

An Extension of π -Calculus with Real-Time and its Realization in Logic Programming

Neda Saeedloei and Gopal Gupta

Department of Computer Science,
University of Texas at Dallas,
Richardson, TX 75080, USA
{nxs048000, gupta}@utdallas.edu

Abstract. We extend π -calculus with real-time by adding clocks, and give an operational semantics for it. We introduce timed bisimilarity and develop its properties, in particular, expansion theorem for real-time, concurrent, mobile processes. We present the outline of an implementation based on co-induction, coroutining, and constraint logic programming over reals of this operational semantics. We show how timed π -calculus can be used for modeling and verification of complex real-time systems. We illustrate our work by applying it to modeling of practical problems such as the generalized railroad crossing problem.

1 Introduction

The π -calculus was introduced by Milner et al. [15] with the aim of modeling concurrent/mobile processes. The π -calculus provides a conceptual framework for describing systems whose components interact with each other. It contains an algebraic language for descriptions of processes in terms of the communication actions they can perform. Theoretically, the π -calculus can model mobility, concurrency and message exchange between processes as well as infinite computation (through the infinite replication operator (!)).

In many cases, processes tend to be controllers that control physical devices; therefore, they have to deal with physical quantities such as time, distance, pressure, acceleration, etc. However, these systems cannot be fully expressed within π -calculus. π -calculus is not equipped to model concurrent real-time systems or cyber-physical systems (CPS) [11, 7] and reason about their behavior related to time and other physical quantities. Several extensions of π -calculus with time have been proposed to overcome this problem [3, 12, 4, 5]; all these approaches discretize time rather than represent it faithfully as a continuous quantity. Discretizing means that time is represented through finite time-intervals. As a result, *infinitesimally small time-intervals cannot be represented or reasoned about in these approaches*. In practical real-time systems, e.g., a nuclear reactor, two or more events *can* occur within an infinitesimally small interval.

In this paper we consider the extension of π -calculus with continuous time as a powerful formalism for describing concurrent real-time systems and CPS and

reasoning about their behaviors¹. We develop an executable operational semantics of π -calculus in Logic Programming (LP) in which concurrency is modeled by *coroutining* and (rational) infinite computation by *co-induction* [20, 6]. This operational semantics is extended with continuous real-time, which we model with *constraint logic programming over reals* [9]. The executable operational semantics thus realized, automatically leads to an implementation of the timed π -calculus. Note that there is past work on developing LP-based executable operational semantics of the π -calculus (but not timed π -calculus) [22] but it is unable to model infinite processes and infinite replication since co-inductive logic programming is a recently developed concept [20, 6]. Thus, our contributions are twofold: (i) we extend the π -calculus with real-time clocks; in contrast to other extensions, in our work the notion of time and clocks is adopted directly from the well-understood formalism of timed automaton; (ii) we show how an executable operational semantics of the π -calculus can be elegantly realized through logic programming extended with coinduction and coroutining; further adding constraint logic programming over reals to the mix allows this executable operational semantics to capture real-time behavior faithfully as well. This executable semantics allows us to prove behavioral and timing properties of systems modeled as timed π -calculus processes.

The rest of the paper is organized as follows. We first give the motivation for our work. Next, we present the syntax of timed π -calculus and provide its operational semantics. We also develop the notion of timed bisimilarity and investigate its properties. Our method for implementation of timed π -calculus in logic programming is outlined next. In this method, concurrent real-time processes are modeled elegantly as coroutined co-inductive logic programs extended with constraints over reals. We illustrate our method of modeling concurrent real-time processes with the *Generalized Railroad Crossing problem (GRC)* of Lynch and Heitmeyer [8]; for simplicity of presentation we consider only one track. We assume that the reader is familiar with *Constraint Logic Programming over reals (CLP(R))* [9]. An overview of *Co-inductive Logic Programming (Co-LP)* [20, 6] is presented in Appendix A.

2 Motivation

The major application of π -calculus is in expressing the interaction between concurrent, mobile processes. Such processes tend to run on controllers that control physical devices; these processes also communicate with each other and may also be mobile. Therefore, they have to deal with physical quantities such as time, distance, pressure, acceleration, etc. Examples include communicating controller systems in cars (Anti-lock Brake System, Cruise Controllers, Collision Avoidance, etc.), automated manufacturing, smart homes, etc. This class of systems, which are termed cyber-physical systems [11, 7], cannot be modeled

¹ While we only focus on extending the π -calculus with continuous time, our method serves as a model for extending the π -calculus with other continuous quantities. An instance of this, though not in the context of π -calculus, can be found in [18].

within π -calculus. In a real-time/cyber-physical system the correctness of the system behavior depends not only on the tasks that the system is designed to perform, but also on the physical instants at which these tasks are performed. Real-time systems can be described as the class of systems that have to respond to externally generated signals or inputs within specified time limits. Also, the future behavior of such a system depends on the time at which the external signal is received. These type of systems include many safety critical systems such as in robotics and flight control. In most of the cases a real-time system is composed of multiple mobile components that are working concurrently within some time constraints. π -calculus can handle mobility and concurrency, however, it is not equipped to model real-time systems. We extend π -calculus with time so that these systems can be modeled and reasoned about. Several extensions of π -calculus with time have been proposed. Martin Berger has considered extension of π -calculus with time and introduced (π_t -calculus) [3]. Other efforts [10, 14] show how time can be included in π -calculus in the context of web services. Other extensions of process algebras with time have been studied elsewhere [4, 12, 5]. However, time is treated as a discrete entity in these works, resulting in frameworks that cannot model continuous real-time faithfully. In our approach for extending π -calculus with time, time is faithfully modeled as a continuous quantity. The resulting calculus is an expressive way of describing mobile concurrent real-time processes naturally.

3 Timed π -calculus

3.1 Design Decisions

The future behavior of a real-time system depends on not only the content of a receiving input/message (externally generated signal), but also the time-stamp of the message. In other words, the time-stamp of the arriving/sent message should satisfy certain time constraints specified by the system for future transitions to take place. To meet this requirement, we introduce *clocks* and *clock operations*. We also introduce two new variables for carrying the timing information of all the incoming and outgoing messages. The first real-valued variable is the time-stamp of the message; while the second variable is the clock of the process that sends the message. As in π -calculus, timed π -calculus processes use names (including clock names) to interact, and pass names to one another by mentioning them in interactions. These processes are identical to processes in π -calculus except that they have access to clocks which they can manipulate. The idea is for a process to send its clock and also the time-stamp of the message, when it outputs a message in a channel². Thus, messages are represented by triples of the form $\langle m, t_m, c \rangle$, where m is the message, t_m is the time-stamp on m , and c is the clock used to generate t_m . It is important for the process to send its clock (the

² Sending a clock can be implemented in different ways. For example, one can implement this by sending the (global) wall clock time at which the clock was last reset, assuming that all processes have access to the (global) wall clock.

clock that is used to generate the time-stamp of the message), because the time-stamp of the incoming message in conjunction with the receiving clock is used by the receiving process to reason about timing requirements of the system.

We assume an infinite set \mathcal{N} of names (channel names and non-clock names passing through channels) and use x, y, z, u, v, w as metavariables over names. We also assume an infinite set Γ of clock names (disjoint from \mathcal{N}), and an infinite set Θ of variables representing time-stamps (disjoint from \mathcal{N} and Γ). We use c, d, e, f as metavariables over clock names, and $t_x, t_y, t_z, t_u, t_v, t_w$ and sometimes t, t' to represent time-stamps.

Inspired by the notion of name transmission in π -calculus, we can treat time-stamps and clocks just as other names and transmit them on channels. Just as channel transmission results in dynamic configurations of processes, clock and time-stamp transmission can result in dynamic temporal behavior of processes. Note that the clocks are advancing as they are sent among processes and all clocks advance at the same rate. Our notion of real-time and clocks is adopted directly from *timed automata* [1].

3.2 Clocks and Clock Operations

The syntax of the timed π -calculus relies on a set of real-valued clocks and also clock operations. When a transition occurs (transitions are defined formally in section 3.4), some of the clocks may be reset to zero. At any instant, the reading of a clock is equal to the time that has elapsed since the last time the clock was reset. We only consider non-Zeno behaviors, that is, only a finite number of transitions can happen within a finite amount of time. Clock operations C are specified by the following BNF in which C_c is a clock constraint, C_r is a clock reset, c is a clock name in Γ , r is a constant in \mathbf{R}^+ , t is a time-stamp variable in Θ and ϵ represents the empty clock operation.

$$\begin{aligned} C &::= C_c C_r \\ C_c &::= (c \sim r)C_c \mid (c - t \sim r)C_c \mid \epsilon \\ C_r &::= (c := 0)C_r \mid \epsilon \\ \sim &::= < \mid > \mid \leq \mid \geq \mid = \end{aligned}$$

Clock resets are used to remember the time at which particular actions in the system have taken place, while clock constraints indicate the timing constraints between the time-stamps of the various actions that appear in the system. There are multiple ways to measure the time passage while checking for a clock constraint. Time passage can be measured and reasoned about against (i) the last time a clock was reset; this is specified by $(c \sim r)$ in the above BNF. For instance, a constraint $(c < 2)$ on sending message m indicates that m must be sent out within two units of time since the clock c was reset, or (ii) the last time a clock was reset in conjunction with the time-stamp of the arriving message; this is specified by $(c - t \sim r)$. For instance, suppose that a process P sends two consecutive messages that are two units of time apart; if the time-stamp of the first message is t_1 , then P will choose a time t_2 on its clock c such that $c - t_1 = 2$, and sends the second message with the time-stamp t_2 .

A *clock interpretation* \mathcal{I} for a set Γ of clocks assigns a real value to each clock; that is, it is a mapping from Γ to \mathbf{R}^+ . We say that a clock interpretation \mathcal{I} for Γ satisfies a clock constraint C_c over Γ iff C_c evaluates to true using the values given by \mathcal{I} .

3.3 Syntax

The set of timed π -calculus processes is defined as follows (P, P', M , and M' range over timed π -calculus processes):

$$\begin{aligned} M &::= C\bar{x}\langle y, t_y, c \rangle.P \mid Cx(\langle y, t_y, c \rangle).P \mid C\tau.P \mid 0 \mid M + M' \\ P &::= M \mid P \mid P' \mid !P \mid \nu z P \mid [x = y]P \end{aligned}$$

We use $C\bar{x}\langle y, t_y, c \rangle.P$ to stand for a process that sends the name y , its time-stamp t_y , and the clock c via the channel x , after performing the clock operation C , and evolves to P . If the clock operation involves a clock constraint, then the process will send the message if the time constraint is satisfied by the values of clocks at the time of transition, otherwise it will become inactive. If the clock operation involves a clock reset operation, the process will send the message after resetting the clock. For example the expression $(c < 2)(c := 0)\bar{x}\langle y, t_y, c \rangle.P$ indicates that y must be sent out on channel x within two units of time since the clock c was last reset, and moreover the clock c is reset when it is sent. The expression $Cx(\langle y, t_y, c \rangle).P$ stands for a process which is waiting for a message on channel x . When a message arrives, the process will behave like $P\{z/y, t_z/t_y, d/c\}$ (see the definition of substitution below) where z is the content of the message; t_z is the time-stamp of the message; and d is the clock of the sending process that is used to generate t_z . Note that the time-stamp of the incoming message should satisfy the clock constraint expressed by C , otherwise the process will become inactive. The $C\tau.P$ can evolve invisibly to P after performing the clock operation C . Process 0 is *inaction*; it is a process that does nothing. The operators $+$ and \mid are used for nondeterministic *choice* and *composition* of timed π -calculus processes. The replication $!P$, can be thought of as an infinite composition $P \mid P \mid \dots$. The *restriction* $\nu z P$, behaves as P with z local to P , meaning that z cannot be used as a channel over which to communicate with other processes or the environment. $[x = y]P$ evolves to P if x and y are the same name.

