C_1 C_2 \ldots C_n$ *where* $C_i = (\text{if}^3\ \delta \bowtie 0)$*.* C *holds in a valuation* v *(see Definition 31) if all* C_i *hold (or evaluate to* true*) w.r.t.* v*.*

If a timed action (c or d) is prefixed with a constraint, then the execution of this timed action will be restricted so that it only becomes enabled if the constraint holds. You can see in Figure 7 how a more complex timed behavior than those depicted in Figure 6 can be specified for some process P using this constraint system:

$$P ::= (\text{if } s < 2)\ c_s^{s,0.5} \cdot a \cdot P + (\text{if } s \geq 2)\ c_s^{s,1} \cdot b \cdot P$$

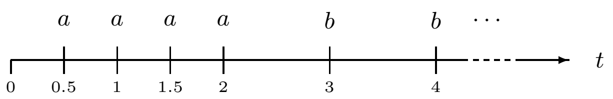

Fig. 7. Time constraints using conditional prefixing of time variables

[3] We often drop "if" for simplicity

We define time constraints in the context of modifying the behavior of timed action prefixes. Each timed action prefix defines one or more points in time where it can fire. Constraints restrict this set to a subset of possible execution opportunities. Therefore, we prefix only active timed actions with constraints, while passive ones execute unrestricted. We will address this point formally in Section 3.6 when we formalize structural congruence of timed actions.

In the following, we will be presenting examples about how to use timers to model sensor networks and the NTP protocol.

Example 33. Sensor networks Let's consider the system Sys that consists of two processes. The sensor node generates a reading each 3 time units and sends it to the receiver over channel ch, while the receiver replies immediately over the same channel with an acknowledgment or sends a reset message if the sensor fails to produce a reading after 4 time units. Upon reception of a reset message, the sensor resets its timer and starts over. A possible solution using absolute timing is proposed in the following:

$$Sys ::= Sen\langle 3\rangle \mid Rec\langle 4\rangle \mid C\langle d\rangle$$

$$\begin{aligned} Sen(p) &::= ch(reset) \cdot d_t \cdot Sen\langle p\rangle + d_t^{t+p} \cdot \overline{ch}\langle data\rangle \cdot Sen'\langle p\rangle \\ Sen'(p) &::= ch(reset) \cdot d_t \cdot Sen\langle p\rangle + ch(ack) \cdot Sen\langle p\rangle \end{aligned}$$

$$\begin{aligned} Rec(q) &::= d_s^{s+q} \cdot \overline{ch}\langle reset\rangle \cdot Rec\langle q\rangle + ch(data) \cdot d_s \cdot Rec'\langle q\rangle \\ Rec'(q) &::= d_s^{s+q} \cdot \overline{ch}\langle reset\rangle \cdot Rec\langle q\rangle + \overline{ch}\langle ack\rangle \cdot Rec\langle q\rangle \end{aligned}$$

Another solution using relative timing is proposed in the following:

$$Sys ::= Sen\langle 0, 3\rangle \mid Rec\langle 0, 4\rangle \mid C\langle d\rangle$$

$$\begin{aligned} Sen(t, p) &::= ch(reset) \cdot d_t \cdot Sen\langle t, p\rangle + d^{t,p} \cdot \overline{ch}\langle data\rangle \cdot Sen'\langle t + p, p\rangle \\ Sen'(t, p) &::= ch(reset) \cdot d_t \cdot Sen\langle t, p\rangle + ch(ack) \cdot Sen\langle t, p\rangle \end{aligned}$$

$$\begin{aligned} Rec(s, q) &::= d^{s,q} \cdot \overline{ch}\langle reset\rangle \cdot Rec\langle s + q, q\rangle + ch(data) \cdot d_s \cdot Rec'\langle s, q\rangle \\ Rec'(s, q) &::= d^{s,q} \cdot \overline{ch}\langle reset\rangle \cdot Rec\langle s + q, q\rangle + \overline{ch}\langle ack\rangle \cdot Rec\langle s, q\rangle \end{aligned}$$

In any of the above solutions, if message loss occurs on ch, the protocol will continue to execute and will never deadlock. We assumed that both processes synchronize with a global clock $C(d)$. ■

Example 34. Network Time Protocol (NTP) The NTP protocol is used to synchronize the clock of a client with a time server. We consider a simple system of two processes; a server and a client. The client makes one synchronization each 24 hours. Upon reception of the client's request, the server registers the time stamp of its arrival, then it computes the response for the client and registers the time at that point. These two time stamps are then sent to the client. The client receives the server's response and registers its time stamp. The client then calculates the round trip delay $\delta = (t_3 - t_0) - (t_2 - t_1)$ and the offset $\theta = \frac{(t_1-t_0)-(t_2-t_3)}{2}$ in and goes back to normal operation.

$$Sys ::= Srv \mid Cln$$

$$Srv ::= \text{new}c'(c(req) \cdot c'_{t_1} \cdot \tau \cdot c'_{t_2} \cdot \overline{ch}\langle rsp, t_1, t_2\rangle \cdot Srv \mid C\langle c'\rangle)$$

$$Cln ::= \text{new}c''(c''^{s,24h}_{s} \cdot Cln_s \mid C\langle c''\rangle)$$
$$Cln_s ::= \text{new}c''(\overline{ch}\langle req\rangle \cdot c''_{t_0} \cdot ch(rsp, t_1, t_2) \cdot c''_{t_3} \cdot \tau \cdot Cln \mid C\langle c''\rangle)$$

We assumed that the server synchronizes with its private clock over channel c', and that the client synchronizes with its private clock over channel c''. ■

3.5 Signature of Real-Time π^τ-*Calculus*

We are now ready to conclude the signature of real-time π^τ-*calculus*. We consider here only our contribution to π-*calculus* without repeating the classic constructs of it. In real-time π^τ-*Calculus*, we use the following symbology: $\dot{t}$ is the current time of the reference clock and it ranges over $\mathbb{R}^{\geq 0}$. $d^{q,p}$, d^p, d_s, $d_s^{q,p}$, d_s^p, $\tau^{q,p}$, τ^p, τ_s, $\tau_s^{q,p}$ and τ_s^p are the absolute, relative and passive visible and invisible time action prefixes (or timers) whose parameters q, p and s range over $\mathbb{R}^{\geq 0}$. We use π^τ to reference the set of all timed action prefixes which is a subset of π. If available, superscript parameters define when the timer should fire according to the reference clock, and subscript parameters are alpha-converted on execution time to hold the current value of $\dot{t}$. The same applies to $c_s^{q,p}$, c_s^p, c_s, $c_s^{q,p}$ and c_s^p that reference internal real-time clocks of their processes. $\mathcal{C}^\tau$ is the set of all constraints in which $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$ is the comparison operator[4].

3.6 Structural congruence of timed actions

Our timer system imposes little change to the signature of π-*calculus* and makes the consequent proofs simple. In addition to the list of structural congruence rules of π-*calculus* we define two rules to incorporate the new time operators $\tau_s^{q,p}$ and $d_s^{q,p}$, but before we can do that, we must setup some basic definitions.

Definition 35. Mapping function Ω *Let $C \in \mathcal{C}^\tau$ where $\mathcal{C}^\tau$ is the set of all constraints as shown in Definition 32 and let $\eta \in \pi^\tau$ where π^τ is the set of all timed action prefixes (a subset of π). We define the mapping function $\Omega : (\dot{\mathbb{T}}, \mathcal{C}^\tau, \pi^\tau) \to \{\mathbb{R}^{\geq 0}\}$ to be the set of execution times (or opportunities) that η can encounter after being restricted with C, where $\dot{t}$ can only range over $\dot{\mathbb{T}} \subseteq \mathbb{R}^{\geq 0}$.*

Of course, if C is empty (represented with ε) then $\Omega(\mathbb{R}^{\geq 0}, \varepsilon, \eta)$ is the set of all execution time points of η assuming that $\dot{t}$ cannot be negative. There can be cases where $\Omega(\dot{\mathbb{T}}, C, \eta) = \phi$. In this case the constraint completely blocks the execution of its action prefix. Example 36 shows some expressions that are evaluated with Ω.

[4] instead of only = and ≠ in classic π-*calculus*

Example 36. Mapping Function Ω In this example, $A(s,t)$ is a sequential composition of $A_1(s)$ and $A_2(t)$. Figure 8 shows the evaluation using the Ω function on each of those timed processes.

$$A_1(t) := (\text{if } t > 2)c^t. \tag{2}$$
$$A_2(s) := (\text{if } s < 5)c_s^{s,0.5}. \tag{3}$$
$$A(s,t) := (\text{if } t > 2)c^t.(\text{if } s < 5)c_s^{s,0.5}. \tag{4}$$

Evaluating Ω on $A_1(t)$ (for arbitrary t) produces $]2,\infty[$, while evaluating Ω on $A_2(s)$ yields $[0.5, 5.5[$. The composition of both processes evaluates to $]2.5, 5.5[$.

