Under consideration for publication in Formal Aspects of Computing

A Refinement-based Process Algebra for Timed Automata [1]

Stefano Cattani and Marta Kwiatkowska

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
United Kingdom

Abstract. We propose a real-time extension to the process algebra CSP. Inspired by timed automata, a very successful formalism for the specification and verification of real-time systems, we handle real time by means of clocks, i.e. real-valued variables that increase at the same rate as time. This differs from the conventional approach based on timed transitions. We give a discrete trace and failures semantics to our language and define the resulting refinement relations. One advantage of our proposal is that it is possible to automatically verify refinement relations between processes. We demonstrate how this can be achieved and under which conditions.

1. Introduction

The specification and verification of concurrent systems has been a major research topic for more than twenty years. Many approaches have been proposed, and that of process algebras, e.g. CSP [Hoa85] and CCS [Mil89], is undoubtedly one of the most successful. Our work will focus on CSP, which we extend with real-time constructs. The key features of classical CSP are a denotational model based on traces and failures, together with the definition of process equivalence founded on the concept of process refinement. Refinement allows one to establish whether a process implements a specification by checking if every behaviour of the former is allowed by the latter; in this way, implementations can be further refined, allowing for chains of refinements leading towards the final implementation. CSP has been the subject of extensive research and, most notably, has an effective associated software tool, FDR2 [For93], that can automatically verify refinement relations.

Traditional process calculi can only verify functional properties of systems, that is, properties that are not time sensitive. More recently, substantial effort has been directed towards describing systems and their timed behaviour in order to establish their *real-time* properties, consequently extending existing models. Some proposals have been made to extend CSP to describe real-time systems, of which the most important is Timed CSP [RR88, DJR+92]. Much work has been done since its introduction, but its take up has been hampered by the lack of automatic verification algorithms. The main difficulty with Timed CSP, and with

[1] Partially supported by EPSRC grant GR/N22960.

Correspondence and offprint requests to: Stefano Cattani, School of Computer Science, The University of Birmingham, Birmingham B15 2TT, United Kingdom. e-mail: s.cattani@cs.bham.ac.uk.

most real-time systems with a dense representation of time, is that their behaviour is infinite and continuous, making them hard to analyse.

There are two main techniques that have been proposed to verify Timed CSP: *timewise refinement* [Sch97, Sch99] and *digitisation* [Oua01]. The idea behind timewise refinement is to ignore time and to establish only functional properties of a Timed CSP process. This is done by considering an untimed CSP specification and a Timed CSP implementation. It is possible to verify whether the functional behaviour of the implementation refines the specification. This approach is limited because no timed properties can be verified; moreover, it is not possible to have decidable chains of refinements in the same model. More interesting and promising is the work on digitisation: this technique has been known for at least 10 years [HMP92] in the area of timed systems and its main purpose is to identify the conditions under which it is possible to reduce the dense representation of time to a discrete one while preserving the relations among processes. This technique has been extended to Timed CSP [Oua01], making it possible to use FDR2 to verify refinement relations. The main difficulty is that it is, in general, undecidable to know whether digitisation techniques can be applied and chains of refinements are not possible.

In the domain of real-time systems, the most successful approach is arguably that of *timed automata* [AD92]. Timed automata extend traditional labelled transition systems with clocks, real-valued variables that record the passage of time and influence how the system evolves. The success of timed automata is due to the availability of model checking techniques that allow us to verify properties expressed in the logic TCTL [ACD93]. Efficient model checking tools are available, e.g. Uppaal [LPY97] and KRONOS [Yov97]. Some work has been done to relate CSP to timed automata: Jackson's thesis [Jac92] shows how to translate Timed CSP processes into timed automata in order to use timed automata techniques to verify logical properties of processes. More recently, equivalence between Timed CSP and closed timed automata has been proved [OW03], and it has also been shown how to extend digitisation techniques to timed automata in order to use FDR2 to verify refinement of timed traces.