In the grammar of timed π -calculus presented above, $\bar{x}\langle y, t_y, c \rangle$, $x(\langle y, t_y, c \rangle)$ and τ are called *prefixes*, represented by α in the rest of the paper. Analogous to π -calculus, processes of the form $\nu y C\bar{x}\langle y, t_y, c \rangle.P$ (when $x \neq y$) contain an irreducible restriction operator in timed π -calculus.

Definition 1 *If $x \neq y$, then $C\bar{x}(\langle y, t_y, c \rangle).P$ means $\nu y C\bar{x}\langle y, t_y, c \rangle.P$, and the prefix $\bar{x}(\langle y, t_y, c \rangle)$ is called a *derived prefix*.*

Note that in the syntax of timed π -calculus, clock operations proceed the prefixes and allow us to reason about timing behavior of real-time systems as well as delays. Note also that empty clock operations can be omitted.

Analogous to π -calculus, restriction and input-prefix in timed π -calculus are name-binding operators of different nature. Restriction specifies the scope of a name, while input-prefix acts as a place-holder for a name which may be received as input. In each process of one of the forms $Cx(\langle y, t_y, c \rangle).P$ and $\nu y P$ the occurrence of y is a binding occurrence, and in each case the scope of the occurrence is P .

Definition 2 *An occurrence of $y \in \mathcal{N}$ in an agent is said to be free if it does not lie within the scope of a binding occurrence of y . The set of names occurring free in P is denoted $fn(P)$. An occurrence of a name $y \in \mathcal{N}$ in a process is said to be bound if it is not free. The set of bound names of P , $bn(P)$, is defined in such a way that it contains all names which occur bound in P . We write $n(P)$ for the set $fn(P) \cup bn(P)$ of names of P .*

Definition 3 *A substitution is a function σ from a set of names \mathcal{N} to \mathcal{N} which is almost everywhere the identity. If $x_i\sigma = y_i$ for all i with $1 \leq i \leq n$ (and $x\sigma = x$ for all other names x), we write $\{y_1/x_1, \dots, y_n/x_n\}$ for σ .*

The effect of applying a substitution σ to a process P is to replace each free occurrence of each name x in P by $x\sigma$, with change of bound names to avoid captures (to preserve the distinction of bound names from the free names just as in λ -calculus). Substitution for clock names and time-stamps can be defined similarly.

Example 1 *The expression $x(\langle m, t_m, c \rangle).(c - t_m > 5)\bar{y}\langle n, t_n, c \rangle$ represents a process that receives a message m on channel x and then sends a message n after 5 units of time since the time m was sent; the process will use the clock c to choose a time t_n on c such that $c - t_m > 5$, and sends n with time-stamp t_n on channel y .*

Example 2 *Consider a system which is composed of two processes P and Q that run in parallel. Moreover, there is a clock c that can be accessed by both P and Q which should be reset before the parallel execution begins. This scenario can be expressed by the expression $(c := 0)\tau.(P \mid Q)$.*

Note that since we are associating time-stamps with all names including channel names, on outputting names, we are able to also reason about the time interval at which a particular channel is used.

3.4 Operational Semantics

In order to understand the behavior of a system expressed by timed π -calculus, it is necessary to define the actions that processes can perform. A transition in timed π -calculus is of the form $P \xrightarrow{\alpha_t} Q$. Intuitively, this transition means that P can evolve into Q , and in doing so performs the *timed action* α_t . The operational semantics of timed π -calculus, represented in Table 1, is expressed as set of transition rules which are labeled by the following timed actions:

$$\alpha_t ::= C_r, \bar{x}\langle y, t_y, c \rangle \mid C_r, x(\langle y, t_y, c \rangle) \mid C_r, \bar{x}(\langle y, t_y, c \rangle) \mid C_r, \langle \tau, t \rangle$$

| | |
|-----------|---|
| TAU | $\frac{[C_c]}{C_c C_r \tau . P \xrightarrow{C_r, \langle \tau, t \rangle} P}$ |
| OUT | $\frac{[C_c]}{C_c C_r \bar{x}(y, t_y, c) . P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P}$ |
| INP | $\frac{[C_c \{d/c\}]}{C_c C_r x(\langle z, t_z, c \rangle) . P \xrightarrow{C_r \{d/c\}, x(\langle y, t_y, d \rangle)} P\{y/z, t_y/t_z, d/c\}} \quad y \notin fn(\nu z P)$ |
| MAT | $\frac{P \xrightarrow{\alpha_t} P'}{[x = x] P \xrightarrow{\alpha_t} P'}$ SUM $\frac{P \xrightarrow{\alpha_t} P'}{P + Q \xrightarrow{\alpha_t} P'}$ |
| PAR | $\frac{P \xrightarrow{\alpha_t} P'}{P \mid Q \xrightarrow{\alpha_t} P' \mid Q} \quad bn(\alpha_t) \cap fn(Q) = \emptyset$ |
| COM | $\frac{P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P' \quad Q \xrightarrow{C'_r, x(\langle z, t_z, d \rangle)} Q'}{P \mid Q \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} P' \mid Q'\{y/z, t_y/t_z, c/d\}}$ |
| CLOSE | $\frac{P \xrightarrow{C_r, \bar{x}(\langle z, t, c \rangle)} P' \quad Q \xrightarrow{C'_r, x(\langle z, t, c \rangle)} Q'}{P \mid Q \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu z(P' \mid Q')} \quad z \notin fn(Q)$ |
| RES | $\frac{P \xrightarrow{\alpha_t} P'}{\nu z P \xrightarrow{\alpha_t} \nu z P'} \quad z \notin n(\alpha_t)$ |
| OPEN | $\frac{P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P'}{\nu y P \xrightarrow{C_r, \bar{x}(\langle y, t_y, c \rangle)} P'} \quad y \neq x$ |
| REP-ACT | $\frac{P \xrightarrow{\alpha_t} P'}{!P \xrightarrow{\alpha_t} P' \mid !P}$ |
| REP-COM | $\frac{P \xrightarrow{C_r, \bar{x}(y, t, c)} P' \quad P \xrightarrow{C'_r, x(\langle y, t, c \rangle)} P''}{!P \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} (P' \mid P'') \mid !P}$ |
| REP-CLOSE | $\frac{P \xrightarrow{C_r, \bar{x}(\langle z, t, c \rangle)} P' \quad P \xrightarrow{C'_r, x(\langle z, t, c \rangle)} P''}{!P \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} (\nu z(P' \mid P'')) \mid !P} \quad z \notin fn(P)$ |

Table 1. Timed π -calculus Transition Rules

The first action is the *free output* action. The transition $P \xrightarrow{C_r, \bar{x}(y, t_y, c)} Q$ means that P resets the clocks expressed by C_r while sending y , its time-stamp t_y ,

and its clock c via x , and evolves to Q . The second action is the *input action*. The transition $P \xrightarrow{C_r, x(\langle y, t_y, c \rangle)} Q$ means that P resets the set of clocks expressed by C_r on receiving any name w with its time-stamp t_w , and a clock d via x and becomes $Q\{w/y, t_w/t_y, d/c\}$. The *bound output action* is expressed by $C_r, \bar{x}(\langle y, t_y, c \rangle)$. The transition $P \xrightarrow{C_r, \bar{x}(\langle y, t_y, c \rangle)} Q$ means that P emits a private name (i.e., a name bound in P) along with its time-stamp and a clock on port x , and becomes Q . The set of clocks that should be reset in this transition is specified by C_r . Bound output actions arise from free output actions which carry names out of their scope, as e.g. in the process $\nu y \bar{x}(\langle y, t_y, c \rangle).P$. The last action is the *silent action*. The transition $P \xrightarrow{C_r, \langle \tau, t \rangle} Q$ expresses that P can take an internal action at time t , while resetting the set of clocks specified by C_r . Silent actions can naturally arise from processes of the form $\tau.P$, but also from communications within a process. Note that C_r can be omitted if in a timed action α_t , the set of clocks to be reset is empty.

The silent and free output actions are called *free* actions, while input actions and bound output actions are called *bound* actions. In the output and input actions mentioned above, x is the *subject* and y is the *object*. The object is said to be *bound* in the bound actions and *free* in the free actions. The set of *bound names* $bn(\alpha_t)$ of an action α_t is the empty set if α_t is a free action; otherwise it contains just the bound object of α_t . The set of *free names* $fn(\alpha_t)$ of α_t contains the subject and free object of α_t , and *names* $n(\alpha_t)$ of α_t is the union of $bn(\alpha_t)$ and $fn(\alpha_t)$. $c(\alpha_t)$ is the set of clock names in action α_t .

Note that the transition of the form $C_c C_r \alpha.Q \xrightarrow{\alpha_t} Q$ in which α is a prefix, takes place only if C_c is satisfied by the values of the clocks at the time of transition³. This is specified as premises of the form $[C_c]$ in the transition rules for *TAU*, *OUT* and *INP* in Table 1. We assume the atomic actions are instantaneous; e.g., if a process Q is ready to send a name $\langle z, t_z, c \rangle$ via channel x , then $x(\langle y, t_y, d \rangle).P$ performs the input action immediately and becomes $P\{z/y, t_z/t_y, c/d\}$ in doing so.

3.5 Timed Bisimulation

We would like to identify two processes which cannot be distinguished by an observer. We assume that the observer is able to observe all kinds of actions and moreover, it can observe the time-stamp of the events.

Definition 4 *The set of clock names in a clock operation C is denoted by $c(C)$.*

Definition 5 *For two clock operations $C = C_c C_r$ and $C' = C'_c C'_r$, in which C_c and C'_c are clock constraints and C_r and C'_r are clock resets; we say $C \models_c C'$ if $c(C_c) = c(C'_c)$, $c(C_r) = c(C'_r)$, and $C_c \models C'_c$ (\models is the standard entailment relation).*

³ We assume that the checking of clock constraints can be done instantaneously, because each constraint is a simple comparison. In practice a real-time system designer should account for time it takes for checking these constraints.

Definition 6 A binary relation \mathcal{S} on timed agents is a (strong) timed simulation if PSQ implies that:

1. If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$, and $P' \mathcal{S} Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
2. If $P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t'_y, c)} Q'$ with premise $[C'_c]$, and $P' \mathcal{S} Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
3. If $P \xrightarrow{C_r, x(\langle y, t_y, c \rangle)} P'$ with premise $[C_c]$ and $y \notin n(P, Q)$, then for some C'_c and Q' , $Q \xrightarrow{C_r, x(\langle y, t'_y, c \rangle)} Q'$ with premise $[C'_c]$, and for all $\langle w, t_w, d \rangle$, $P' \{w/y, t_w/t_y, d/c\} \mathcal{S} Q' \{w/y, t_w/t'_y, d/c\}$, $C_c \models C'_c$ and $C'_c \models C_c$,
4. If $P \xrightarrow{C_r, \bar{x}(\langle y, t_y, c \rangle)} P'$ with premise $[C_c]$ and $y \notin n(P, Q)$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(\langle y, t'_y, c \rangle)} Q'$ with premise $[C'_c]$, and $P' \mathcal{S} Q'$, $C_c \models C'_c$ and $C'_c \models C_c$.

The relation \mathcal{S} is a (strong) timed bisimulation if both \mathcal{S} and its inverse are timed simulation. The relation \sim , (strong) bisimilarity, on timed agents is defined by $P \sim Q$ if and only if there exists a timed bisimulation \mathcal{S} such that PSQ .

Intuitively, two timed processes are strong timed bisimilar if they take the same set of actions and the time-stamp of the actions satisfy the same clock constraints.