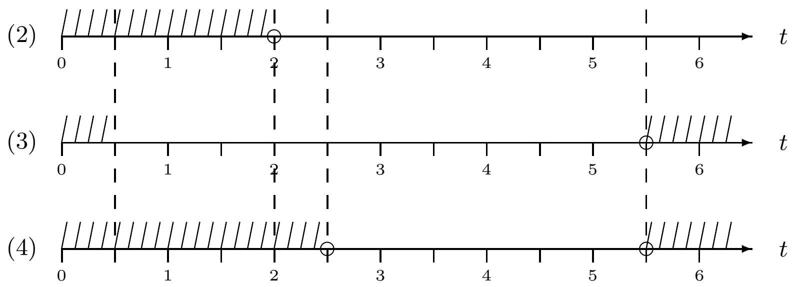

Fig. 8. Evaluating the Ω function (Example 36)

If a timed action is not prefixed with another timed action, the Ω function takes $\mathbb{R}^{\geq 0}$ as the first parameter, which implies that the starting execution window is unrestricted before evaluating. In a sequence, on the other hand, the evaluation of Ω on a timed action is taken as the first parameter of Ω on the next timed action, which reduces the available execution window for that action before applying Ω on it. The following set of Ω applications explains the sequential effect of its evaluation on sequences:

$$\begin{aligned} \Omega_{A_1} &= \Omega(\mathbb{R}^{\geq 0}, t > 2,\ 0t) &&=]2, \infty[\\ \Omega_{A_2} &= \Omega(\mathbb{R}^{\geq 0}, s < 5,\ s0.5s) &&= [0.5, 5.5[\\ \Omega_{A} &= \Omega(\Omega_{A_1}, s < 5,\ s0.5s) &&=]2.5, 5.5[\end{aligned}$$

The concatenation of timed actions reduces the resulting execution time of them in total. ■

Definition 37. Non-blocking sequence *Let η be either τ or d, $C_i \in \mathcal{C}^\tau$, and let $\prod_{i \in I} C_i \cdot \eta_i^{q_i,p_i}$ be a sequence of timed actions and their constraints, where $I = \{1, 2, \cdots, n\}$. We consider this sequence to be non-blocking if $\forall i \in I \backslash \{n\}; q_i + p_i \leq q_{i+1} + p_{i+1}$ and $\forall i \in I; \Omega(\dot{\mathbb{T}}_i, C_i, \eta_i) \neq \phi$ where $\dot{\mathbb{T}}_i = \Omega(\dot{\mathbb{T}}_{i-1}, C_{i-1}, \eta_{i-1})$ and $\dot{\mathbb{T}}_1 = \mathbb{R}^{\geq 0}$.*

This is an intuitive but important property which allows us to construct useful sequences of timed actions. It says that a concatenation of timed actions and their constraints should guarantee the continued execution of all its actions. This can be achieved if no action prefix finishes after the start of the next one and if no constraint completely blocks the execution of its timed action prefix. It is also important to keep in mind that the evaluation of Ω function for one $C_i\epsilon_i$ pair proceeds dependently on the previous evaluation of the preceding pair. We proceed now to the definition of timed structural congruence.

Lemma 38. Reduction of non-blocking sequences *Let η be either τ or d. Any non-blocking sequence $\prod_{i\in I} C_i\eta_i$, where $I = \{1, 2, \cdots, n\}$ and η is an arbitrary timed action prefix (active, passive or both), can be reduced to some $C\eta$.*

Proof. The proof proceeds by showing that a non-blocking sequence of two or more timed action prefixes and their constraints is reducible according to the classic structural congruence rules and according to the dynamics of timers. This proof operates on two items at a time, starting with the two leftmost items and iterating on the sequence from left to right. In this algorithm we depict the case of relative active timed action prefixes, which can be easily generalized to cases with absolute active timers by ignoring respective instances of q in the superscript.

We assume a sequence $P ::= C_i\eta_i \cdot C_{i+1}\eta_{i+1} \cdots$. The reduction algorithm is as follows:

1. Create a new sequence P' by concatenating an unconstrained passive timer instance $\eta_i = \tau_{\tilde{t}}$ where $\tilde{t} \notin n(P)$ at the front of the sequence P and advancing the index of the rest of timed action prefixes by one so that $P' ::= \eta_i \cdot C_{i+1}\eta_{i+1} \cdot C_{i+2}\eta_{i+2} \cdots$. Formally we write $P' ::= \eta_i \cdot \{^{i+1}/_i\}\, P$. With this arrangement $\tilde{t}$ is supposed to hold the value of time at which the sequence starts to execute, and hence $\tilde{t}$ becomes a reference point. It is easier to compare different reduced sequences when they use the same symbolic reference to their start time. This step does not alter the timed behavior of the original sequence P because of the assumption that timed actions consume no execution time and that passive timers execute instantly when they are unguarded, then $P \equiv P'$.
2. In P' assume $\mathcal{N} = bn(\eta_i) \cap n(C_{i+1})$ is the set of names that appear in the subscript of η_i and anywhere in C_{i+1}. If $\mathcal{N} \neq \phi$ there are two possibilities:
   - if $\tilde{t} \in \mathrm{bn}(\eta_i)$ we define $P'' ::= C_iC'_{i+1} \cdot \eta_i\eta_{i+1} \cdots$ where $\forall \alpha_j \in \mathcal{N} : C'_{i+1} = \left\{^{\tilde{t}}/_{\alpha_j}\right\} C_{i+1}$,
   - otherwise, we define $P'' ::= C_iC'_{i+1} \cdot \eta_i\eta_{i+1} \cdots$ where $\forall \alpha_j \in \mathcal{N} : C'_{i+1} = \left\{^{q_i+p_i}/_{\alpha_j}\right\} C_{i+1}$

   we can then write $P \equiv P'$. This step holds according to the structural congruence rules of Parrow that consider constraints. If η_i was passive or bidirectional, then the passive part of it is binding over the names that appear in its subscript. If the set $\mathrm{bn}(\eta_i)$ of bound variables of η_i contains free names that appear in C_{i+1} then we have to perform alpha-conversions on all names in the intersection of those two sets before we can shift the restriction C'_{i+1} one position to the left.
3. In P' if $\mathcal{N} = bn(\eta_i) \cap n(C_{i+1}) = \phi$ then $P'' ::= C_iC_{i+1} \cdot \eta_i\eta_{i+1} \cdots$. Contrary to the previous step but based on the same structural congruence rules, if there is no intersection between $bn(\eta_i)$ and $n(C_{i+1})$ we can safely shift C_{i+1} one position to the left and $P' \equiv P''$ holds.
4. In P'' if $\{q_{i+1}, p_{i+1}\} = \phi$, then $\eta_i\eta_{i+1} \equiv \eta'_i$ where $\mathrm{bn}(\eta'_i) = \mathrm{bn}(\eta_i) \cup \mathrm{bn}(\eta_{i+1})$ and $q_{\eta'_i} = q_{\eta_i}$ and $p_{\eta'_i} = p_{\eta_i}$. If η_{i+1} is only passive, then we can reduce $\eta_i\eta_{i+1}$ to a modified η_i, where the set of bound names of η_i includes the set of bound

names of η_{i+1} as well. The super script of η_i remains the same. We enjoy this property because of our basic assumptions that timed action prefixes do not consume time on their own, and that passive timed actions execute right away when they are unguarded with higher priority than other untimed actions. This is the reason why η_i and η_{i+1} eventually execute at the same time and their bound variables then hold the same value.

5. In P'' if $\mathcal{M} = bn(\eta_i) \cap \{q_{i+1}, p_{i+1}\} \neq \phi$, where $\{q_{i+1}, p_{i+1}\}$ is the set of all names that appear in the superscript of η_{i+1}, then we have two possibilities:
   - if $\tilde{t} \in \text{bn}(\eta_i)$ then $\eta_i\eta_{i+1} \equiv \left\{{}^{\tilde{t}}/_{\alpha_j}\right\} \text{fn}(\eta_{i+1})$, $\forall \alpha_j \in \mathcal{M}$.
   - otherwise, $\eta_i\eta_{i+1} \equiv \left\{{}^{q_i+p_i}/_{\alpha_j}\right\} \text{fn}(\eta_{i+1})$, $\forall \alpha_j \in \mathcal{M}$.

   We assumed that P is non-blocking which guarantees that $q_i+p_i \leq q_{i+1}+p_{i+1}$, which means that all execution opportunities $q_i + p_i$ are scheduled before $q_{i+1} + p_{i+1}$ or coincide with them. A non-blocking list affects the execution time of whatever action prefix or process that is serially composed with it. The effect of these serially composed timed action prefixes is then only dependent on when η_{i+1} executes. However, if $\mathcal{M} \neq \phi$ then there are names in η_{i+1} whose values are dependent on the actual execution time of η_i. To safely remove η_i without loosing information, we perform as many alpha-conversions as needed to copy all necessary time information from η_i to η_{i+1}.
6. In P'' if $\mathcal{M} = bn(\eta_i) \cap \{q_{i+1}, p_{i+1}\} = \phi$ then $\eta_i\eta_{i+1} \equiv \eta_{i+1}$. In this case, the calculation of execution times of η_{i+1} does not depend on the valuation of names (and hence the execution time) of η_i. Since this is a non-blocking list, we can safely remove η_i.
7. Replace P' with the result of reduction P''. If the number of remaining items in the sequence is greater than one, perform the above algorithm recursively from Step 2 by taking the new two leftmost elements, and proceed each time by replacing the sequence you started with at Step 2 with the result of the reduction which decreases the number of items by one after each iteration. Repeat the execution until there is no more reduction possible (one item is left).

Note that Step 2 and Step 3 are mutually exclusive. The same applies to Steps 4, 5 and 6. The resulting reduced list consists of a constraint that is the conjunction of all previous constraints (with or without alpha-conversion as described above) prefixing the rightmost η (again, with or without alpha-conversions as deemed necessary by the algorithm).