It is possible to give timed automata semantics in terms of timed traces, and to define relations based on trace inclusion: a timed automaton, the specification, is refined by another, the implementation, if the traces of the implementation are a subset of the traces of the specification. Unfortunately, trace inclusion is undecidable in general [AD92]. Despite this result, there has been much work aimed at identifying the conditions under which the trace inclusion problem becomes decidable; apart from the previously mentioned digitisation techniques [HMP92, OW03], it is possible to place restrictions on specifications by allowing only determinisable specifications [AFH99] or specifications with at most one clock [OW04]. However, under these restrictions, it is not possible to have chains of refinements unless the same restrictions are applied to the implementations as well as to the specifications, hence generality is lost.

Our aim is to extend CSP to model timed automata directly. We are aware of alternative extensions of process algebras that model timed automata (e.g. [D'A99, YPD95]), but, to our knowledge, no attempt has been made to extend CSP in this way. Aware of the undecidability results for trace inclusion for timed automata, we take a different approach and employ the successful techniques of timed automata (e.g. the region automaton) to discretise the infinite state space caused by the representation of time in order to define a semantic model and refinement relations in the style of CSP. Firstly, we extend the syntax to describe the timed automata constructs. The operational semantics is a straightforward extension of the usual CSP rules, but defining a denotational semantics is more problematic; the reason for this is that we have to deal with undecidability results (because of the continuous representation of time) and with obstacles to compositionality. We describe how we resolve these difficulties and the inevitable resulting trade-offs. The main result is that it is possible (with some limitations) to use FDR2 to verify timed properties of processes expressed in the extended CSP and refinement relations between them. Moreover, it is possible to verify chains of refinements.

In this paper we present the theoretical foundations of our timed extension of CSP, by giving it a formal semantics and by showing its relationship with timed automata. We also show how to model check timed properties of processes, but more research is necessary in order to achieve a fully automatic procedure for this.

This paper is structured in the following way. In Section 2 we give the background on timed automata and CSP needed to understand the rest of the paper. In Section 3 we introduce the extended language, Clocked CSP, and in Section 4 we give it an operational semantics. The denotational model for traces is described in Section 5, together with a brief overview of its extension to failures. Section 6 describes how it is possible to use Clocked CSP to verify properties of processes with FDR2. Finally, in Section 7 we discuss the advantages and disadvantages of our approach and future work.

2. Preliminaries

2.1. Labelled Transition Systems, Traces and Failures

We give the basic definitions of labelled transition systems that we will need in the remainder of the paper. A labelled transition system (lts) is a tuple $L = (Q, \overline{q}, \Sigma, \rightarrow)$, where Q is the set of states, $\overline{q}$ is the initial state, Σ is the alphabet and $\rightarrow$ is the transition relation, that is, $\rightarrow \subseteq Q \times (\Sigma \cup \{\tau\}) \times Q$. We write $p \xrightarrow{a} q$ whenever$(p, a, q) \in \rightarrow$. The alphabet Σ does not include the special internal action τ; we call all other actions visible. A finite execution for an lts is a sequence of states and actions $q_0 a_1 q_1 \cdots q_n$ such that, for all $i = 0..n-1$, we have $(q_i, a_{i+1}, q_{i+1}) \in \rightarrow$. If $s = a_1 \cdots a_n$ is the sequence of actions of the execution, including internal actions, we write $q_0 \xrightarrow{s} q_n$. Given an execution $q_0 a_1 \cdots q_n$, the corresponding trace is the sequence $t \in \Sigma^*$ of visible actions obtained by removing internal actions from the sequence $a_1, a_2 \cdots a_n$. In this case we say that $q_0 \stackrel{t}{\Longrightarrow} q_n$. Traces are denoted by the corresponding list of actions, i.e. $\langle a_1 \cdots a_n \rangle$. Given two traces $t_1 = \langle a_1 \cdots a_n \rangle$ and $t_2 = \langle b_1 \cdots b_m \rangle$ we define their concatenation $t_1 \hat{\ } t_2$ as $\langle a_1 \cdots a_n b_1 \cdots b_m \rangle$. We sometimes write $a \hat{\ } t$, where $a \in \Sigma$ and $t \in \Sigma^*$, instead of $\langle a \rangle \hat{\ } t$. Given an lts L, its trace semantics is given by $\mathcal{T}(L) = \{t \in \Sigma^* | \exists q. \overline{q} \stackrel{t}{\Longrightarrow} q\}$.