Analogous to strong bisimilarity in π -calculus, \sim is not in general preserved by substitutions of names. It follows that strong timed bisimilarity is not preserved by input prefix. A collection of algebraic laws for (strong) timed bisimilarity is presented in Table 2. Note that Propositions 6(d) and (e) in Table 2 are also valid if α is a derived prefix.

Proposition 7 *Expansion*

Let $P \equiv \sum_i C_{ci} C_{ri} \alpha_i . P_i$ and $Q \equiv \sum_j C'_{cj} C'_{rj} \beta_j . Q_j$ where α_i and β_j are prefixes; $bn(\alpha_i) \cap fn(Q) = \emptyset$ for all i and $bn(\beta_j) \cap fn(P) = \emptyset$ for all j ; then

$$P \mid Q \sim \sum_i \sum_j E_i C_{ri} \alpha_i . F_j C'_{rj} \beta_j . (P_i \mid Q_j) + \sum_j \sum_i G_j C'_{rj} \beta_j . H_i C_{ri} \alpha_i . (P_i \mid Q_j) + \sum_{\alpha_i \text{ comp } \beta_j} T C_{ri} C'_{rj} \tau . R_{ij}$$

E_i and F_j are clock constraints that result from actions in which P_i precede Q_j . Similarly, G_j and H_i are clock constraints that result from actions in which Q_j precede P_i . T represents the set of clock constraints that corresponds to τ actions. All these parameters are defined in Table 3 and Table 4 in Appendix B. The relation $\alpha_i \text{ comp } \beta_j$ (α_i complements β_j) holds in the following four cases, which also defines R_{ij} :

1. α_i is $\bar{x}(u, t_u, d)$ and β_j is $x(\langle v, t_v, e \rangle)$; then R_{ij} is $P_i \mid Q_j \{u/v, t_u/t_v, d/e\}$.

Proposition 1 \sim is an equivalence relation.

Proposition 2.a If $P \sim Q$ then

- (a) $P + R \sim Q + R$
- (b) $P \mid R \sim Q \mid R$
- (c) $[x = y]P \sim [x = y]Q$
- (d) $\nu w P \sim \nu w Q$
- (e) $C\tau.P \sim C'\tau.Q$ $C \models_c C', C' \models_c C$
- (f) $C\bar{x}\langle y, t_y, c \rangle.P \sim C'\bar{x}\langle y, t_y, c \rangle.Q$ $C \models_c C', C' \models_c C$

Proposition 2.b If for all $\langle v, t_v, d \rangle$ in which $v \in fn(P) \cup fn(Q) \cup \{y\}$, $P\{v/y, t_v/t_y, d/c\} \sim Q\{v/y, t_v/t_y, d/c\}$ then $Cx(\langle y, t_y, c \rangle).P \sim C'x(\langle y, t_y, c \rangle).Q$ if $C \models_c C'$ and $C' \models_c C$

Proposition 3 Match

- (a) $[x = y]P \sim 0$ if $x \neq y$
- (b) $[x = x]P \sim P$

Proposition 4 Summation

- (a) $P + 0 \sim P$
- (b) $P + P \sim P$
- (c) $P + Q \sim Q + P$
- (d) $P + (Q + R) \sim (P + Q) + R$

Proposition 6 Composition

- (a) $P \mid 0 \sim P$
- (b) $P_1 \mid P_2 \sim P_2 \mid P_1$
- (c) $\nu y P_1 \mid P_2 \sim \nu y (P_1 \mid P_2)$ $y \notin fn(P_2)$
- (d) $(P_1 \mid P_2) \mid P_3 \sim P_1 \mid (P_2 \mid P_3)$
- (e) $\nu y (P_1 \mid P_2) \sim \nu y P_1 \mid \nu y P_2$ $y \notin fn(P_1) \cap fn(P_2)$

Proposition 5 Restriction

- (a) $\nu y P \sim P$ $y \notin fn(P)$
- (b) $\nu x \nu y P \sim \nu y \nu x P$
- (c) $\nu y (P + Q) \sim \nu y P + \nu y Q$
- (d) $\nu y C\alpha.P \sim C\alpha.\nu y P$ $y \notin n(\alpha)$
- (e) $\nu y C\alpha.P \sim 0$ y is the subject of α

Table 2. Timed Bisimilarity Algebraic Laws

2. α_i is $\bar{x}\langle u, t, d \rangle$ and β_j is $x\langle v, t_v, e \rangle$; then R_{ij} is $\nu w (P_i\{w/u\} \mid Q_j\{w/v, t/t_v, d/e\})$, where w is not free in $\nu u P_i$ or in $\nu v Q_j$.
3. α_i is $x\langle v, t_v, e \rangle$ and β_j is $\bar{x}\langle u, t_u, d \rangle$; then R_{ij} is $P_i\{u/v, t_u/t_v, d/e\} \mid Q_j$.
4. α_i is $x\langle v, t_v, e \rangle$ and β_j is $\bar{x}\langle u, t, d \rangle$; then R_{ij} is $\nu w (P_i\{w/v, t/t_v, d/e\} \mid Q_j\{w/u\})$, where w is not free in $\nu v P_i$ or in $\nu u Q_j$.

To illustrate this, the expansion theorem is presented next for a simple case in which all the clock constraints appearing in P and Q are of the form $(c < a_i)$ and $(c < b_j)$ respectively. This case corresponds to the first rows of Tables 3 and 4 in Appendix B.

$$P \mid Q \sim \sum_i \sum_j (c < \min(a_i, b_j)) C_{ri} \alpha_i. (c < b_j) C'_{rj} \beta_j. (P_i \mid Q_j) +$$

$$\sum_j \sum_i (c < \min(a_i, b_j)) C'_{rj} \beta_j. (c < a_i) C_{ri} \alpha_i. (P_i \mid Q_j) +$$

$$\sum_{\alpha_i \text{ comp } \beta_j} (c < \min(a_i, b_j)) C_{ri} C'_{rj} \tau . R_{ij}$$

Example 3 Let $P = (c < 1)\bar{x}\langle z, t_z, c \rangle$ and $Q = (c < 0.5)(c := 0)x(\langle w, t_w, c \rangle)$, then

$$P \mid Q \sim (c < 0.5)\bar{x}\langle z, t_z, c \rangle . (c < 0.5)(c := 0)x(\langle w, t_w, c \rangle) + \\ (c < 0.5)(c := 0)x(\langle w, t_w, c \rangle) . (c < 1 - t_w)\bar{x}\langle z, t_z, c \rangle + (c < 0.5)(c := 0)\tau$$

Proofs of above propositions are extensions of the proofs for untimed π -calculus processes [15] (these extensions take clocks into account) which are not presented here but shown in Appendix B.

Note that analogous to strong bisimilarity in π -calculus, (strong) timed bisimilarity in timed π -calculus is not a congruence since it is not preserved by input prefix.

4 Example

In this section we present the specification of a simplified version of the GRC problem in timed π -calculus. Informally, the GRC problem [8] describes a railroad crossing system with several tracks and an unspecified number of trains traveling through the tracks. The gate at the railroad crossing should be operated (by a controller) in a way that guarantees the *safety* and *utility* properties. The *safety* property stipulates that the gate must be down while one or more trains are in the crossing. The *utility* property states that the gate must be up when there is no train in the crossing. The system is composed of three components: train, controller and gate which communicate by sending and receiving signals. The controller at the railroad crossing might receive various signals from trains in different tracks. In order to avoid receiving mix signals from different trains by the controller, each train communicates through a private channel with the controller. A new channel is established for each approaching train to the crossing area through which the communication between the train and controller takes place. For simplicity of presentation we consider only one track in this example. The components of 1-track GRC are specified via three timed automata in Fig.1.

In our modeling of GRC in timed π -calculus, each component of the system is considered as a timed π -calculus process. The behavior of the system can be seen as a set of concurrent processes which is expressed graphically in Fig.1. Note that the design of 1-track GRC shown in Fig.1 (originally from [1]) does not account for the delay between the sending of the *approach* (*exit*) signal by the *train* to the *controller*. In this original design, it is assumed that both *train* and *controller* receive the *approach* signal at the same time, and the *controller* sends the *lower* signal to the *gate* in exactly one unit of time since it receives the *approach* signal. In contrast, in our specification of GRC in timed π -calculus, we are considering the delays; therefore, all the time-related reasoning in the system are performed

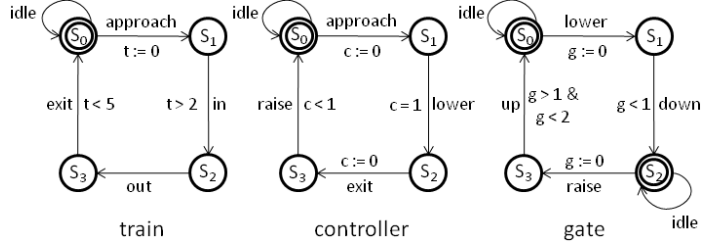


Fig. 1. Timed automata for train, controller, and gate in GRC with one track

against the *train*'s clock and the time-stamp of the *approach* signal (sent by the *train* to the *controller*). The timed π -calculus expressions for processes of 1-track GRC are displayed in Fig.3. Note that in the π -calculus expression for *train*, the two consecutive τ actions correspond to train's internal actions *in* and *out*. Similarly, in the π -calculus expression for *gate*, the first τ represents the gate's internal action *down*; while the second τ corresponds to gate's internal action, *up*.

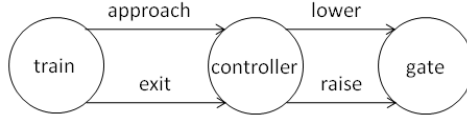


Fig. 2. *train* | *controller* | *gate*

Next we present the implementation of timed π -calculus in LP extended with co-induction and constraints.

5 Operational Semantics in Logic Programming

First, we present the syntax of the language used in our implementation. We use \mathcal{P} to denote the set of all process expressions; \mathcal{PN} to denote the enumerable set of process names; \mathcal{CN} to denote the set of clock names; \mathcal{N} to denote an enumerable set of names distinct from \mathcal{CN} ; \mathcal{R} to denote the set of real-valued numbers; \mathcal{T} to denote the set of time-stamps; \mathcal{C} for clock operations; and \mathcal{A} for set of actions. We use \mathcal{M} of the form $(\mathcal{N}, \mathcal{T}, \mathcal{CN})$ to express messages along with their time-stamps and accompanying clocks. Finally, \mathcal{D} is used to denote the set of process definitions.

$$\mathcal{A} ::= out((\mathcal{C}, \mathcal{N}, \mathcal{M}), \mathcal{P}) \mid in((\mathcal{C}, \mathcal{N}, \mathcal{M}), \mathcal{P}) \mid tau(\mathcal{C}, \mathcal{P}) \mid zero \mid choice(\mathcal{P}, \mathcal{P}) \mid par(\mathcal{P}, \mathcal{P}) \mid rep(\mathcal{P}) \mid nu(\mathcal{N}, \mathcal{P}) \mid match(\mathcal{N} = \mathcal{N}, \mathcal{P})$$

$$\mathcal{C} ::= reset(\mathcal{CN}) \mid const(\mathcal{CN} \sim \mathcal{R}) \mid const(\mathcal{CN} - \mathcal{T} \sim \mathcal{R})$$

$$\mathcal{D} ::= proc(\mathcal{PN}, \mathcal{P})$$

The encoding of the timed π -calculus expressions for GRC components in our implementation is presented as follows.

```

proc(train,                                proc(controller,
  nu(out(ch1, (pc, tp, t)),                in(ch1, (pc, tp, c)),
  in(reset(p), pc, (approach, ta, t)),    in(pc, (x1, t1, c)),
  tau(t > 2),                             out((c=1)(c:=0), ch2, (lower, t1, c)),
  tau(_),                                 in(pc, (x2, t2, c)),
  out((t < 5), pc, (exit, te, t))).      out((c < 1)(c:=0), ch2, (raise, tr, c))).

proc(gate,
  in(ch2, (x, tx, g)),
  choice(match(x=lower, tau(g < 1),
              match(x=raise, tau((g > 1)(g < 2)))))).