Example 39. Some reductions

1. Lets take:
$$P ::= (p > 2)\tau_s^p \cdot (s < 5)\tau_s^{s,0.5}.$$
   By inserting $\tau_{\tilde{t}}$ at the beginning of P we get:
$$P' ::= \tau_{\tilde{t}}(p > 2)\tau_s^p \cdot (s < 5)\tau_s^{s,0.5}.$$
   Shifting $(p > 2)$ one position to the left requires no alpha-conversion since there is no free name in $(p > 2)$ that is bound in τ_t. Hence:
$$P' \equiv (p > 2)\tau_{\tilde{t}}\tau_s^p \cdot (s < 5)\tau_s^{s,0.5}.$$

We merge $\tau_{\tilde{t}}\tau_s^p$ together according to Step 6 of the reduction algorithm because there are no common names between the set of free names in τ_s^p (p in this case) and the set of bound names of $\tau_{\tilde{t}}$ ($\tilde{t}$ here). At this stage we get:

$$P' \equiv (p > 2)\tau_s^p \cdot (s < 5)\tau_s^{s,0.5}.$$

We find s in the subscript of τ_s^p as well as in th constraint ($s < 5$), then Step 2 of the reduction algorithm applies here. Then:

$$P' \equiv (p > 2)(p < 5) \cdot \tau_s^p\tau_s^{s,0.5}.$$

We also find that s is in the superscript of $\tau_s^{s,0.5}$, which makes Step 5 of the algorithm applicable $\Rightarrow$

$$P' \equiv (p > 2)(p < 5) \cdot \tau_s^{p,0.5}.$$

We have reduced P into a list of a single action prefix $\eta = \tau_s^{p,0.5}$ and a composed constraint $C = (p > 2)(p < 5)$.

2. Lets take a slightly modified example:

$$P ::= (p > 2)\tau^p \cdot (s < 5)\tau_s^{s,0.5}.$$

We insert $\tau_{\tilde{t}}$ according to Step 1 we get:

$$P' ::= \tau_{\tilde{t}}(p > 2)\tau^p \cdot (s < 5)\tau_s^{s,0.5}.$$

We then apply Step 3 because $\tilde{t}$ does not appear in the set of free names of $(p > 2)$. Hence we get:

$$P' \equiv (p > 2)\tau_{\tilde{t}}\tau^p \cdot (s < 5)\tau_s^{s,0.5}.$$

We next must apply Step 6, we get:

$$P' \equiv (p > 2)\tau^p \cdot (s < 5)\tau_s^{s,0.5}.$$

With Step 3 we find:

$$P' \equiv (p > 2)(s < 5) \cdot \tau^p\tau_s^{s,0.5}.$$

With Step 6 we get:

$$P' \equiv (p > 2)(s < 5) \cdot \tau_s^{s,0.5}.$$

This is possible because of the original assumption that the sequence P is non-blocking, which implicitly means that $p \leq s + 0.5$.

3. Assume: $P ::= \tau_s \cdot (s < 5)\tau^{s,2}$, by applying Step 1:

$$P' ::= \tau_{\tilde{t}}\tau_s \cdot (s < 5)\tau^{s,2}.$$

We next apply Step 4:

$$P' \equiv \tau_{\tilde{t},s} \cdot (s < 5)\tau^{s,2}.$$

According to Step 2 we get:

$$P' \equiv (\tilde{t} < 5)\tau_{\tilde{t},s} \cdot \tau^{s,2}.$$

We then need to apply step 5 to get the reduced sequence:

$$P' = (\tilde{t} < 5)\tau^{\tilde{t},2}.$$

4. We start with $P ::= \tau_s(s<10)\tau_s^{s,2}\tau_s(s<12)\tau^{s,4}$:

$$
\begin{aligned}
P &::= \tau_s(s<10)\tau_s^{s,2}\tau_s(s<12)\tau^{s,4} \quad \overset{Step1}{\Rightarrow}\\
P' &::= \tau_{\tilde{t}}\tau_s(s<10)\tau_s^{s,2}\tau_s(s<12)\tau^{s,4} \overset{Step4}{\Rightarrow}\\
P' &\equiv \tau_{\tilde{t},s}(s<10)\tau_s^{s,2}\tau_s(s<12)\tau^{s,4} \overset{Step2}{\Rightarrow}\\
P' &\equiv (\tilde{t}<10)\tau_{\tilde{t},s}\tau_s^{s,2}\tau_s(s<12)\tau^{s,4} \overset{Step5}{\Rightarrow}\\
P' &\equiv (\tilde{t}<10)\tau_s^{\tilde{t},2}\tau_s(s<12)\tau^{s,4} \quad \overset{Step4}{\Rightarrow}\\
P' &\equiv (\tilde{t}<10)\tau_s^{\tilde{t},2}\cdot(s<12)\tau^{s,4} \quad \overset{Step2}{\Rightarrow}\\
P' &\equiv (\tilde{t}<10)(\tilde{t}+2<12)\tau_s^{\tilde{t},2}\tau^{s,4} \quad \overset{Step2}{\Rightarrow}\\
P' &\equiv (\tilde{t}<10)(\tilde{t}+2<12)\tau^{\tilde{t}+2,4}
\end{aligned}
$$

■

We have now enough tools for defining timed structural congruence.

Definition 310. Rules for timed structural congruence *Let η be either c or d, $C_i, C_j \in \mathcal{C}^\tau$ and P be any process,*

1. *let $\prod_{i\in I} C_i\eta_i^{q_i,p_i}$ and $\prod_{j\in J} C_j\eta_j^{q_j,p_j}$ be non-blocking sequences of which $C_I\eta_I$ and $C_J\eta_J$ are the reductions concluded according to Lemma 38. We consider $\prod_{i\in I} C_i\eta_i^{q_i,p_i} \overset{\tau}{\equiv} \prod_{j\in J} C_j\eta_j^{q_j,p_j}$ if $q_{n_I}+p_{n_I} = q_{n_J}+p_{n_J}$, where $n_I = |I|$ and $n_J = |J|$ and if $\Omega(\mathbb{R}^{\geq 0}, C_I, \eta_I) = \Omega(\mathbb{R}^{\geq 0}, C_J, \eta_J)$. The same applies to absolute time operators by implicitly taking all q_i and q_j to be zero.*
2. *$\eta^{q,p} \overset{\tau}{\equiv} 0$ if $q+p \neq \dot{t}$, where $\dot{t}$ is the current time of the reference clock. As above, the same applies to absolute time operators by taking q to be zero.*

For better readability and to avoid ambiguity we decorate the structural equivalence symbol $\equiv$ with a τ in order to distinguish time related operations from the classic ones. This is only a notational convention without any modification to the concept of $\equiv$. For example, in (5) we mark the equivalence step with $\overset{\tau}{\equiv}$, while in the next step we use the original symbol $\equiv$. However, we consider $\overset{\tau}{\equiv}$ to be a subset of $\equiv$ and when we generally talk about equivalence we will use $\equiv$ to reference timed and untimed equivalences together.

Rule 1 in the above list deals with the timed congruence between two processes which might have different timing schemes. It stresses that different time schemes (relative or absolute) can still be equivalent in terms of actual execution time. E.g. c^5, $c^{2,3}$ and $c^{1,4}$ are all equivalent because they execute at the same time. Another example for equivalence would be $\tau_s^{0,s+5}$ and $\tau_s^{s,5}$. This rule makes successive timed taus equivalent if the supremum of the consequent actions coincide and none of them is blocking, e.g. $c_s^2 \cdot c^{s,1}$ and $c^{0,3}$. We imply here, that

q_n and p_n incorporate all alpha-conversions that took place due to passive c_s or d_s or due to process parameterization or input actions. The importance of this rule comes from the result that we do not need different calculi for relative and absolute time, and that we can replace processes with different - but equivalent - timing schemes without breaking the system.

The convenience of rule 2 arises from the fact that it defines the condition for deadlocks in timed π-*calculus* and stresses the continuity and universality properties of time in our calculus. We consider time as a flowing external value that increases continuously, and each process preserves its own account of time - in other words its own clock. If the current time exceeds the threshold a timer, this action becomes permanently inaccessible and can be *garbage collected* by utilizing the congruence with process null. The including context will also deadlock if no other execution paths were available. For example:

$$
\begin{aligned}
c^{q,p} \cdot a + b &\overset{\tau}{\equiv} 0 + b \quad \text{ if } q + p \neq \dot{t} \\
&\equiv b
\end{aligned} \tag{5}
$$

where $\dot{t}$ is the current local time of this process. Deadlocks occur if a timer was guarded by an action - or a set of actions - that execute beyond its threshold $q+p$. In case of a summation, as in the example above, if the process executes b at a point of time $t < q+p$ then the alternative path of this process in which this instance of $c^{q,p}$ occurs will not be available and $c^{q,p}$ will not fire when $\dot{t} = q+p$ because choosing "b" excludes other choices.

3.7 Commitment rules

Rule 2 of timed structural congruence raises the problem of race. By considering the example in (5), if $\dot{t} = q+p$ and "$\bar{b}$" was simultaneously enabled in another parallel process so that "b" can also execute, then according to the current commitment rules of π-*calculus* the depicted process in (5) might go either way. This is not optimal if we consider real world scenarios. Timers and interrupts have higher priority than normal execution flow in programming languages, therefore we need to incorporate this notion in our calculus so that system transitions are performed correctly. To make this concept precise we introduce in Figure 9 new commitment rules that define this behavior for active and passive timed actions.

$$
\text{PRIO-A}_\text{C}: \frac{Q \overset{\alpha}{\to} Q' \qquad\qquad\qquad C(c) \overset{c(q,p)}{\to} C(c)}{C(c) \mid Q \mid (\overline{c}\langle q,p\rangle \cdot P + \overline{\alpha} \cdot R) \overset{\tau^{q,p}}{\to} C(c) \mid Q \mid \langle q,p\rangle P} \; \textit{if } \dot{t} = q+p
$$

$$
\text{PRIO-P}_\text{C}: \frac{Q \overset{\alpha}{\to} Q' \qquad\qquad\qquad C(c) \overset{\overline{c}\langle \dot{t}\rangle}{\to} C(c)}{C(c) \mid Q \mid (c(s) \cdot P + \overline{\alpha} \cdot R) \overset{\tau_s}{\to} C(c) \mid Q \mid (s)P}
$$

Fig. 9. Priority commitment rules for timed actions

Obviously, REACTion with the external clock $C(d)$ through the timed action d follows the same arrangement and the rules in Figure 9 apply to these REACTions by replacing c with d and $C(c)$ with $C(d)$. This results in timed actions having higher priority than untimed ones which solves the race problem.