A state q refuses an action a if there is no state q' such that $(q, a, q') \in \rightarrow$. We say that q is stable if there is no τ transition leaving q. We define the refusal set of a stable state q as follows: $q \,\mathsf{ref} F$ if q is stable and q refuses all actions in F. We can define the stable failures of an lts L as $\mathcal{F}(L) = \{(t, F) \mid \exists q. \overline{q} \stackrel{t}{\Longrightarrow} q \text{ and } q \,\mathsf{ref}\, F\}$.

The notions of traces and failures given above for labelled transition systems are the operational equivalent of traces and failures in the CSP denotational semantics.

Given two labelled transition systems $L_1 = (Q_1, \overline{q}_1, \Sigma, \rightarrow_1)$ and $L_2 = (Q_2, \overline{q}_2, \Sigma, \rightarrow_2)$, the parallel composition with respect to the interface alphabet $A \subseteq \Sigma$ is the new labelled transition system $L_1 \underset{A}{||} L_2 = (Q_1 \times Q_2, (\overline{q}_1, \overline{q}_2), \Sigma, \rightarrow)$ with $\rightarrow$ defined as follows: if $a \in A$, then $(p_1, p_2) \xrightarrow{a} (q_1, q_2)$ if $p_1 \xrightarrow{a} q_1$ and $p_2 \xrightarrow{a} q_2$; if $a \notin A$, then $(p_1, p_2) \xrightarrow{a} (q_1, p_2)$ if $p_1 \xrightarrow{a} q_1$ or $(p_1, p_2) \xrightarrow{a} (p_1, q_2)$ if $p_2 \xrightarrow{a} q_2$.

2.2. Timed Automata

Timed automata have become a standard formalism to describe timed systems. They are an extension of traditional labelled transition systems, obtained by augmenting them with clocks and operations on them. We summarise the notions that we are going to use in this paper; most of this is standard (see [AD94]).

Given a set $\mathcal{C}$ of real-valued variables called clocks, the set $\mathcal{B}(\mathcal{C})$ of clock constraints is generated by the grammar: $\phi ::= x \prec c \mid \phi \wedge \phi \mid \neg\phi$, for $x \in \mathcal{C}$, $\prec \in \{<, \leq\}$ and $c \in \mathbb{N}$. Given a set of clocks $\mathcal{C}$, a valuation ν is a function $\nu : \mathcal{C} \rightarrow \mathbb{R}^+$ that assigns a non negative real value to each clock. Given a valuation ν and a non-negative real value d, the valuation $\nu + d$ is defined as $(\nu + d)(x) = \nu(x) + d$ for all clocks $x \in \mathcal{C}$. Given a valuation ν and a set of clocks $X \subseteq \mathcal{C}$, $\nu[X]$ is a new valuation that agrees with ν on all clocks except for those in X, whose value has been set to 0; formally, $\nu[X](x) = 0$ if $x \in X$, $\nu[X](x) = \nu(x)$ otherwise. A clock valuation ν satisfies a constraint ϕ ($\nu \models \phi$) if ϕ evaluates to true when clocks are replaced by their valuation under ν. We say that a constraint ϕ is *past closed* if, for all valuations ν and positive reals d, if $\nu + d \models \phi$ then $\nu \models \phi$, and denote the set of past closed constraints by $\mathcal{B}_c(\mathcal{C})$; informally, past closed constraints denote "until" properties, that is, they only enforce constraints on the upper limits of the values of clocks.