```

For a complete encoding of the operational semantics of timed π -calculus, we need to handle the facts that: (i) we are dealing with mobile concurrent processes that are expressed as π -calculus expressions, (ii) clock constraints are posed over continuous time, and (iii) infinite computations are defined in timed π -calculus (and also π -calculus) through the infinite replication operator (!). We have developed an operational semantics of timed π -calculus using logic programming extended with constraints over reals and co-induction, in which channels are modeled as streams. All three aspects can be elegantly handled in this LP framework.

$$\begin{array}{ll}
 \text{train} \equiv \nu \text{pc} \overline{\text{ch1}}\langle \text{pc}, t_p, t \rangle. & \text{controller} \equiv \\
 (t := 0) \overline{\text{pc}}\langle \text{approach}, t_a, t \rangle. & \text{ch1}\langle \text{pc}, t, c \rangle. \text{pc}\langle x_1, t_1, c \rangle. \\
 (t > 2) \tau. & (c = 1)(c := 0) \overline{\text{ch2}}\langle \text{lower}, t_p, c \rangle. \\
 \tau. (t < 5) \overline{\text{pc}}\langle \text{exit}, t_e, t \rangle & \text{pc}\langle x_2, t_2, c \rangle. \\
 & (c - t_2 < 1)(c := 0) \overline{\text{ch2}}\langle \text{raise}, t_r, c \rangle \\
 \\
 \text{gate} \equiv \text{ch2}\langle x, t_x, g \rangle. & \\
 ([x = \text{lower}](g < 1) \tau + & \\
 [x = \text{raise}](g > 1)(g < 2) \tau) & \\
 \\
 \text{main} \equiv \text{rep}(\text{par}(\text{train}, \text{controller}, \text{gate})) &
 \end{array}$$

Fig. 3. The timed π -calculus expressions for the train, controller, and gate

In our LP formulation of the operational semantics of timed π -calculus, clock expressions and time constraints are handled using *constraint logic programming over reals* [9], (rational) infinite computations⁴ are handled using *co-inductive*

⁴ We handle only *rational* infinite computations, where finite number of finite behaviors are repeated infinitely often.

logic programming [20, 6], and finally concurrency is handled by allowing *coroutining* within logic programming computations⁵. Details are not presented due to lack of space. Our operational semantics expressed as a co-inductive coroutined constraint logic program can be regarded as an interpreter for timed π -calculus expressions, and can be used for modeling and verification of real-time systems.

Note that the coroutining feature of LP deals with having LP goals scheduled for execution as soon as some conditions are fulfilled. In LP the most commonly used condition is the instantiation (binding) of a variable. Scheduling a goal to be executed immediately after a variable is bound can be used to model the actions taken by the processes as soon as some messages in some particular channels are received.

Once the system is modeled as a co-inductive coroutined CLP(R) program it can be used to verify interesting properties of the system by posing simple queries. Given a property Q to be verified, we specify its negation as a logic program, notQ. If the property Q holds, the query notQ will fail w.r.t. the logic program that models the system. If the query notQ succeeds, the answer provides a counterexample to why the property Q does not hold⁶. We have used our system to verify the safety and utility of the GRC problem and safety for the reactor temperature control system [19].

6 Related/Future Work and Conclusions

Since the π -calculus was proposed by Milner et al. [15], many researchers have extended it for modeling distributed real-time systems. Berger has introduced timed π -calculus (π_t -calculus) [3]; asynchronous π -calculus with timers and a notion of *discrete* time, locations, and message failure; and explored some of its basic properties. Carlos Olarte has studied temporal CCP (tcc) as a model of concurrency for mobile, timed reactive systems in his Ph.D thesis [16]. He has developed a process calculus called utcc, Universal Temporal CCP. His work can be seen as adding mobile operation to the tcc. In utcc, like tcc, time is conceptually divided into time intervals (or time units); therefore it is discretized. Lee et al. [12] introduced another timed extension of π -calculus called real-time π -calculus (π RT-calculus). They have introduced the time-out operator and considered a global clock, single observer as part of their design, as is common in other (static) real-time process algebras. They have used the set of natural numbers as the time domain, i.e., time is *discrete* and is strictly increasing. Ciobanu et al. [4] have introduced and studied a model called timed distributed π -calculus in which they have considered timers for channels, by which they restrict access to channels. They use *decreasing* timers, and time is discretized in their approach also. Many other timed calculi have similar constructs which also discretize time [5, 10, 14].

⁵ Coroutining can be practically realized through delay/freeze construct supported in most Prolog [21] systems.

⁶ In LP the entire state space can be traversed via backtracking. This feature of LP facilitates verification.

In summary, all these approaches share some common features; they use a *discrete* time-stepping function or timers to increase/decrease the time-stamps after every action (they assume that every action takes exactly one unit of time). *In contrast, our approach for extending π -calculus with time faithfully treats time as continuous.*

Rounds et al. [17] have proposed Φ -calculus as a hybrid extension of π -calculus, while we are focused on extending the π -calculus with real-time. They have proposed the notion of strong bisimilarity; however, their notion of bisimilarity does not take into account time or other continuous quantities. In contrast, we fully develop the notion of (*strong*) *timed bisimilarity*. We have also developed the executable operational semantics of timed π -calculus which is missing from their work.

The work in [23] shows how to introduce time into Milner's CCS to model real time systems. An extra variable t is introduced which records the time delay before a message on some channel α is available, and also a timer for calculating delays. The idea is to use delay operators to suspend activities. In our opinion, it is much harder to specify real-time systems using delays. Our approach provides a more direct way of modeling time in π -calculus via stop watches, and can also elegantly reason about delays.

With respect to the future work, in our extension of π -calculus with time presented in this paper, the notion of clock passing is defined similar to the notion of name passing in Milner's π -calculus. However, we do not make any distinction between clock bound names and clock free names. In other words, all the clock names appear free in the processes. This can be interpreted as having all the clocks as global clocks. While this design decision does not affect on power and expressiveness of the language, we would like to extend our current work so that free clock names can be distinguished from the bound clock names. This way we would be able to simulate local clocks as well as global clocks. Future work also includes developing of the complete implementation based on logic programming and making it available to other researchers, as well as trying out more advanced applications. We also plan to extend the timed π -calculus to include other continuous quantities and use it for modeling and reasoning about cyber-physical systems [18].

To conclude, a combination of constraint over reals, co-induction, and corouting facility of logic programming provides an expressive and easy-to-use framework for implementing the π -calculus extended with real-time via clocks, which then can be used for elegant modeling of complex real-time systems and cyber-physical systems and verifying their interesting properties.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* 126(2), 183–235 (1994)
2. Barwise, J., Moss, L.: *Vicious circles: on the mathematics of non-wellfounded phenomena*. Center for the Study of Language and Information, Stanford, CA, USA (1996)

3. Berger, M.: Towards abstractions for distributed systems. Tech. rep. (2004)
4. Ciobanu, G., Prisacariu, C.: Timers for distributed systems. *Electr. Notes Theor. Comput. Sci.* 164(3), 81–99 (2006)
5. Degano, P., Vincent Loddo, J., Priami, C.: Mobile processes with local clocks. In: LOMAPS. pp. 296–319. Springer-Verlag (1996)
6. Gupta, G., Bansal, A., Min, R., Simon, L., Mallya, A.: Coinductive logic programming and its applications. In: ICLP. pp. 27–44. Springer (2007)
7. Gupta, R.: Programming models and methods for spatiotemporal actions and reasoning in cyber-physical systems. In: NSF Workshop on CPS (2006)
8. Heitmeyer, C.L., Lynch, N.A.: The generalized railroad crossing: A case study in formal verification of real-time systems. In: IEEE RTSS. pp. 120–131 (1994)
9. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. *J. Log. Program.* 19/20, 503–581 (1994)
10. Laneve, C., Zavattaro, G.: Foundations of web transactions. pp. 282–298. Springer (2005)
11. Lee, E.A.: Cyber-physical systems: Design challenges. In: ISORC (May 2008)
12. Lee, J.Y., Zic, J.: On modeling real-time mobile processes. *Aust. Comput. Sci. Commun.* 24(1), 139–147 (2002)
13. Lloyd, J.W.: Foundations of logic programming / J.W. Lloyd. Springer, Berlin, New York, 2nd, extended edn. (1987)
14. Mazzara, M.: Timing issues in web services composition. In: EPEW/WS-FM. pp. 287–302 (2005)
15. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, parts i and ii. *Inf. Comput.* 100(1), 1–77 (September 1992), <http://theory.lcs.mit.edu/~iandc/ic92.html>
16. Olate, C.: Universal Temporal Concurrent Constraint Programming. Ph.D. thesis, LIX, Ecole Polytechnique (2009)
17. Rounds, W.C., Song, H.: The phi-calculus: A language for distributed control of reconfigurable embedded systems. In: HSCC. pp. 435–449 (2003)
18. Saeedloei, N., Gupta, G.: A logic-based modeling and verification of cps. In: To appear in Proceedings of International Conference on Cyber-Physical Systems, Work-in-Progress (WiP) session 2011, SIGBED review.
19. Saeedloei, N., Gupta, G.: A logic based model for the reactor temperature control system. Tech. rep., University of Texas at Dallas, <http://www.utdallas.edu/nxs048000/reactor.pdf> (2011)
20. Simon, L., Bansal, A., Mallya, A., Gupta, G.: Co-logic programming: Extending logic programming with coinduction. In: ICALP. pp. 472–483 (2007)
21. Sterling, L., Shapiro, E.: The art of Prolog (2nd ed.): advanced programming techniques. MIT Press, Cambridge, MA, USA (1994)
22. Yang, P., Ramakrishnan, C.R., Smolka, S.A.: A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. In: VMCAI 2003. pp. 116–131
23. Yi, W.: CCS + time = an interleaving model for real time systems. In: ICALP. pp. 217–228. Springer-Verlag New York, Inc. (1991)

Note : Appendices are attached only for reviewer’s convenience, and can be ignored if the reviewr chooses to.

Appendix A: Co-inductive Logic Programming

One difficulty in modeling timed/hybrid automata is that the underlying automaton is an ω -automaton which accepts infinite strings. Standard logic programming (which computes least fixed-points) is not equipped to model automata that accept infinite strings (which belong to the greatest fixed-points). Recently *co-induction* [2] has been introduced into logic programming by Simon et al [6, 20] to overcome this problem. Co-inductive LP can also be used for reasoning about unfounded sets, behavioral properties of (interactive) programs, elegantly proving liveness properties in model checking, type inference in functional programming, etc. [6].

Co-induction is the dual of induction and corresponds to the greatest fixed-point (*gfp*) semantics. Simon et al’s work gives an operational semantics—similar to SLD resolution [21]—for computing the greatest fixed-point of a logic program. This operational semantics (called co-*SLD* resolution) relies on the *co-inductive hypothesis rule* and systematically computes elements of the *gfp* of a program via backtracking. It is briefly described below. The semantics is limited only to *regular proofs*, i.e., those cases where the infinite behavior is obtained by infinite repetition of a finite number of finite behaviors (and, thus, is powerful enough for modeling ω -automata).