3.8 Observation equivalence & process congruence

Observation or behavioral equivalence deals with the the problem of determining whether two differently implemented processes appear behaviorally similar to an external observer. Our design concept for bisimulation is to use weak bisimulation for relaxed timed constraints and to use strong bisimulation for time properties we want to match. Here comes the visible timed action into consideration, because it is a simple way of exposing timed properties of processes using the global clock. We can imagine scenarios where a composition of hidden and visible timed actions are available in one process. Here, again, the use of weak bisimulation is a must, but we know that we only hide time properties which are relaxed and ignorable while the observable timed actions retain their visibility. In the following we explain our approach to using bisimulation in timed processes.

Weak Bisimulation Our timed τ is an internal action, whose agents do not appear to the external observer. In this case, the classic weak bisimulation of *π-calculus* will be enough to prove equivalence. We use the standard definition of weak bisimulation as proposed in [11]:

Definition ([11, Def. 9])**. Weak bisimulation**[5] *A weak (late) bisimulation is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ and $P \overset{\alpha}{\to} P'$ where the bound name $bn(\alpha)$ is fresh implies that:*

1. *If $\alpha = a(x)$ then $\exists Q'' : Q \Rightarrow\overset{a(x)}{\to} Q'' \wedge \forall u \exists Q' : Q'' \{^{u}/_{x}\} \Rightarrow Q' \wedge P' \{^{u}/_{x}\} \mathcal{R} Q'$.*
2. *If α is not an input then $\exists Q' : Q \overset{\hat{\alpha}}{\Rightarrow} Q' \wedge P'\mathcal{R}Q'$*

P and Q are weakly (late) bisimilar, written $P \dot{\approx} Q$, if they are related by a weak bisimulation.

We demonstrate using examples how the classic weak bisimulation of *π-calculus* can be used to decide behavioral equivalence of timed processes.

Example 311. Timed weak bisimulation Let us consider a system that generates the depicted timed sequence in Figure 10.

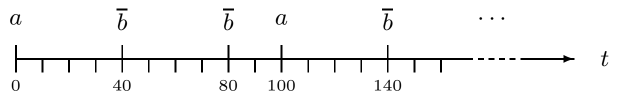

Fig. 10. Example of a timed system

We assume a global clock $C(d)$. The process P generates the required sequence:

$$
\begin{aligned}
P &::= \text{new } c(P_1 \mid C\langle c\rangle) \\
P_1 &::= a(x) \cdot c_{s,r} \cdot P_1' \\
P_1' &::= d_{s,r}^{s,100} \cdot a(x) \cdot P_1' + c_r^{r,40} \cdot \overline{b}\langle y\rangle \cdot P_1'
\end{aligned}
$$

[5] $\overset{\hat{\alpha}}{\Rightarrow}$ means an experiment $\overset{\alpha}{\Rightarrow}$ if $\alpha \neq \tau$ and $\Rightarrow$ if $\alpha = \tau$.

We can introduce another process, Q, which faithfully produces the same sequence:

$$Q ::= \text{new } c(Q_1 \mid C\langle c\rangle)$$
$$Q_1 ::= a(x) \cdot c_t^{t,40} \cdot \overline{b}\langle y\rangle \cdot c_t^{t,40} \cdot \overline{b}\langle y\rangle \cdot d_t^{t,20} \cdot Q_1$$

For an external observer, the two processes behave as follows:

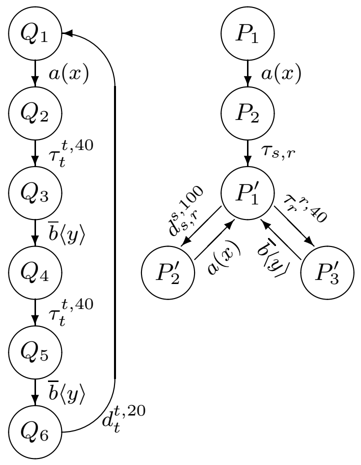

Fig. 11. Weakly timed bisimulation

To prove bisimulation, we need to build the binary relation $\mathcal{S}$ that represents all related states of P and Q:

$$\mathcal{S} = \{(P_1, Q_1), (P_2, Q_2), (P_3', Q_3), (P_1', Q_4), (P_3', Q_5), (P_1', Q_6), (P_2', Q_1), (P_1', Q_2)\}$$

since all processes are x forgetful, we can consider that the condition $\forall u \exists Q' : Q'' \{{}^{u}\!/\!_{x}\} \Rightarrow Q' \wedge P' \{{}^{u}\!/\!_{x}\} \mathcal{R} Q'$ for input actions is always fulfilled. We consider each pair of $\mathcal{S}$:

- (P_1, Q_1) there is a transition $P_1 \overset{a(x)}{\rightarrow} P_2$ for which a simulating experiment $Q_1 \overset{a(x)}{\Rightarrow} Q_2$ exists and $\forall u, (P_2 \{{}^{u}\!/\!_{x}\}, Q_2) \in \mathcal{S}$.
- (P_2, Q_2) there is a transition $P_2 \overset{\tau_{s,r}}{\rightarrow} P_1'$ for which a zero length simulating experiment from Q_2 exists and $(P_1', Q_2) \in \mathcal{S}$. After this step $s = r = 0$.
- (P_1', Q_2) there are two timer transitions simultaneously available from P_1'. In this example however, one timer transition will always fire from P_1' before the other one according to the value of time. With regard to the sequence of events till now, the timer transition $P_1' \overset{\tau_r^{r,40}}{\rightarrow} P_3'$ will fire first. For this transition, there exists a simulating experiment $Q_2 \overset{\tau_s^{s,40}}{\Rightarrow} Q_3$ and $(P_3', Q_3) \in \mathcal{S}$. After this step $r = 40$ and $t = 40$.
- (P_3', Q_3) there is a transition $P_3' \overset{b\langle y\rangle}{\rightarrow} P_1'$ for which a simulating experiment $Q_3 \overset{b\langle y\rangle}{\Rightarrow} Q_4$ exists and $(P_1', Q_4) \in \mathcal{S}$.
- (P_1', Q_4) with similar argumentation to step (P_1', Q_2), the timer transition $P_1' \overset{\tau_r^{r,40}}{\rightarrow} P_3'$ will fire first. There is a simulating experiment $Q_4 \overset{\tau_s^{s,40}}{\Rightarrow} Q_5$ and $(P_3', Q_5) \in \mathcal{S}$. After this step $r = 80$ and $t = 80$.

– (P_3', Q_5) there is a transition $P_3' \stackrel{b\langle y\rangle}{\rightarrow} P_1'$ for which a simulating experiment $Q_5 \stackrel{b\langle y\rangle}{\Rightarrow} Q_6$ exists and $(P_1', Q_6) \in \mathcal{S}$.
– (P_1', Q_6) at this point in time, the timer transition $P_1' \stackrel{d_{s,r}^{s,100}}{\rightarrow} P_2'$ will fire first, for which a simulating experiment $Q_6 \stackrel{d_t^{t,20}}{\Rightarrow} Q_1$ exists and $(P_2', Q_1) \in \mathcal{S}$. These two times are visible actions, whose execution point in time coincide at time 100, because in P: $s + 100 = 100$, and in Q: $t + 20 = 100$. Hence these two timers are timely congruent according to the chronological sequence of events and the valuation of previous time variables. After this step $r = s = 100$ and $t = 100$.
– (P_2', Q_1) there is a transition $P_2' \stackrel{a(x)}{\rightarrow} P_1'$ for which a simulating experiment $Q_1 \stackrel{a(x)}{\Rightarrow} Q_2$ exists and $\forall u, (P_1' \{{}^{u}/_{x}\}, Q_2) \in \mathcal{S}$.