In the co-inductive LP (co-LP) paradigm the declarative semantics of the predicate is given in terms of *infinitary Herbrand (or co-Herbrand) universe*, *infinitary Herbrand (or co-Herbrand) base* [13], and *maximal models (computed using greatest fixed-points)* [20]. The operational semantics under co-induction is identical to Prolog’s operational semantics except for the following addition [20]: a predicate call $p(\bar{t})$ succeeds if it unifies with one of its ancestor calls. Thus, every time a call is made, it has to be remembered. This set of ancestor calls constitutes the *co-inductive hypothesis set*. Under co-LP, infinite *rational* answers can be computed, and infinite rational terms are allowed as arguments of predicates. Infinite terms are represented as solutions to unification equations and the occurs check is omitted during the unification process: for example, $X = [1 \mid X]$ represents the binding of X to an infinite list of 1’s. Thus, in co-*SLD* resolution, given a single clause (note the absence of a base case)

$$p([1 \mid X]) :- p(X).$$

the query $?- p(A)$ will succeed in 2 resolution steps with the (infinite) answer:

$$A = [1 \mid A]$$

which is a finite representation of the infinite answer: $A = [1, 1, 1, \dots]$.

An important application of co-inductive LP is in directly representing and verifying properties of Kripke structures and ω -automata (automata that accept infinite strings). Just as automata that accept finite strings can be directly programmed using standard LP, automata that accept infinite strings can be directly represented using co-inductive LP (one merely has to drop the base

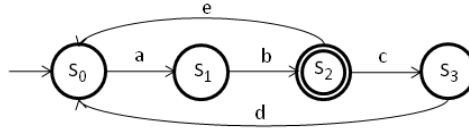


Fig. 4. An Automaton

case). Consider the automata (over finite strings) shown in Figure 4 which is represented by the logic program below.

```

automaton([X | T], St):-
  trans(St, X, NewSt), automaton(T, NewSt).
automaton([], St) :- final(St).
trans(s0, a, s1).      trans(s1, b, s2).
trans(s2, c, s3).      trans(s3, d, s0).
trans(s2, e, s0).      final(s2).
  
```

A call to `?- automaton(X, s0)` in a standard LP system will generate all finite strings accepted by this automaton. Now suppose we want to turn this automaton into an ω -automaton, i.e., it accepts infinite strings (an infinite string is accepted if states designated as final state are traversed an infinite number of times), then the (co-inductive) logic program that simulates this automaton can be obtained by simply dropping the base case (for simplicity, we'll ignore the requirement that final states occur infinitely often; but it can be easily checked by using a co-inductive version of the standard `member` predicate (called `co-member`) [6])

```

automaton([X | T], St) :-
  trans(St, X, NewSt), automaton(T, NewSt).
  
```

Under co-inductive semantics, posing the query `?- automaton(X, s0).` will yield the cyclical solutions:

```
X = [a, b, c, d | X];
X = [a, b, e | X];
```

This feature of co-inductive LP can be leveraged to modeling and verifying properties of (timed/hybrid) ω -automata directly and elegantly.

Appendix B: Proofs of Algebraic Laws for Timed Bisimilarity

In this section we present the proofs of the Propositions stated in Section 3.5. We first give a series of fundamental lemmas and definitions which underpin many later results.

Definition 7 *The symbol \equiv_α denotes the relation of α -convertibility on agents defined in the standard way. (The subscript α here bears no relation to the actions α defined earlier in the paper.)*

Lemma 1 *If $P \xrightarrow{\alpha_t} P'$ with premise $[C_c]$, then (i) $fn(\alpha_t) \subseteq fn(P)$ and (ii) $fn(P') \subseteq fn(P) \cup bn(\alpha_t)$.*

Proof: The proof is by induction on depth of inference. We consider in turn each transition rule as the last rule applied in the inference of the antecedent $P \xrightarrow{\alpha_t} P'$. We give two cases.

(INP) Then $\alpha_t = C_r, x(y, t_y, c)$ and $P \equiv C_c C_r x(z, t_z, d). P_1$ with $y \notin fn(\nu z P_1)$ and $P' \equiv P_1\{y/z, t_y/t_z, c/d\}$, so (i) holds and (ii) $fn(P') \subseteq (fn(P_1) - \{z\}) \cup \{y\} \subseteq fn(P) \cup \{y\}$.

(CLOSE) Then $\alpha_t = C_r, C'_r, \langle \tau, t \rangle$ and $P \equiv P_1 \mid P_2$ with $P_1 \xrightarrow{C_r, \bar{x}(\langle y, t, c \rangle)} P'_1$, $P_2 \xrightarrow{C'_r, x(\langle y, t, c \rangle)} P'_2$ and $P' \equiv \nu y(P'_1 \mid P'_2)$, so (i) holds, and $fn(P'_1) \subseteq fn(P_1) \cup \{y\}$ and $fn(P'_2) \subseteq fn(P_2) \cup \{y\}$, so $fn(P') = (fn(P'_1) \cup fn(P'_2)) - \{y\} \subseteq fn(P)$.

Definition 8 *In the following lemmas the phrase:*

if $P \xrightarrow{\alpha_t} P'$ then equally $Q \xrightarrow{\alpha'_t} Q'$

means that if $P \xrightarrow{\alpha_t} P'$ may be inferred from the transition rules then so, by an inference of no greater depth, may be $Q \xrightarrow{\alpha'_t} Q'$.

Lemma 2 *Suppose that $P \xrightarrow{C_r, a(\langle y, t, c \rangle)} P'$ with premise $[C_c]$, where $a = x$ or $a = \bar{x}$ and that $z \in \mathcal{N}$ and $z \notin n(P)$. Then equally for some $P'' \equiv_\alpha P'\{z/y\}$, $P \xrightarrow{C_r, a(\langle z, t, c \rangle)} P''$ with premise $[C_c]$.*

Proof: By induction on depth of inference, similar to proof of Lemma 1. Similar proof can be done for t and c (not shown here).

Definition 9 *If α_t is an action and σ is a substitution then $\alpha_t \sigma$ is defined as follows:*

$$\begin{aligned} (C_r, \bar{x}(\langle y, t, c \rangle))\sigma &= C_r \sigma, \bar{x} \sigma(\langle y \sigma, t \sigma, c \sigma \rangle) \\ (C_r, \langle \tau, t \rangle)\sigma &= C_r \sigma, \langle \tau, t \sigma \rangle \\ (C_r, a(\langle y, t, c \rangle))\sigma &= C_r \sigma, a \sigma(\langle y, t \sigma, c \sigma \rangle) \text{ if } a = x \text{ or } a = \bar{x} \end{aligned}$$

Lemma 3 *If $P\{w/z\} \xrightarrow{\alpha_t} P'$ where $w \in \mathcal{N}$ and $w \notin fn(P)$ and $bn(\alpha_t) \cap fn(P, w) = \emptyset$, then equally for some Q and β_t with $Q\{w/z\} \equiv_\alpha P'$ and $\beta_t \sigma = \alpha_t$, $P \xrightarrow{\beta_t} Q$.*

Proof: By induction on depth of inference.

Lemma 4 *If $P \dot{\sim} Q$ and $w \in \mathcal{N}$, $w \notin fn(P, Q)$, and $z \in n(P)$ (z is not a clock name nor a time-stamp variable), then $P\{w/z\} \dot{\sim} Q\{w/z\}$.*

Proof: Let $\mathcal{S} = \bigcup_{n < \omega} \mathcal{S}_n$ where

$$\mathcal{S}_0 = \dot{\sim}$$

$$\mathcal{S}_{n+1} = \{(P\{w/z\}, Q\{w/z\}) \mid PS_n Q, w \notin fn(P, Q)\}$$

We show that \mathcal{S} is a strong timed bisimulation by showing by induction on n that if $PS_n Q$ then

1. If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$ and $P'SQ', C_c \models C'_c$ and $C'_c \models C_c$,
2. If $P \xrightarrow{C_r, \bar{x}(y, t, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t', c)} Q'$ with premise $[C'_c]$ and $P'SQ', C_c \models C'_c$ and $C'_c \models C_c$,
3. If $y \in \mathcal{N}, y \notin n(P, Q)$ and $P \xrightarrow{C_r, x(\langle y, t, c \rangle)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, x(\langle y, t', c \rangle)} Q'$ with premise $[C'_c]$ and for all $\langle v, t_v, d \rangle$, $P'\{v/y, t_v/t, d/c\}SQ'\{v/y, t_v/t', d/c\}$, $C_c \models C'_c$ and $C'_c \models C_c$,
4. If $y \in \mathcal{N}, y \notin n(P, Q)$ and $P \xrightarrow{C_r, \bar{x}(\langle y, t, c \rangle)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(\langle y, t', c \rangle)} Q'$ with premise $[C'_c]$ and $P'SQ', C_c \models C'_c$ and $C'_c \models C_c$.

If $n = 0$ then 1,2,3 and 4 hold since $\mathcal{S}_0 = \dot{\sim}$.

Suppose $n > 0$ and that $P\sigma S_n Q\sigma$ where $PS_{n-1}Q$ and $\sigma = \{w/z\}$ where $w \in \mathcal{N}$ and $w \notin fn(P, Q)$. We consider only 4.

Suppose that $P\sigma \xrightarrow{C_r, \bar{x}(\langle y, t, c \rangle)} P'$ with premise $[C_c\sigma]$, where $y \notin fn(P\sigma, Q\sigma)$. Note that since z is not a clock name nor a time-stamp variable, $C_c\sigma = C_c$. Choose $y' \in \mathcal{N}, y' \notin n(P, Q, w, z)$. Then $P\sigma \xrightarrow{C_r, \bar{x}(\langle y', t, c \rangle)} P'' \equiv P'\{y'/y\}$ with premise $[C_c]$. Hence by Lemma 3 for some P''' and x' with $P'''\sigma \equiv P''$ and $x'\sigma = x$, $P \xrightarrow{C_r, \bar{x}'(\langle y', t, c \rangle)} P'''$ with premise $[C_c]$. Since $PS_{n-1}Q$ and $y' \notin n(P, Q)$ for some Q''' , $Q \xrightarrow{C_r, \bar{x}'(\langle y', t, c \rangle)} Q'''$ with premise $[C'_c]$, and $P'''\mathcal{S}Q'''$. Hence $Q\sigma \xrightarrow{C_r, \bar{x}(\langle y', t, c \rangle)} Q'' \equiv Q'\sigma$ with premise $[C'_c\sigma] = [C'_c]$ (because $z \notin C'_c$), and so $Q\sigma \xrightarrow{C_r, \bar{x}(\langle y, t, c \rangle)} Q' \equiv Q''\{y/y'\}$ with premise $[C'_c]$. Then $P' \equiv P'''\{w/z\}\{y/y'\}\mathcal{S} Q'''\{w/z\}\{y/y'\}$ since $y \notin fn(P'''\{w/z\}, Q'''\{w/z\}) \equiv Q'$

Definition 10 *A relation \mathcal{S} is a strong timed simulation up to restriction iff whenever PSQ then*

1. If $w \in \mathcal{N}, w \notin fn(P, Q)$ then $P\{w/z\}SQ\{w/z\}$, and

2. (a) If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$ and either $P'SQ'$ or for some P'', Q'' and w , $P' \equiv \nu w P''$, $Q' \equiv \nu w Q''$ and $P''SQ''$ and $C_c \models C'_c$ and $C'_c \models C_c$,
- (b) If $P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t'_y, c)} Q'$ with premise $[C'_c]$ and $P'SQ'$, $C_c \models C'_c$ and $C'_c \models C_c$,
- (c) If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, x(y, t_y, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, x(y, t'_y, c)} Q'$ with premise $[C'_c]$ and for all $\langle v, t_v, d \rangle$, $P'\{v/y, t_v/t_y, d/c\}SQ'\{v/y, t_v/t'_y, d/c\}$, $C_c \models C'_c$ and $C'_c \models C_c$,
- (d) If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t'_y, c)} Q'$ with premise $[C'_c]$ and $P'SQ'$, $C_c \models C'_c$ and $C'_c \models C_c$.

A relation \mathcal{S} is a strong timed bisimulation up to restriction iff both \mathcal{S} and its inverse are strong timed simulation up to restriction.

Lemma 5 If \mathcal{S} is a strong timed bisimulation up to restriction then $\mathcal{S} \subseteq \sim$.

Proof: Let $\mathcal{S}^* = \bigcup_{n < \omega} \mathcal{S}_n$ where

$$\mathcal{S}_0 = \mathcal{S}$$

$$\mathcal{S}_{n+1} = \{(\nu w P, \nu w Q) \mid P \mathcal{S}_n Q, w \in \mathcal{N}\}$$

We show that \mathcal{S}^* is a strong timed bisimulation. We show that by induction on n , if $P \mathcal{S}_n Q$ and $w \notin fn(P, Q)$, then $P\{w/z\} \mathcal{S}_n Q\{w/z\}$. For $n = 0$ this is immediate from the definition. Suppose $n > 0$ and $\nu v P \mathcal{S}_n \nu v Q$ where $P \mathcal{S}_{n-1} Q$ and $w \notin fn(\nu v P, \nu v Q)$. Then $(\nu v P)\{w/z\} \equiv (\nu u P)\{u/v\}\{w/z\}$ and $(\nu v Q)\{w/z\} \equiv (\nu u Q)\{u/v\}\{w/z\}$ where $u \notin fn(\nu v P, \nu v Q, w)$ and $u\{w/z\} = u$, so $(\nu v P)\{w/z\} \mathcal{S}_n (\nu v Q)\{w/z\}$.

Next we show by induction on n that if $P \mathcal{S}_n Q$ then

1. If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$ and $P'S^*Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
2. If $P \xrightarrow{C_r, \bar{x}(y, t, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t', c)} Q'$ with premise $[C'_c]$ and $P'S^*Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
3. If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, x(y, t, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, x(y, t', c)} Q'$ with premise $[C'_c]$ and for all $\langle v, t_v, d \rangle$, $P'\{v/y, t_v/t, d/c\}S^*Q'\{v/y, t_v/t', d/c\}$, $C_c \models C'_c$ and $C'_c \models C_c$,
4. If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, \bar{x}(y, t, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t', c)} Q'$ with premise $[C'_c]$ and $P'S^*Q'$, $C_c \models C'_c$ and $C'_c \models C_c$.

For $n = 0$ this is immediate from the fact that \mathcal{S}_0 is a strong timed bisimulation up to restriction and the definition of \mathcal{S}^* . The remaining details are omitted.

Proof of Proposition 1: Reflexivity and symmetry are obvious. For transitivity it suffices to show that $\sim\sim$ is a strong timed bisimulation. The proof uses Lemma 2. We give one case.

Suppose that $y \in \mathcal{N}, y \notin n(P, R)$ and $P \xrightarrow{C_r, x(\langle y, t, c \rangle)} P'$ with premise $[C_c]$. Choose $z \in \mathcal{N}, z \notin n(P, Q, R)$. Then $P \xrightarrow{C_r, x(\langle z, t, c \rangle)} P'' \equiv P'z/y$ with premise $[C_c]$ (note that $y \notin c(C_c)$), so for some $Q', Q \xrightarrow{C_r, x(\langle z, t', c \rangle)} Q'$ with premise $[C'_c]$ such that $C_c \models C'_c, C'_c \models C_c$, and for all $w, P''\{w/z\} \sim Q'\{w/z\}$. Hence for some $R', R \xrightarrow{C_r, x(\langle z, t'', c \rangle)} R'$ with premise $[C''_c]$ such that $C'_c \models C''_c, C''_c \models C'_c$ and for all $w, Q'\{w/z\} \sim R'\{w/z\}$. Then $R \xrightarrow{C_r, x(\langle y, t'', c \rangle)} R'' \equiv R'\{y/z\}$ with premise $[C''_c]$ and for all $w, P'\{w/y\} \sim\sim R''\{w/y\}$.

Proof of Proposition 2.a:

- (a) $\{(P + R, Q + R) \mid P \sim Q\} \cup \sim$ is a strong timed bisimulation.
- (b) Let $\mathcal{S} = \{(P \mid R, Q \mid R) \mid P \sim Q\}$. It suffices by Lemma 5 to show that \mathcal{S} is a strong timed bisimulation up to restriction. To see this note first that if $P \sim Q$ and $w \in \mathcal{N}, w \notin fn(P, Q)$ then by Lemma 4, $P\{w/z\} \sim Q\{w/z\}$ and so $(P \mid R)\{w/z\} \mathcal{S} (Q \mid R)\{w/z\}$. It is routine to check that the clauses concerning transitions hold. The only rules applicable are PAR, COM and CLOSE.
- (c) $\{([x = y]P, [x = y]Q) \mid P \sim Q\} \cup \sim$ is a strong timed bisimulation.
- (d) It follows from Lemma 4 that \sim is a strong timed bisimulation up to restriction. Hence by the proof of Lemma 5, if $P \sim Q$ then $\nu w P \sim \nu w Q$.
- (e) $\{(C\tau.P, C'\tau.Q) \mid P \sim Q, C \models_c C', C' \models_c C\} \cup \sim$ is a strong timed bisimulation.
- (f) $\{(C\bar{x}\langle y, t_y, c \rangle.P, C'\bar{x}\langle y, t_y, c \rangle.Q) \mid P \sim Q, C \models_c C', C' \models_c C\} \cup \sim$ is a strong timed bisimulation.

Proof of Proposition 2.b: Note that $\{(Cx(\langle y, t_y, c \rangle).P, C'x(\langle y, t_y, c \rangle).Q) \mid$ for all $\langle w, t_w, e \rangle$ in which $w \in fn(P, Q, y), P\{w/y, t_w/t_y, e/c\} \sim Q\{w/y, t_w/t_y, e/c\}$ and $C \models_c C', C' \models_c C\}$ is a strong timed bisimulation.

Proof of Proposition 3: We can prove the following relations to be strong timed bisimulations:

$$\mathcal{S}_a = \{([x = y]P_1, 0) \mid P_1 \text{ process}, x \neq y\}$$

$$\mathcal{S}_b = \{([x = x]P_1, P_1) \mid P_1 \text{ process}\} \cup \mathbf{Id}$$

where \mathbf{Id} is the identity on agents.

Proof of Proposition 4: The following relations are easily seen to be strong timed bisimulations:

$$\mathcal{S}_a = \{(P_1 + 0, P_1) \mid P_1 \text{ process}\} \cup \mathbf{Id}$$

$$\mathcal{S}_b = \{(P_1 + P_1, P_1) \mid P_1 \text{ process}\} \cup \mathbf{Id}$$

$$\begin{aligned}\mathcal{S}_c &= \{(P_1 + P_2, P_2 + P_1) \mid P_1, P_2 \text{ agents}\} \cup \mathbf{Id} \\ \mathcal{S}_d &= \{(P_1 + (P_2 + P_3), (P_1 + P_2) + P_3) \mid P_1, P_2, P_3 \text{ agents}\} \cup \mathbf{Id}\end{aligned}$$

Proof of Proposition 5: It is straightforward to show that the following relations are strong timed bisimulations:

$$\begin{aligned}\mathcal{S}_a &= \{(\nu y P_1, P_1) \mid P_1 \text{ process}, y \notin fn(P_1)\} \\ \mathcal{S}_b &= \{(\nu x \nu y P_1, \nu y \nu x P_1) \mid P_1 \text{ process}\} \cup \mathbf{Id} \\ \mathcal{S}_c &= \{(\nu y (P_1 + P_2), \nu y P_1 + \nu y P_2) \mid P_1, P_2 \text{ agents}\} \cup \mathbf{Id} \\ \mathcal{S}_d &= \{(\nu y C\alpha.P_1, C\alpha.\nu y P_1) \mid P_1 \text{ process}, y \notin n(\alpha)\} \cup \mathbf{Id} \\ \mathcal{S}_e &= \{(\nu y C\alpha.P_1, 0) \mid P_1 \text{ process}, y \text{ is the subject of } \alpha\}\end{aligned}$$

Proof of Proposition 6: The proofs of Proposition 6 (a) and Proposition 6 (b) are straightforward.

Proof of Proposition 6 (c): In the proof we use the idea of a *strong timed bisimulation up to \sim and restriction*. We introduce first the following concept.

Definition 11 A relation \mathcal{S} is a strong timed simulation up to \sim iff whenever PSQ then

1. If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$ and $P' \sim \mathcal{S} \sim Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
2. If $P \xrightarrow{C_r, \bar{x} \langle y, t_y, c \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x} \langle y, t'_y, c \rangle} Q'$ with premise $[C'_c]$ and $P' \sim \mathcal{S} \sim Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
3. If $y \in \mathcal{N}, y \notin n(P, Q)$ and $P \xrightarrow{C_r, x \langle y, t_y, c \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, x \langle y, t'_y, c \rangle} Q'$ with premise $[C'_c]$ and for all $\langle w, t_w, d \rangle$, $P' \{w/y, t_w/t_y, d/c\} \sim \mathcal{S} \sim Q' \{w/y, t_w/t'_y, d/c\}$, $C_c \models C'_c$ and $C'_c \models C_c$,
4. If $y \in \mathcal{N}, y \notin n(P, Q)$ and $P \xrightarrow{C_r, \bar{x} \langle y, t_y, c \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x} \langle y, t'_y, c \rangle} Q'$ with premise $[C'_c]$ and $P' \sim \mathcal{S} \sim Q'$ and $C_c \models C'_c$ and $C'_c \models C_c$.

\mathcal{S} is a strong timed bisimulation up to \sim iff both \mathcal{S} and its inverse are strong timed simulations up to \sim .

Lemma 6 If \mathcal{S} is a strong timed bisimulation up to \sim then $\mathcal{S} \subseteq \sim$.

Proof: Let $\mathcal{S}^* = \bigcup_{n < \omega} \mathcal{S}_n$ where

$$\mathcal{S}_0 = \sim \mathcal{S} \sim$$

$$\mathcal{S}_{n+1} = \{(P\{w/z\}, Q\{w/z\}) \mid P\mathcal{S}_n Q, w \in \mathcal{N}, w \notin fn(P, Q)\}$$

Then by an argument very similar to that in the proof of Lemma 4 it can be shown that \mathcal{S}^* is a strong timed bisimulation. We omit the details.

Definition 12 A relation \mathcal{S} is a strong timed simulation up to \sim and restriction iff whenever PSQ then

1. If $w \in \mathcal{N}$, $w \notin \text{fn}(P, Q)$ then $P\{w/z\}SQ\{w/z\}$,
2. If $P \xrightarrow{C_r, \langle \tau, t \rangle} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \langle \tau, t' \rangle} Q'$ with premise $[C'_c]$ and either $P' \sim \mathcal{S} \sim Q'$ or for some P'' , Q'' and w , $P' \sim \nu w P''$, $Q' \sim \nu w Q''$ and $P''SQ''$, $C_c \models C'_c$ and $C'_c \models C_c$,
3. If $P \xrightarrow{C_r, \bar{x}(y, t_y, c)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(y, t'_y, c)} Q'$ with premise $[C'_c]$ and $P' \sim \mathcal{S} \sim Q'$, $C_c \models C'_c$ and $C'_c \models C_c$,
4. If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, x(\langle y, t_y, c \rangle)} P'$ with premise $[C_c]$, then for some Q' , $Q \xrightarrow{C_r, x(\langle y, t'_y, c \rangle)} Q'$ with premise $[C'_c]$, and for all $\langle w, t_w, d \rangle$, $P'\{w/y, t_w/t_y, d/c\} \sim \mathcal{S} \sim Q'\{w/y, t_w/t'_y, d/c\}$ and $C_c \models C'_c$ and $C'_c \models C_c$,
5. If $y \in \mathcal{N}$, $y \notin n(P, Q)$ and $P \xrightarrow{C_r, \bar{x}(\langle y, t_y, c \rangle)} P'$ with premise $[C_c]$, then for some C'_c and Q' , $Q \xrightarrow{C_r, \bar{x}(\langle y, t'_y, c \rangle)} Q'$ with premise $[C'_c]$ and $P' \sim \mathcal{S} \sim Q'$, $C_c \models C'_c$ and $C'_c \models C_c$.