We can say that Q weakly simulates P. Let us consider the inverse relation:

$$\mathcal{S}^{-1} = \{(Q_1, P_1), (Q_2, P_2), (Q_3, P_3'), (Q_4, P_1'), (Q_5, P_3'), (Q_6, P_1'), (Q_1, P_2'), (Q_2, P_1')\}$$

The same argumentation about these two processes fulfilling the alpha-conversion condition on input action because of x forgetfulness applies. We consider each pair of $\mathcal{S}^{-1}$:

– (Q_1, P_1) we have a transition $Q_1 \stackrel{a(x)}{\rightarrow} Q_2$ for which a simulating experiment $P_1 \stackrel{a(x)}{\Rightarrow} P_2$ exists and $\forall u, (Q_2 \{{}^{u}/_{x}\}, P_2) \in \mathcal{S}$.
– (Q_2, P_2) we have a transition $Q_2 \stackrel{\tau_t^{t,40}}{\rightarrow} Q_3$ for which a simulating experiment $P_2 \stackrel{\tau_r^{r,40}}{\Rightarrow} P_3'$ exists and $(Q_3, P_3') \in \mathcal{S}$. After this step $r = 40$ and $t = 40$ (s is zero by default).
– (Q_3, P_3') we have a transition $Q_3 \stackrel{b\langle y\rangle}{\rightarrow} Q_4$ for which a simulating experiment $P_3' \stackrel{b\langle y\rangle}{\Rightarrow} P_1'$ exists and $(Q_4, P_1') \in \mathcal{S}$.
– (Q_4, P_1') we have a transition $Q_4 \stackrel{\tau_t^{t,40}}{\rightarrow} Q_5$ for which a simulating experiment $P_1' \stackrel{\tau_r^{r,40}}{\Rightarrow} P_3'$ exists and $(Q_5, P_3') \in \mathcal{S}$. After this step $r = 80$ and $t = 80$.
– (Q_5, P_3') we have a transition $Q_5 \stackrel{b\langle y\rangle}{\rightarrow} Q_6$ for which a simulating experiment $P_3' \stackrel{b\langle y\rangle}{\Rightarrow} P_1'$ exists and $(Q_6, P_1') \in \mathcal{S}$.
– (Q_6, P_1') we have a transition $Q_6 \stackrel{\tau_t^{t,20}}{\rightarrow} Q_1$ for which a simulating experiment $P_1' \stackrel{\tau_{s,r}^{s,100}}{\Rightarrow} P_2'$ exists and $(Q_1, P_2') \in \mathcal{S}$. After this step $s = r = 100$ and $t = 100$.
– (Q_1, P_2') we have a transition $Q_1 \stackrel{a(x)}{\rightarrow} Q_2$ for which a simulating experiment $P_2' \stackrel{a(x)}{\Rightarrow} P_1'$ exists and $\forall u, (Q_2 \{{}^{u}/_{x}\}, P_1') \in \mathcal{S}$.
– (Q_2, P_1') we have a transition $Q_2 \stackrel{\tau_t^{t,40}}{\rightarrow} Q_3$ for which a simulating experiment $P_1' \stackrel{\tau_r^{r,40}}{\Rightarrow} P_3'$ exists and $(Q_3, P_3') \in \mathcal{S}$. After this step $r = 140$ and $t = 140$.

The proof concludes inductively since these processes are endless loops and the values of r, s and t increase continuously. We conclude then that P weakly simulates Q as well, and hence, $\mathcal{S}$ is a weak bisimulation. ∎

For weak bisimulation, the invisible timers *don't* have to coincide. The above example however, shows that even the timed τ instances do coincide. This is just because we specified a timed behavior in Figure10 which two different implementations must fulfill, not because invisible timers have to be timely congruent. A modified process Q' as follows:

$$Q'_1 ::= a(x) \cdot d_t \cdot \overline{b}\langle y\rangle \cdot \overline{b}\langle y\rangle \cdot d_t^{t,100} \cdot Q'_1$$

will also prove to be weakly bisimilar to P. As a matter of fact, weak bisimulation does not convey all the information about the timing properties of bisimilar processes because it ignores tau actions (timed or non-timed) and does not consider the values of time parameters. However, book-keeping of current values of time variables must continue in order to be able to evaluate congruence of subsequent visible timers. Weak bisimulation is suitable for relaxed timed behavior that can be ignored.

Example 312. Timed vs. untimed processes A scenario where weak bisimulation might be undesirable or insufficient is when a process like

$$R ::= a(x) \cdot \overline{b}\langle y\rangle \cdot \overline{b}\langle y\rangle \cdot R$$

also proves to be bisimilar to either of P or Q of the example above. It is easy to see from Figure 12 that $R \dot{\approx} Q$, however, we do not spare the proof. We start by generating the binary relation

$$\mathcal{S}' = \{(Q_1, R_1), (Q_2, R_2), (Q_3, R_2), (Q_4, R_3), (Q_5, R_3), (Q_6, R_1)\}$$

and consider each pair in it:

- (Q_1, R_1) for the transition $Q_1 \overset{a(x)}{\rightarrow} Q_2$ there exists a simulating experiment $R_1 \overset{a(x)}{\Rightarrow} R_2$ and since both processes are x forgetful $\forall u, (Q_2\,\{u/x\}\,, R_2) \in \mathcal{S}'$.
- (Q_2, R_2) for the transition $Q_2 \overset{\tau_s^{s,40}}{\rightarrow} Q_3$ there exists a zero-length experiment from R_2 and $(Q_3, R_2) \in \mathcal{S}'$.
- (Q_3, R_2) for the transition $Q_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} Q_4$ there exists a simulating experiment $R_2 \overset{\overline{b}\langle y\rangle}{\Rightarrow} R_3$ and $(Q_4, R_3) \in \mathcal{S}'$.
- (Q_4, R_3) for the transition $Q_4 \overset{\tau_s^{s,40}}{\rightarrow} Q_5$ there exists a zero-length experiment from R_3 and $(Q_5, R_3) \in \mathcal{S}'$.
- (Q_5, R_3) for the transition $Q_5 \overset{\overline{b}\langle y\rangle}{\rightarrow} Q_6$ there exists a simulating experiment $R_3 \overset{\overline{b}\langle y\rangle}{\Rightarrow} R_1$ and $(Q_6, R_1) \in \mathcal{S}'$.
- (Q_6, R_1) for the transition $Q_6 \overset{\tau_s^{s,20}}{\rightarrow} Q_1$ there exists a zero-length experiment from R_1 and $(Q_1, R_1) \in \mathcal{S}'$.

We conclude that R_1 weakly simulates Q_1.

We consider now the inverse binary relation

$$\mathcal{S}'^{-1} = \{(R_1, Q_1), (R_2, Q_2), (R_2, Q_3), (R_3, Q_4), (R_3, Q_5), (R_1, Q_6)\}$$

and discuss each pair in it:

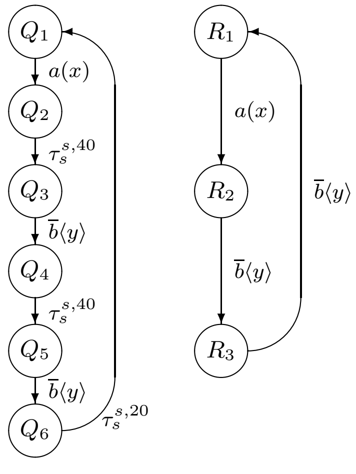

Fig. 12. Weak bisimulation between timed and untimed processes

- (R_1, Q_1) for the transition $R_1 \overset{a(x)}{\rightarrow} R_2$ there exists a simulating experiment $Q_1 \overset{a(x)}{\Rightarrow} Q_2$ and since both processes are x forgetful $\forall u, (R_2 \{{}^{u}\!/_{x}\}, Q_2) \in \mathcal{S}'$.
- (R_2, Q_2) for the transition $R_2 \overset{\overline{b}\langle y\rangle}{\rightarrow} R_3$ there exists a simulating experiment $Q_2 \overset{\overline{b}\langle y\rangle}{\Rightarrow} Q_4$ and $(R_3, Q_4) \in \mathcal{S}'$.
- (R_2, Q_3) for the transition $R_2 \overset{\overline{b}\langle y\rangle}{\rightarrow} R_3$ there exists a simulating experiment $Q_3 \overset{\overline{b}\langle y\rangle}{\Rightarrow} Q_4$ and $(R_3, Q_4) \in \mathcal{S}'$.
- (R_3, Q_4) for the transition $R_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} R_1$ there exists a simulating experiment $Q_4 \overset{\overline{b}\langle y\rangle}{\Rightarrow} Q_6$ and $(R_1, Q_6) \in \mathcal{S}'$.
- (R_3, Q_5) for the transition $R_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} R_1$ there exists a simulating experiment $Q_5 \overset{\overline{b}\langle y\rangle}{\Rightarrow} Q_6$ and $(R_1, Q_6) \in \mathcal{S}'$.
- (R_1, Q_6) for the transition $R_1 \overset{a(x)}{\rightarrow} R_2$ there exists a simulating experiment $Q_6 \overset{a(x)}{\Rightarrow} Q_2$ and $\forall u, (R_2 \{{}^{u}\!/_{x}\}, Q_2) \in \mathcal{S}'$.

Then Q_1 weakly simulates R_1, and hence $R_1 \dot{\approx} Q_1$. Since bisimulation is transitive then $R_1 \approx P_1$ too. This example represents the extreme case of untimed process simulating another one with completely hidden timing. ■

Strong bisimulation In contrast to weak bisimulation, strong bisimulation assumes that that all actions are visible and requires that each visible action from one side to be matched exactly by the same action on the other side. We need to use a modified version of strong equivalence to study bisimilarity between processes with visible timed actions d. This is because we have to consider the actual execution time (in other words the parameters) of these actions rather than just the semantics (relative vs. absolute) of them. We study the properties of timed structural congruence and strong simulation to integrate our structural congruence rules into the classical π-*calculus* congruence rules. As a basis, we take the classic theorem and prove the correctness under the rules of Definition 310.

Theorem 313 (extending [10, 12.8]). *If $P \xrightarrow{\alpha} A$ and $P \stackrel{\tau}{\equiv} Q$, then there exists B such that $Q \xrightarrow{\alpha} B$ and $A \stackrel{\tau}{\equiv} B$. Hence $\stackrel{\tau}{\equiv}$ is a strong bisimulation.*

Proof. For each possible last step of inference of commitment $P \xrightarrow{\alpha} A$, all possible congruences $P \stackrel{\tau}{\equiv} Q$ that result from a single application of a timed structural congruence rule must be proved (we do not reprove classic rules but build on them instead). The general case of congruences that result from multiple applications of structural congruence rules can be followed by iterating the special case. we assume in all proof parts that the clock process $C(d)$ exists and reacts with the processes in question. Since the absolute timer d^p can be generalized to the relative one $d^{0,p}$, there is no need to make separate proofs for each of them. In addition, d_s is an input action and follows classic π-*calculus* congruence rules by default, therefore it needs no special proof here.

SUM$_\mathbf{C}$

$$M + \alpha A + N \xrightarrow{\alpha} A$$

We assume that $M + \alpha A + N$ is P. Regarding timed structural congruence, there are several ways in which Q will be structurally congruent with P:

- α is $\overline{d}^{q,p}$ and $Q = M + \overline{d}^{q',p'} A + N$ where $\overline{d}^{q,p} \stackrel{\tau}{\equiv} \overline{d}^{q',p'}$. In this case, Q will take the transition $Q \xrightarrow{\overline{d}^{q',p'}} A$ at the same time as P would take $P \xrightarrow{\overline{d}^{q,p}} A$ and since no alpha-conversion takes place here we can directly say that the two resulting agents are structurally congruent as well.
- assume that $\dot{t} \neq q + p$. We take P to be $\overline{d}^{q,p} B + \alpha A + N$, and we take Q to be $0 + \alpha A + N$. As long as $\dot{t}$ is not equal to $q + p$ and according to Rule 2 of timed structural congruence we get $Q \stackrel{\tau}{\equiv} P$, and both processes will take the same commitment $\xrightarrow{\alpha}$ to the same agent A.

L-PAR$_\mathbf{C}$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}$$

- assume $R = P|Q$. For some process S to be structurally congruent with R we assume that $S = P'|Q$ where $P' \stackrel{\tau}{\equiv} P$ using one of the timed structural congruence rules:
  - using Rule 1 of timed structural congruence: we assume $\alpha = \overline{d}^{q,p}$, and $P' \xrightarrow{\overline{d}^{q',p'}} A'$ where $\overline{d}^{q',p'} \stackrel{\tau}{\equiv} \overline{d}^{q,p}$. The inference tree becomes:

    $$\frac{P' \xrightarrow{\overline{d}^{q',p'}} A'}{P' \mid Q \xrightarrow{\overline{d}^{q',p'}} A' \mid Q} \text{ L-PAR}_\text{c}$$

    We take R' to be $A|Q$ and S' to be $A'|Q$. By induction we conclude $A' \equiv A$ and hence $S' \equiv R'$.
  - using Rule 2 of timed structural congruence: $P \xrightarrow{\alpha} A$ as long as $\dot{t} \neq q+p$ and $P' \stackrel{\tau}{\equiv} P$ under this condition, then $P' \xrightarrow{\alpha} A'$ and using L-PAR$_\mathbf{C}$ we get $S \xrightarrow{\alpha} A'|Q$. By induction we get $A' \equiv A$ and then $S' \equiv R'$.

R-PAR$_\mathbf{C}$

$$\frac{Q \overset{\alpha}{\rightarrow} A}{P \mid Q \overset{\alpha}{\rightarrow} P \mid A}$$

The proof proceeds similar to L-PAR$_\mathbf{C}$ for all three timed operators $\overline{d}^{q,p}, \overline{d}^{p}$ and d_s.

RES$_\mathbf{C}$

$$\frac{P \overset{\alpha}{\rightarrow} A}{\text{new } x \; P \overset{\alpha}{\rightarrow} \text{new } x \; A} \; \textit{if } \alpha \notin \{x, \overline{x}\}$$

In principle, all names in timed action prefixes in their public form $\overline{d}^{q,p}, \overline{d}^{p}$ and d_s are not restricted to the process scope in order to keep them visible to the external observer. Therefore we consider that the condition $\alpha \notin \{x, \overline{x}\}$ is always fulfilled for these actions.

- if $\alpha = \overline{d}^{q,p}$ and $P \overset{\overline{d}^{q,p}}{\rightarrow} A$ we take R to be new x P. For some process S to be structurally congruent with R using timed structural congruence rules we have the following possibilities:
  - by taking $S =$ new x P' where P' and P are timely congruent according to Rule 1 of timed structural congruence. Then by RES$_\mathbf{C}$ P' offers the commitment $P' \overset{\overline{d}^{q',p'}}{\rightarrow} A'$ where $\overline{d}^{q',p'} \overset{\tau}{\equiv} \overline{d}^{q,p}$ and $n(\overline{d}^{q',p'}) \notin \{x, \overline{x}\}$. By induction we get that $A' \equiv A$, and by taking R' to be new x A and S' to be new x A' then $R' \equiv S'$ too.
  - using Rule 2 of timed structural congruence: $P \overset{\alpha}{\rightarrow} A$ as long as $\dot{t} \neq q+p$ and $P' \overset{\tau}{\equiv} P$ under this condition, then $P' \overset{\alpha}{\rightarrow} A'$. By induction, $A' \overset{\tau}{\equiv} A$. Assume S to be new x P', then $S \overset{\alpha}{\rightarrow} A'$ using the RES$_\mathbf{C}$ commitment rule. Same as above, we conclude that $A' \equiv A$, and $R' \equiv S'$.

REP$_\mathbf{C}$

$$\frac{P \mid !P \overset{\alpha}{\rightarrow} A}{!P \overset{\alpha}{\rightarrow} A}$$

We have defined clocks to be replicated timed processes. However, depending on the application we want to model, client processes can be replicated too.

- assume $\alpha = \overline{d}^{q,p}$ and $R = P|!P$. Then there is $P' \overset{\tau}{\equiv} P$ such that $P' \overset{\overline{d}^{q',p'}}{\rightarrow} A'$ and $\overline{d}^{q',p'} \overset{\tau}{\equiv} \overline{d}^{q,p}$ according to Rule 1 of timed structural congruence. By induction we get $A \equiv A'$. Take $Q = P'|!P'$ then according to above $Q \overset{\tau}{\equiv} R$. Since $P' \overset{\overline{d}^{q',p'}}{\rightarrow} A'$ we conclude $Q \overset{\overline{d}^{q',p'}}{\rightarrow} A'$, and therefore $!P' \overset{\overline{d}^{q',p'}}{\rightarrow} A'$. We take Q' to be $!P'$ and R' to be $!P$, then $Q' \equiv R'$ as required.
- we assume there is $P' \overset{\tau}{\equiv} P$ according to Rule 2 of timed structural congruence. As long as $\dot{t}$ is not equal to the time threshold of d in P, then P' and P are equally committed to α. Under this very condition, we take $R = P|!P$ and $Q = P'|!P'$ and hence $Q \overset{\tau}{\equiv} R$. Analogously we get $Q' \equiv R'$ by taking $Q' =!P'$ and $R' =!P$. By induction on $P \overset{\alpha}{\rightarrow} A$ and $P' \overset{\alpha}{\rightarrow} A'$ we get $A' \equiv A$.

PRIO-A$_\mathbf{C}$

$$\frac{Q \overset{\alpha}{\rightarrow} Q' \qquad\qquad C(d) \overset{d(q,p)}{\rightarrow} C(d)}{C(d) \mid Q \mid (\overline{d}\langle q,p\rangle P + \overline{\alpha}R) \overset{\tau^{q,p}}{\rightarrow} C(d) \mid Q \mid \langle q,p\rangle P} \; \textit{if } \dot{t} = q+p$$

We assume S is $C(d)|Q|(\overline{d}\langle q,p\rangle P+\overline{\alpha}R)$ and S' is $C(d)|Q|\langle q,p\rangle P$. For T to be structurally congruent with S using a single application of timed structural congruence rules we have the following cases:

- using Rule 1 of timed structural congruence T can only be $C(d)|Q|(\overline{d}\langle q',p'\rangle P+\overline{\alpha}R)$ where $\overline{d}\langle q+p\rangle \overset{\tau}{\equiv} \overline{d}\langle q',p'\rangle$. The commitment $\overline{d}\langle q',p'\rangle P \overset{\overline{d}\langle q',p'\rangle}{\rightarrow} \langle q',p'\rangle P$ indicates the induction $\langle q,p\rangle P \equiv \langle q',p'\rangle P$. We finally take T' to be $C(d)|Q|\langle q',p'\rangle P$ and we get $S' \equiv T'$.
- using Rule 2 of timed structural congruence T can be $C(d)|Q|(0+\overline{\alpha}R)$. $T \overset{\tau}{\equiv} S$ as long as $\dot{t} \neq q+p$. This congruence rule is irrelevant for PRIO-A$_{\mathbf{C}}$ because time predicates are opposites.

PRIO-P$_{\mathbf{C}}$

$$\frac{Q \overset{\alpha}{\rightarrow} Q' \qquad\qquad C(d) \overset{\overline{d}\langle t\rangle}{\rightarrow} C(d)}{!C(d) \mid Q \mid (d(s)P+\overline{\alpha}R) \overset{\tau_s}{\rightarrow} C(d) \mid Q \mid (s)P}$$

The congruence of sub process $d(s)P$ and hence the congruence of $C(d)|Q|(d(s)P+\overline{\alpha}R)$ with some other processes is not different from the classic cases of input action prefixes that leave abstractions behind. These cases are dealt with in the classic commitment theory of π-*calculus*.

Having covered all possibilities for timed congruence we conclude that timed congruence respects commitment rules.

In conjunction with Theorem 5.13 in [10] we conclude that timed structural congruence is strong bisimulation.