\mathcal{S} is a strong timed bisimulation up to \sim and restriction iff both \mathcal{S} and its inverse are strong timed simulations up to \sim and restriction.

Lemma 7 If \mathcal{S} is a strong timed bisimulation up to \sim and restriction then $\mathcal{S} \subseteq \dot{\sim}$.

Proof: Let $\mathcal{S}^* = \bigcup_{n < \omega} \mathcal{S}_n$ where

$$\mathcal{S}_0 = \dot{\sim} \mathcal{S} \dot{\sim}$$

$$\mathcal{S}_{n+1} = \dot{\sim} \{(\nu w P, \nu w Q) \mid P \mathcal{S}_n Q, w \in \mathcal{N}\} \dot{\sim}$$

Then by an argument similar to that in the proof of Lemma 5 it can be shown that \mathcal{S}^* is a strong timed bisimulation. We omit the details.

Returning to the main proof of Proposition 6 (c), we prove that the relation $\mathcal{S} = \{(\nu y P_1 \mid P_2, \nu y(P_1 \mid P_2)) \mid P_1, P_2 \text{ agents}, y \notin \text{fn}(P_2)\} \cup \mathbf{Id}$ is a strong timed bisimulation up to \sim and restriction. Thus, for each P and Q such that PSQ and each transition $P \xrightarrow{\alpha_t} P'$, we must find a simulating transition $Q \xrightarrow{\beta_t} Q'$ satisfying the requirements of a strong timed simulation up to restriction and equivalence, and vice versa. Note that in this case α_t and β_t differ only on their time-stamps. Clearly, if $P \equiv Q$ this is trivial, so we assume that $P \equiv \nu y P_1 \mid P_2$, $Q \equiv \nu y(P_1 \mid P_2)$, and $y \notin \text{fn}(P_2)$.

The proof that there always exists an appropriate transition $Q \equiv \nu y(P_1 \mid P_2) \xrightarrow{\beta_t} Q'$ is by a case analysis on how the transition $P \equiv \nu y P_1 \mid P_2 \xrightarrow{\alpha_t} P'$ is derived, and vice versa. There are sixteen cases in all from which we draw a sample of two.

For each case the derivations of transitions from P and Q are presented in the following way:

We then have to prove three things:

(\Downarrow): that the premises of the upper derivation imply the premises of the lower derivation;

$$\frac{\vdots}{\nu y P_1 \mid P_2 \xrightarrow{\alpha_t} P'}$$

$$\Downarrow$$

$$\frac{\vdots}{\nu y(P_1 \mid P_2) \xrightarrow{\alpha_t} Q'}$$

(\Uparrow): conversely that the premises of the lower derivation imply the premises of the upper derivation;

(\mathcal{S}): that the derivatives P' and Q' satisfy the requirement of a strong timed bisimulation up to \sim and restriction.

Note that by the definition of strong timed simulation we only have to consider α_t such that $y \notin \text{bn}(\alpha_t)$, since y occurs in the agents P and Q .

Case:

$$\text{RES: } \frac{P_1 \xrightarrow{C_r, x(\langle z, t, c \rangle)} P'_1 \quad x, z \neq y}{\nu y P_1 \xrightarrow{C_r, x(\langle z, t, c \rangle)} \nu y P'_1} \quad P_2 \xrightarrow{C'_r, \bar{x}(\langle z, t, c \rangle)} P'_2$$

$$\text{COM: } \frac{\nu y P_1 \mid P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y P'_1 \mid P'_2}{\nu y P_1 \mid P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y P'_1 \mid P'_2}$$

$$\Downarrow$$

$$\text{COM: } \frac{P_1 \xrightarrow{C_r, x(\langle z, t, c \rangle)} P'_1 \quad P_2 \xrightarrow{C'_r, \bar{x}(\langle z, t, c \rangle)} P'_2}{P_1 \mid P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} P'_1 \mid P'_2}$$

$$\text{RES: } \frac{P_1 \mid P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} P'_1 \mid P'_2}{\nu y(P_1 \mid P_2) \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y(P'_1 \mid P'_2)}$$

(\Downarrow): Trivial.

(\Uparrow): From $y \notin \text{fn}(P_2)$ and Lemma 1 we get that $x \neq y$. We cannot prove that $z \neq y$, but if $z = y$ then we use a fresh z' instead of z to get a simulating transition as follows: from Lemma 2 we get that $P_1 \xrightarrow{C_r, x(\langle z', t, c \rangle)} P'_1\{z'/y\}$. The simulating transition then is :

$$\nu y P_1 \mid P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y P'_1\{z'/y\} \mid P'_2 \quad (*)$$

(\mathcal{S}): From Lemma 1 with $y \notin \text{fn}(P_2)$ we get that $y \notin \text{fn}(P'_2)$, so

$$\nu y P'_1 \mid P'_2 \mathcal{S} \nu y(P'_1 \mid P'_2)$$

as required. For the simulating transition (*) we know that $z = y$, so it holds (since z' is chosen fresh) that

$$\nu y P_1\{z'/y\} \mid P'_2 \equiv \nu y P'_1 \mid P'_2 \mathcal{S} \nu y(P'_1 \mid P'_2)$$

Case:

$$\begin{array}{c}
\text{RES: } \frac{P_1 \xrightarrow{C_r, \bar{x}(z, t, c)} P'_1 \quad x \neq y}{\nu y P_1 \xrightarrow{C_r, \bar{x}(z, t, c)} \nu y P'_1} \quad P_2 \xrightarrow{C'_r, x((z, t, c))} P'_2 \\
\text{COM: } \frac{\nu y P_1 | P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y P'_1 | P'_2}{\nu y P_1 | P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y P'_1 | P'_2} \\
\Downarrow \\
\text{COM: } \frac{P_1 \xrightarrow{C_r, \bar{x}(z, t, c)} P'_1 \quad P_2 \xrightarrow{C'_r, x((z, t, c))} P'_2}{P_1 | P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} P'_1 | P'_2} \\
\text{RES: } \frac{P_1 | P_2 \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} P'_1 | P'_2}{\nu y(P_1 | P_2) \xrightarrow{C_r, C'_r, \langle \tau, t \rangle} \nu y(P'_1 | P'_2)}
\end{array}$$

(\Downarrow): Trivial.

(\Uparrow): From Lemma 1 and $y \notin \text{fn}(P_2)$ we get $x \neq y$.

(\mathcal{S}): From Lemma 1 and $y \notin \text{fn}(P_2)$ we get that $y = z$ or $y \notin \text{fn}(P'_2)$. This proves as required

$$\nu y P'_1 | P'_2 \mathcal{S} \nu y(P'_1 | P'_2)$$

Proof of Proposition 6 (d): The proof involves showing that the relation

$$\mathcal{S} = \{((P_1 | P_2) | P_3, P_1 | (P_2 | P_3)) | P_1, P_2, P_3 \text{ agents}\}$$

is a strong timed bisimulation up to \sim and restriction. Thus, for each P and Q such that PSQ and each transition $P \xrightarrow{\alpha_t} P'$ we must find a simulating transition $Q \xrightarrow{\beta_t} Q'$ satisfying the requirements of a strong timed simulation up to \sim and restriction, and vice versa. Note that in this case α_t and β_t differ only on their time-stamps.

The proof that there always exists an appropriate transition $Q \xrightarrow{\beta_t} Q'$ is by a case analysis on how the transition $P \xrightarrow{\alpha_t} P'$ is derived, and vice versa. There are 30 cases in total. We present one sample case in the same style as in the proof of Proposition 6 (c).

Case:

(\Downarrow): By Lemma 2 there exists a fresh z' such that $P_1 \xrightarrow{C_r, \bar{x}(\langle z', t, c \rangle)} P'_1\{z'/z\}$ and $P_2 \xrightarrow{C'_r, x(\langle z', t, c \rangle)} P'_2\{z'/z\}$.

(\mathcal{S}): Note that z' is a fresh name. By alpha-converting z to z' and then applying Proposition 6 (c) we get that $\nu z(P'_1 | P'_2) | P_3 \equiv \nu z'(P'_1\{z'/z\} | P'_2\{z'/z\}) | P_3 \sim \nu z'((P'_1\{z'/z\} | P'_2\{z'/z\}) | P_3)$

so the condition for a timed simulation up to \sim and restriction is satisfied:

$$(P'_1\{z'/z\} | P'_2\{z'/z\}) | P_3 \mathcal{S} P'_1\{z'/z\} | (P'_2\{z'/z\} | P_3)$$

$$\begin{array}{c}
\text{CLOSE: } \frac{P_1 \xrightarrow{C_r, \bar{x}((z, t, c))} P'_1 \quad P_2 \xrightarrow{C'_r, x((z, t, c))} P'_2}{P_1 \mid P_2 \xrightarrow{C_r, C'_r, (\tau, t)} \nu z(P'_1 \mid P'_2)} \\
\text{PAR: } \frac{(P_1 \mid P_2) \mid P_3 \xrightarrow{C_r, C'_r, (\tau, t)} \nu z(P'_1 \mid P'_2) \mid P_3}{\downarrow} \\
\text{PAR: } \frac{P_2 \xrightarrow{C'_r, x((z', t, c))} P'_2\{z'/z\} \quad z' \notin fn(P_3)}{P_2 \mid P_3 \xrightarrow{C'_r, x((z', t, c))} P'_2\{z'/z\} \mid P_3} \quad P_1 \xrightarrow{C_r, \bar{x}((z', t, c))} P'_1\{z'/z\} \\
\text{CLOSE: } \frac{P_1 \mid (P_2 \mid P_3) \xrightarrow{C_r, C'_r, (\tau, t)} \nu z'(P'_1\{z'/z\} \mid (P'_2\{z'/z\} \mid P_3))}{}
\end{array}$$

proof of Proposition 6 (e): If $y \notin fn(P_1) \cap fn(P_2)$, then y cannot be free in both P_1 and P_2 . Assume that y is not free in P_2 . Then by Propositions 5 (a) and 6 (c):

$$\nu y(P_1 \mid P_2) \sim \nu y P_1 \mid P_2 \sim \nu y P_1 \mid \nu y P_2$$

The situation when y is not free in P_1 is similar.

Proof of Proposition 8 Expansion Theorem: Write R for the right hand side of the equation. Define the relation \mathcal{S} by

$$\mathcal{S} = \{(P \mid Q), R\} \cup \text{Id}$$

It can be easily seen that \mathcal{S} is a timed bisimulation.

Note that the side conditions $bn(\alpha_i) \cap fn(Q) = \emptyset$ and $bn(\beta_j) \cap fn(P) = \emptyset$ are important, otherwise a bound object in α_i (or β_j) would bind names in Q (or P) in the right hand side but not in the left hand side.

Notations used in Expansion Theorem: Note that two types of clock constraints can appear in clock operations of proposed timed π -calculus: ($Clock \sim x$) and ($Clock - t \sim x$ where $Clock$ is a clock name in Γ , t is a time-stamp variable in Θ and $\sim \in \{<, >, \leq, \geq, =\}$). Since $Clock$ and t are instantiated at the time of inferring transitions, clock constraints of the form ($Clock - t \sim x$) can be simplified to ($Clock \sim x'$), in which $x' = t + x$. Therefore in Table 3 and Table 4 all the clock constraints are of the form ($Clock \sim x$).