The integration of timed operators in π-*calculus* continues smoothly without impact on Lemma 12.9, Lemma 12.10 and Theorem 12.11 of [10] so that we can define timed strong bisimulation as a strong equivalence. This property comes from Definition 12.13 of [10]:

Definition ([10, 12.13])**. Strong simulation** *A binary relation $\mathcal{S}$ over $\mathcal{P}^\pi$ is a strong simulation if, whenever $P\mathcal{S}Q$,*
if $P \overset{\alpha}{\rightarrow} A$ then there exists B such that $Q \overset{\alpha}{\rightarrow} B$ and $A\mathcal{S}B$.
If both $\mathcal{S}$ and its converse are strong simulations then $\mathcal{S}$ is a strong bisimulation. Two agents A and B are strongly equivalent, written $A \sim B$, if the pair (A,B) is in some strong bisimulation.

Since our contribution does not alter the concept of agents in π-*calculus*, the above definition is also valid for π^τ-*calculus*. The elaborate case here is the passive time operators τ_s and d_s for which the equivalence rule for abstractions should hold in our extended calculus as well. This rule says that if $F = (s).P$ and $G = (s).Q$ are abstractions then they are bisimilar if $F \sim G$ and $\{^s/_x\}\, P \sim \{^s/_x\}\, Q$. In Examples 311 and 312 we were lucky to have x forgetful processes, which only made the proof simpler. However, care should be taken when proving simulations that involve the passive time operator, which is no difference from classic simulation in π-*calculus*. The case of concretions (relevant for absolute and relative timing operators) is harmless since no alpha-conversion takes place in the local process.

Example 314. Let's consider the sensor process in Example 33. We rewrite[6] this process in its absolute and relative formats using the visible time operator d instead of τ.

$$S'(p) ::= d_t^{t+p} \cdot \overline{c}\langle d\rangle \cdot c(a) \cdot S'\langle p\rangle + c(r) \cdot d_t \cdot S'\langle p\rangle$$
$$S''(t,p) ::= d^{t,p} \cdot \overline{c}\langle d\rangle \cdot c(a) \cdot S''\langle t+p,p\rangle + c(r) \cdot d_t \cdot S''\langle t,p\rangle$$

After detaching the sequential compositions of $S(p)$ and $S(t,p)$ we can easily create $\mathcal{S}$:

$$\begin{aligned} S'(p) &::= d_t^{t+p} \cdot S'_1 + c(r) \cdot S'_3 \\ S'_1 &::= \overline{c}\langle d\rangle \cdot S'_2 \\ S'_2 &::= c(a) \cdot S'\langle p\rangle \\ S'_3 &::= d_t \cdot S'\langle p\rangle \end{aligned} \qquad \begin{aligned} S''(t,p) &::= d^{t,p} \cdot S''_1 + c(r) \cdot S''_3 \\ S''_1 &::= \overline{c}\langle d\rangle \cdot S''_2 \\ S''_2 &::= c(a) \cdot S''\langle t+p,p\rangle \\ S''_3 &::= d_t \cdot S''\langle t,p\rangle \end{aligned}$$

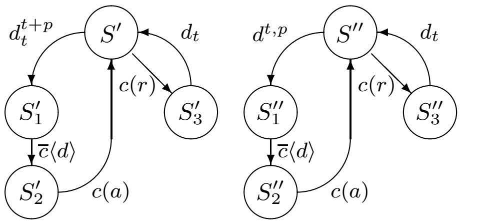

Fig. 13. Strong bisimulation between timed processes

$$\mathcal{S} = \{(S', S''), (S'_1, S''_1), (S'_2, S''_2), (S'_3, S''_3)\}$$

By looking at Figure 13 it is obvious that for most state pairs the commitments are simulated in both directions (the inverse relation $\mathcal{S}^{-1}$ too). Because of the different time schemes however, we would like to verify that (S', S'') correctly simulate each other through the two timed action prefixes $d_t^{t,p}$ and d^{t+p}. From state S' the execution time of d is $t+p$ after which t presumes that value. The next execution round takes place when the process finishes the loop and goes back to S' where the timed action d_t^{t+p} will be enabled with the execution time set to $t+p+p$ because of alpha conversion of t. From state S'' the execution time of $d^{t,p}$ is p relative to t which is $t+p$, which is same as above. Alpha conversion of t takes place later when the process is recursively invoked with $S''\langle t+p,p\rangle$, which sets the next execution time of d to $t+p+p$.

We also would like to verify that (S'_3, S''_3) and its counterpart in $\mathcal{S}^{-1}$ also perform their timed actions in a correctly simulated fashion. From S'_3, t will be alpha converted to the current time and the next execution time of d after invoking $S'\langle p\rangle$ becomes $t+p$. From S''_3 using the same mechanism, t assumes the value of the current time and the next execution time of d after invoking $S''\langle t,p\rangle$ becomes $t+p$ as well. We can safely say, that S' and S'' strongly simulate each other because they imitate the behavior (timed and untimed) faithfully. ■

[6] We simplified the processes and also did some renaming to preserve space.

Example 315. Let's consider the timed behavior in Figure 10. We want to show how this behavior can be achieved with global clocks only, which means that there are no silent timers. We assume a global clock $C(d)$, and re-declare P and Q from Example 311 without internal (or private) clocks as follows:

$$
\begin{aligned}
P_1 &::= a(x) \cdot d_{s,r} \cdot P_1' \\
P_1' &::= d_{s,r}^{s,100} \cdot a(x) \cdot P_1' + d_r^{r,40} \cdot \overline{b}\langle y \rangle \cdot P_1'
\end{aligned}
$$

$$Q_1 ::= a(x) \cdot d_t^{t,40} \cdot \overline{b}\langle y \rangle \cdot d_t^{t,40} \cdot \overline{b}\langle y \rangle \cdot d_t^{t,20} \cdot Q_1$$

For an external observer, the two processes behave as follows:

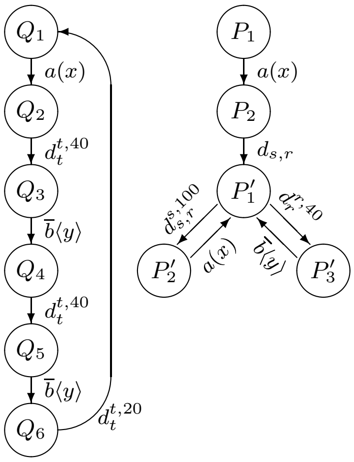

Fig. 14. Strong timed bisimulation

We build the binary relation $\mathcal{S}$ that represents all related states of P and Q:

$$\mathcal{S} = \{(P_1, Q_1), (P_2, Q_2), (P_3', Q_3), (P_1', Q_4), (P_3', Q_5), (P_1', Q_6), (P_2', Q_1), (P_1', Q_2)\}$$

since all processes are x forgetful, we can consider that the condition $\forall u \exists Q' : Q'' \{u/x\} \Rightarrow Q' \wedge P' \{u/x\} \mathcal{R} Q'$ for input actions is always fulfilled. We consider each pair of $\mathcal{S}$:

- (P_1, Q_1) there is a transition $P_1 \overset{a(x)}{\to} P_2$ for which a simulating transition $Q_1 \overset{a(x)}{\to} Q_2$ exists and $\forall u, (P_2 \{u/x\}, Q_2) \in \mathcal{S}$.
- (P_2, Q_2) after reducing the sequence $d_{s,r} \cdot d_r^{r,40}$ to $d_r^{\tilde{t},40}$ as discussed in Lemma 38, the transition $P_2 \overset{d_r^{\tilde{t},40}}{\to} P_3'$ has a simulating transition $Q_2 \overset{d_t^{t,40}}{\to} Q_3$ and $(P_3', Q_3) \in \mathcal{S}$. After this step $s = 0, r = 40$ and $t = 40$.
- (P_3', Q_3) there is a transition $P_3' \overset{\overline{b}\langle y \rangle}{\to} P_1'$ for which a simulating transition $Q_3 \overset{\overline{b}\langle y \rangle}{\to} Q_4$ exists and $(P_1', Q_4) \in \mathcal{S}$.
- (P_1', Q_4) there is a transition $P_1' \overset{d_r^{r,40}}{\to} P_3'$ for which a simulating transition $Q_4 \overset{d_s^{s,40}}{\to} Q_5$ and $(P_3', Q_5) \in \mathcal{S}$. After this step $r = 80$ and $t = 80$.

– (P'_3, Q_5) there is a transition $P'_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} P'_1$ for which a simulating transition $Q_5 \overset{\overline{b}\langle y\rangle}{\rightarrow} Q_6$ exists and $(P'_1, Q_6) \in \mathcal{S}$.
– (P'_1, Q_6) there is a transition $P'_1 \overset{d_{s,r}^{s,100}}{\rightarrow} P'_2$ for which a simulating transition $Q_6 \overset{d_s^{s,20}}{\rightarrow} Q_1$ exists and $(P'_2, Q_1) \in \mathcal{S}$. These two times are visible actions, whose execution point in time coincide at time 100, because in P: $s+100 = 100$, and in Q: $s + 20 = 100$. Hence these two timers are timely congruent, according to the chronological sequence of events and the valuation of previous time variables. After this step $r = s = 100$ and $t = 100$.
– (P'_2, Q_1) there is a transition $P'_2 \overset{a(x)}{\rightarrow} P'_1$ for which a simulating transition $Q_1 \overset{a(x)}{\rightarrow} Q_2$ exists and $\forall u, (P'_1 \{{}^{u}\!/\!_{x}\}, Q_2) \in \mathcal{S}$.
– (P'_1, Q_2) there are two timer transitions simultaneously available from P'_1. Again, the timer with the shorter time-out value will win. With regard to the sequence of events till now, the timer transition $P'_1 \overset{d_r^{r,40}}{\rightarrow} P'_3$ will fire first. For this transition, there exists a simulating transition $Q_2 \overset{d_s^{s,40}}{\rightarrow} Q_3$ exists and $(P'_3, Q_3) \in \mathcal{S}$. After this step $r = 140$ and $t = 140$.