For two timed processes $P \equiv \sum_i C_{ci} C_{ri} \alpha_i . P_i$ and $Q \equiv \sum_j C'_{cj} C'_{rj} \beta_j . Q_j$, the clock expressions E_i and F_j used in *expansion theorem* are defined as follows. Clock expressions for E_i s are defined in the fourth column of table 3; while clock expressions for F_j s are defined via expressions F'_j (defined in the fifth column of table 3) as follows:

1. α_i is either $\bar{x}\langle u, t, d \rangle$, or $\bar{x}(\langle u, t, d \rangle)$, or $x(\langle u, t, d \rangle)$ and $C_{ri} = \emptyset$; then $F_j = F'_j$.
2. α_i is either $\bar{x}\langle u, t, d \rangle$, or $\bar{x}(\langle u, t, d \rangle)$, or $x(\langle v, t, d \rangle)$ and $C_{ri} \neq \emptyset$; then $F_j = F'_j\{b_j - t/b_j\}$.

Similarly clock expressions for G_j s are defined in the fourth column of table 4; while clock expressions for H_i s are defined via expressions H'_i (defined in the fifth column of table 4) as follows:

1. β_j is either $\bar{x}\langle u, t, d \rangle$, or $\bar{x}(\langle u, t, d \rangle)$, or $x(\langle u, t, d \rangle)$ and $C'_{rj} = \emptyset$; then $H_i = H'_i$.

| C_{ci} | C'_{cj} | condition | E_i | F'_j | T |
|----------------|----------------|-------------|------------------------|----------------|----------------------------|
| $(c < a_i)$ | $(c < b_j)$ | | $(c < \min(a_i, b_j))$ | $(c < b_j)$ | $(c < \min(a_i, b_j))$ |
| $(c < a_i)$ | $(c = b_j)$ | $a_i < b_j$ | $(c < a_i)$ | $(c = b_j)$ | $(c = b_j)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c = b_j)$ | |
| $(c < a_i)$ | $(c > b_j)$ | $a_i < b_j$ | $(c < a_i)$ | $(c > b_j)$ | $(c < a_i)(c > b_j)$ |
| | | $a_i > b_j$ | $(c < a_i)$ | $(c > b_j)$ | |
| $(c < a_i)$ | $(c \leq b_j)$ | $a_i < b_j$ | $(c < a_i)$ | $(c \leq b_j)$ | $(c < \min(a_i, b_j))$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c \leq b_j)$ | $(c < \min(a_i, b_j))$ |
| $(c < a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c < a_i)$ | $(c \geq b_j)$ | $(c < a_i)(c \geq b_j)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c \geq b_j)$ | |
| $(c > a_i)$ | $(c < b_j)$ | $a_i < b_j$ | $(c > a_i)(c < b_j)$ | $(c < b_j)$ | $(c > a_i)(c < b_j)$ |
| $(c > a_i)$ | $(c = b_j)$ | $a_i < b_j$ | $(c > a_i)(c < b_j)$ | $(c = b_j)$ | $(c = b_j)$ |
| $(c > a_i)$ | $(c > b_j)$ | | $(c > a_i)$ | $(c > b_j)$ | $(c > \max(a_i, b_j))$ |
| $(c > a_i)$ | $(c \leq b_j)$ | $a_i < b_j$ | $(c > a_i)(c < b_j)$ | $(c \leq b_j)$ | $(c > a_i)(c \leq b_j)$ |
| $(c > a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c > a_i)(c < b_j)$ | $(c \geq b_j)$ | $(c \geq b_j)$ |
| | | $a_i > b_j$ | $(c > a_i)$ | $(c \geq a_i)$ | $(c > a_i)$ |
| $(c = a_i)$ | $(c < b_j)$ | $a_i < b_j$ | $(c = a_i)$ | $(c < b_j)$ | $(c = a_i)$ |
| $(c = a_i)$ | $(c = b_j)$ | $a_i < b_j$ | $(c = a_i)$ | $(c = b_j)$ | $(c = a_i)$ |
| | | $a_i = b_j$ | $(c = a_i)$ | $(c = b_j)$ | |
| $(c = a_i)$ | $(c > b_j)$ | $a_i < b_j$ | $(c = a_i)$ | $(c > b_j)$ | $(c = a_i)$ |
| | | $a_i > b_j$ | $(c = a_i)$ | $(c > b_j)$ | |
| $(c = a_i)$ | $(c \leq b_j)$ | $a_i < b_j$ | $(c = a_i)$ | $(c \leq b_j)$ | $(c = a_i)$ |
| $(c = a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c = a_i)$ | $(c \geq b_j)$ | $(c = a_i)$ |
| | | $a_i > b_j$ | $(c = a_i)$ | $(c \geq a_i)$ | |
| $(c \leq a_i)$ | $(c < b_j)$ | $a_i < b_j$ | $(c \leq a_i)$ | $(c < b_j)$ | $(c \leq a_i)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c < b_j)$ | $(c < b_j)$ |
| $(c \leq a_i)$ | $(c = b_j)$ | $a_i < b_j$ | $(c \leq a_i)$ | $(c = b_j)$ | $(c = b_j)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c = b_j)$ | |
| $(c \leq a_i)$ | $(c > b_j)$ | $a_i < b_j$ | $(c \leq a_i)$ | $(c > b_j)$ | $(c \leq a_i)(c > b_j)$ |
| | | $a_i > b_j$ | $(c \leq a_i)$ | $(c > b_j)$ | |
| $(c \leq a_i)$ | $(c \leq b_j)$ | $a_i < b_j$ | $(c \leq a_i)$ | $(c \leq b_j)$ | $(c \leq a_i)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c \leq b_j)$ | $(c \leq a_i)$ |
| $(c \leq a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c \leq a_i)$ | $(c \geq b_j)$ | $(c \leq a_i)(c \geq b_j)$ |
| | | $a_i > b_j$ | $(c < b_j)$ | $(c \geq b_j)$ | |
| $(c \geq a_i)$ | $(c < b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c < b_j)$ | $(c \geq a_i)(c < b_j)$ |
| $(c \geq a_i)$ | $(c = b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c = b_j)$ | $(c = b_j)$ |
| $(c \geq a_i)$ | $(c > b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c > b_j)$ | $(c > b_j)$ |
| | | $a_i < b_j$ | $(c \geq a_i)$ | $(c > b_j)$ | $(c \geq b_j)$ |
| $(c \geq a_i)$ | $(c \leq b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c \leq b_j)$ | $(c \geq a_i)(c \leq b_j)$ |
| | | $a_i > b_j$ | $(c \geq a_i)$ | $(c > b_j)$ | $(c \geq b_j)$ |
| $(c \geq a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c \geq b_j)$ | $(c \geq b_j)$ |
| $(c \geq a_i)$ | $(c \geq b_j)$ | $a_i < b_j$ | $(c \geq a_i)$ | $(c \geq b_j)$ | $(c \geq b_j)$ |
| | | $a_i > b_j$ | $(c \geq a_i)$ | $(c > b_j)$ | $(c \geq b_j)$ |

Table 3. Clock Constraints for Expansion Theorem, P_i s proceed Q_j s

2. β_j is either $\bar{x}\langle u, t, d \rangle$, or $\bar{x}\langle u, t, d \rangle$, or $x\langle u, t, d \rangle$ and $C'_{r_j} \neq \emptyset$; then $H_i = H'_i\{a_i - t/a_i\}$.

| C'_{c_j} | C_{c_i} | condition | G_j | H'_i |
|----------------|----------------|----------------------------|-------------------------------------|----------------------------------|
| $(c < b_j)$ | $(c < a_i)$ | | $(c < \min(b_j, a_i))$ | $(c < a_i)$ |
| $(c < b_j)$ | $(c = a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c < b_j)$ $(c < a_i)$ | $(c = a_i)$ $(c = a_i)$ |
| $(c < b_j)$ | $(c > a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c < b_j)$ $(c < b_j)$ | $(c > a_i)$ $(c > a_i)$ |
| $(c < b_j)$ | $(c \leq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c < b_j)$ $(c < a_i)$ | $(c \leq a_i)$ $(c \leq a_i)$ |
| $(c < b_j)$ | $(c \geq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c < b_j)$ $(c < a_i)$ | $(c \geq a_i)$ $(c \geq a_i)$ |
| $(c > b_j)$ | $(c < a_i)$ | $b_j < a_i$ | $(c > b_j)(c < a_i)$ | $(c < a_i)$ |
| $(c > b_j)$ | $(c = a_i)$ | $b_j < a_i$ | $(c > b_j)(c < a_i)$ | $(c = a_i)$ |
| $(c > b_j)$ | $(c > a_i)$ | | $(c > b_j)$ | $(c > a_i)$ |
| $(c > b_j)$ | $(c \leq a_i)$ | $b_j < a_i$ | $(c > b_j)(c < a_i)$ | $(c \leq a_i)$ |
| $(c > b_j)$ | $(c \geq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c > b_j)(c < a_i)$ $(c > b_j)$ | $(c \geq a_i)$ $(c \geq b_j)$ |
| $(c = b_j)$ | $(c < a_i)$ | $b_j < a_i$ | $(c = b_j)$ | $(c < a_i)$ |
| $(c = b_j)$ | $(c = a_i)$ | $b_j < a_i$ $b_j = a_i$ | $(c = b_j)$ $(c = b_j)$ | $(c = a_i)$ $(c = a_i)$ |
| $(c = b_j)$ | $(c > a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c = b_j)$ $(c = b_j)$ | $(c > a_i)$ $(c > a_i)$ |
| $(c = b_j)$ | $(c \leq a_i)$ | $b_j < a_i$ | $(c = b_j)$ | $(c \leq a_i)$ |
| $(c = b_j)$ | $(c \geq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c = b_j)$ $(c = b_j)$ | $(c \geq a_i)$ $(c \geq b_j)$ |
| $(c \leq b_j)$ | $(c < a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \leq b_j)$ $(c < a_i)$ | $(c < a_i)$ $(c < a_i)$ |
| $(c \leq b_j)$ | $(c = a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \leq b_j)$ $(c < a_i)$ | $(c = a_i)$ $(c = a_i)$ |
| $(c \leq b_j)$ | $(c > a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \leq b_j)$ $(c \leq b_j)$ | $(c > a_i)$ $(c > a_i)$ |
| $(c \leq b_j)$ | $(c \leq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \leq b_j)$ $(c < a_i)$ | $(c \leq a_i)$ $(c \leq a_i)$ |
| $(c \leq b_j)$ | $(c \geq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \leq b_j)$ $(c < a_i)$ | $(c \geq a_i)$ $(c \geq a_i)$ |
| $(c \geq b_j)$ | $(c < a_i)$ | $b_j < a_i$ | $(c \geq b_j)$ | $(c < a_i)$ |
| $(c \geq b_j)$ | $(c = a_i)$ | $b_j < a_i$ | $(c \geq b_j)$ | $(c = a_i)$ |
| $(c \geq b_j)$ | $(c > a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \geq b_j)$ $(c \geq b_j)$ | $(c > a_i)$ $(c > a_i)$ |
| $(c \geq b_j)$ | $(c \leq a_i)$ | $b_j < a_i$ | $(c \geq b_j)$ | $(c \leq a_i)$ |
| $(c \geq b_j)$ | $(c \geq a_i)$ | $b_j < a_i$ $b_j > a_i$ | $(c \geq b_j)$ $(c \geq b_j)$ | $(c \geq a_i)$ $(c > a_i)$ |

Table 4. Clock Constraints for Expansion Theorem, Q_j s proceed P_i s