We can say that Q strongly simulates P. Let us consider the inverse relation:

$$\mathcal{S}^{-1} = \{(Q_1, P_1), (Q_2, P_2), (Q_3, P'_3), (Q_4, P'_1), (Q_5, P'_3), (Q_6, P'_1), (Q_1, P'_2), (Q_2, P'_1)\}$$

The same argumentation about these two processes fulfilling the alpha-conversion condition on input action because of x forgetfulness applies. We consider each pair of $\mathcal{S}^{-1}$:

– (Q_1, P_1) we have a transition $Q_1 \overset{a(x)}{\rightarrow} Q_2$ for which a simulating transition $P_1 \overset{a(x)}{\rightarrow} P_2$ exists and $\forall u, (Q_2 \{{}^{u}\!/\!_{x}\}, P_2) \in \mathcal{S}$.
– (Q_2, P_2) we have a transition $Q_2 \overset{d_t^{t,40}}{\rightarrow} Q_3$ for which a simulating transition $P_2 \overset{d_r^{\tilde{t},40}}{\rightarrow} P'_3$ exists (the result of reducing $d_{s,r} \cdot d_r^{r,40}$) and $(Q_3, P'_3) \in \mathcal{S}$. After this step $s = 0, r = 40$ and $t = 40$.
– (Q_3, P'_3) we have a transition $Q_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} Q_4$ for which a simulating transition $P'_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} P'_1$ exists and $(Q_4, P'_1) \in \mathcal{S}$.
– (Q_4, P'_1) we have a transition $Q_4 \overset{d_t^{t,40}}{\rightarrow} Q_5$ for which a simulating transition $P'_1 \overset{d_r^{r,40}}{\rightarrow} P'_3$ exists and $(Q_5, P'_3) \in \mathcal{S}$. After this step $r = 80$ and $t = 80$.
– (Q_5, P'_3) we have a transition $Q_5 \overset{\overline{b}\langle y\rangle}{\rightarrow} Q_6$ for which a simulating transition $P'_3 \overset{\overline{b}\langle y\rangle}{\rightarrow} P'_1$ exists and $(Q_6, P'_1) \in \mathcal{S}$.
– (Q_6, P'_1) we have a transition $Q_6 \overset{d_t^{t,20}}{\rightarrow} Q_1$ for which a simulating transition $P'_1 \overset{d_{s,r}^{s,100}}{\rightarrow} P'_2$ exists and $(Q_1, P'_2) \in \mathcal{S}$. After this step $s = r = 100$ and $t = 100$.
– (Q_1, P'_2) we have a transition $Q_1 \overset{a(x)}{\rightarrow} Q_2$ for which a simulating transition $P'_2 \overset{a(x)}{\rightarrow} P'_1$ exists and $\forall u, (Q_2 \{{}^{u}\!/\!_{x}\}, P'_1) \in \mathcal{S}$.
– (Q_2, P'_1) we have a transition $Q_2 \overset{d_t^{t,40}}{\rightarrow} Q_3$ for which a simulating transition $P'_1 \overset{d_r^{r,40}}{\rightarrow} P'_3$ exists and $(Q_3, P'_3) \in \mathcal{S}$. After this step $r = 140$ and $t = 140$.

The above proof concludes inductively on the endless loop of these two processes. We conclude then that P strongly simulates Q as well, and hence, $\mathcal{S}$ is a strong bisimulation. ■

4 Discrete π^τ-*Calculus*

Up till now, all time variables and values were in the real number domain $\mathbb{R}^{\geq 0}$. However, it is desirable to model time in the discrete domain $\mathbb{N}^0$. We understand discrete time in the context of simulation; it is a time abstraction technique to facilitate building tools that simulate timed systems on computing machines. We adopt the notion of time slices as declared by [1, p. 661], that is:

"in time slice n+1" means "at some time point p such that $n \leq p < n+1$".

Because of the minimal design of π^τ-*calculus*, the impact of discrete time on it is minimal too. In the following we present the contribution of that to the signature and structural congruence rules.

4.1 Signature of Discrete π^τ-*Calculus*

In discrete π^τ-*Calculus*, we use the following symbology: $\underline{\dot{t}}$ is the current time slice of the reference clock and it ranges over $\mathbb{N}^0$. $\underline{d}^{q,p}$, $\underline{d}^p$, $\underline{d}_s$, $\underline{d}_s^{q,p}$, $\underline{\tau}^{q,p}$, $\underline{\tau}^p$ and $\underline{\tau}_s$, are the absolute, relative and passive visible and invisible time action prefixes (or timers) whose parameters q, p and s range over $\mathbb{N}^0$. We use $\underline{\pi}^\tau$ to reference the set of all timed action prefixes which is a subset of π. If available, superscript parameters define when the timer should fire according to the reference clock, and subscript parameters are alpha-converted on execution time to hold the current value of $\underline{\dot{t}}$. The same applies to $\underline{c}_s^{q,p}$, $\underline{c}_s^p$ and $\underline{c}_s$ that reference internal discrete clocks of their processes. $\mathcal{C}^\tau$ is the set of all constraints in which $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$ is the comparison operator.

4.2 Structural Congruence in Discrete π^τ-*Calculus*

Aside from using natural numbers to represent time, there is principally no difference to the real-time structural congruence rules.

Definition 41. Rules for discrete timed structural congruence *Let $\underline{\eta}$ be either $\underline{\tau}$ or $\underline{d}$,*

1. *let $\prod_{i\in I} C_i\underline{\eta}_i^{q_i,p_i}$ and $\prod_{j\in J} C_j\underline{\eta}_j^{q_j,p_j}$ be non-blocking sequences of which $C_I\underline{\eta}_I$ and $C_J\underline{\eta}_J$ are the reductions concluded according to Lemma 38. We consider $\prod_{i\in I} C_i\underline{\eta}_i^{q_i,p_i} \stackrel{\tau}{\equiv} \prod_{j\in J} C_j\underline{\eta}_j^{q_j,p_j}$ if $q_{n_I} + p_{n_I} = q_{n_J} + p_{n_J}$, where $n_I = |I|$ and $n_J = |J|$ and if $\Omega(\mathbb{N}^0, C_I, \underline{\eta}_I) = \Omega(\mathbb{N}^0, C_J, \underline{\eta}_J)$. The same applies to absolute time operators by implicitly taking all q_i and q_j to be zero.*
2. *$\underline{\eta}^{q,p} \stackrel{\tau}{\equiv} 0$ if $q + p \neq \dot{t}$, where $\dot{t}$ is the current time of the reference clock. As above, the same applies to absolute time operators by taking q to be zero.*

We see similarity between the axiomatization of BPA^{sat} and BPA^{dat} - the algebra for real-time and discrete process algebra in [1] respectively - which is no surprise to us.

4.3 Observation equivalence & process congruence

By atomic replacement of real-time structural congruence by the discrete structural congruence, we build the same weak and strong bisimulation for discrete π^{τ}*-calculus* as before.

References

1. J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: real time and discrete time. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, chapter 10, pages 627–684. Elsevier Science Ltd, 2001.
2. M. Berger and N. Yoshida. Timed, distributed, probabilistic, typed processes. *Programming Languages and Systems*, pages 158–174, 2009.
3. J.F. Díaz, C. Rueda, and F.D. Valencia. Pi+-calculus: A calculus for concurrent processes with constraints. *CLEI Electronic Journal*, 1(2), 1998.
4. D. Gilbert and C. Palamidessi. Concurrent constraint programming with process mobility. *Computational Logic-CL 2000*, pages 463–477, 2000.
5. Christian Haack and Alan Jeffrey. Timed spi-calculus with types for secrecy and authenticity. In *CONCUR 2005 – Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 202–216, 2005.
6. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221 – 239, 1995.
7. W. Jin, H. Wang, and M. Zhu. Modeling MARTE sequence diagram with timing pi-calculus. In *14th IEEE Int. Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2011)*, pages 61–66. IEEE, 2011.
8. J.Y. Lee and J. Zic. On modeling real-time mobile processes. In *Proc. of 25th Australasian Conference on Computer Science (ACSC 2002)*, volume 4 of *CRPIT*, pages 139–147. Citeseer, ACS, 2002.
9. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105 – 157, 2003.
10. Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge Univ. Press, 1999.
11. Joachim Parrow. An introduction to the π-calculus. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, chapter 8, pages 479–543. Elsevier Science Ltd, 2001.
12. Qiong Wang, Chenglie Du, Chunyan Ma, and Gang Li. Extension of TD-pi calculus in real-time distributed virtual-test system description. In *Int. Conf. on Computer Science and Software Engineering*, volume 3, pages 363–369. IEEE, December 2008.

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years.
A complete list of reports dating back to 1987 is available from `http://aib.informatik.rwth-aachen.de/`**.**
To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2008-01 *	Fachgruppe Informatik: Jahresbericht 2007
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07	Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10	George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11	George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13	Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14	Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15	Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16	Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17	Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18	Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19	Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-02	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications

2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices

2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation

2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs

2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata

2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms

2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting

2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs

2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut

2010-11 Martin Zimmermann: Parametric LTL Games

2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut

2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems

2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination

2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles

2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten

2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit

2010-19 Felix Reidl: Experimental Evaluation of an Independent Set Algorithm

2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

2011-08 Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog

2011-11 Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains

2011-13 Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.