Definition 2.1. A **timed automaton** $\mathcal{A}$ is a tuple $\left(L, \bar{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow\right)$, where L is the set of locations, $\bar{l}$ is the initial location, Σ is the set of actions (or alphabet), $\mathcal{C}$ is the set of clocks, $\mathcal{I} : L \rightarrow \mathcal{B}_c(\mathcal{C})$ is the location invariant function, $\kappa : L \rightarrow 2^{\mathcal{C}}$ is the set of resets and $\rightarrow \subseteq L \times (\Sigma \cup \{\tau\}) \times \mathcal{B}(\mathcal{C}) \times L$ is the set of edges. We write $s \xrightarrow{a,\phi} s'$ whenever $(s, a, \phi, s') \in \rightarrow$.

A timed automaton is given semantics in terms of a labelled transition system. At each point of the computation one must know the location the system is in and the current value of clocks. The state space of the transition system is thus given by the cross product of locations and clock valuations. The semantics of a timed automaton is given by the lts $LTS_{\mathcal{A}} = (Q, \overline{q}, \Sigma \cup \mathbb{R}^+, \rightarrow_{lts})$, defined as follows:

- Q is the set of states. A state is a pair (l, ν) where $l \in L \cup \mathit{free}(L)$ and ν is a clock valuation. $\mathit{free}(L)$

Action	$\dfrac{l \xrightarrow{a,\phi} l' \quad \nu \models \phi \quad l \in \mathit{free}(L)}{(l,\nu) \xrightarrow{a}_{lts} (l',\nu)}$
Reset	$\dfrac{l \in L}{(l,\nu) \xrightarrow{\kappa(l)}_{lts} (\mathit{free}(l), \nu[\kappa(l)])}$
Delay	$\dfrac{\forall d' \leq d \;\; \nu + d' \models \mathcal{I}(l) \;\; l \in \mathit{free}(L)}{(\mathit{free}(l),\nu) \xrightarrow{d}_{lts} (\mathit{free}(l), \nu + d)}$

Table 1. Transition relation of the lts associated to a timed automaton.

is an additional set of locations for which the set of clock resets is empty, that is, $\kappa(\mathit{free}(l)) = \emptyset$ for all locations l.

- the initial state is $\overline{q} = (\bar{l}, \nu_0)$, where $\nu_0(x) = 0$ for all $x \in \mathcal{C}$.
- $\rightarrow_{lts} \subseteq Q \times ((\Sigma \cup \{\tau\}) \cup \mathbb{R}^+ \cup 2^{\mathcal{C}}) \times Q$ is the set of transitions defined by the rules of Table 1.

The definition we have given is different from the one usually in the literature since we consider clock resets as visible actions. Any subset of clocks $X \subseteq 2^{\mathcal{C}}$ can be reset, and this is visible in the lts. Upon entering a location l, the action of resetting the (possibly empty) set of clocks $\kappa(l)$ is executed, leading to the new clock valuation $\nu[\kappa(l)]$. The reason for this will become clear later on when we give semantics to our language making clock resets visible; this makes no difference in our case because we do not use relations based on the labelled transition systems (e.g., timed bisimulation). It is worth pointing out that, even with this new semantics, the undecidability result for language inclusion still holds, as it is easy to observe that reset actions introduce no branching and therefore they do not add any nondeterminism; it suffices to ignore such actions and we obtain the same traces as with the usual semantics. Note that it is possible to enter a location whose invariant is false, since we do not require the corresponding valuation to satisfy it; in this case no delay transition is possible and the system must perform an action or else deadlock.

The transition system defined above has an infinite (continuous) set of states and actions. In order to model check timed automata, we discretise such state space into equivalence classes that relate clock valuations that agree on the integral part of clocks and on the ordering of their fractional part. Let c_x be the greatest constant against which clock x is compared, $\lfloor x \rfloor$ the integral part of x and $fr(x)$ its fractional part. Given a set of clocks $\mathcal{C}$, two valuations ν and ν' are equivalent ($\nu \equiv_{\mathcal{C}} \nu'$) if all of the following conditions hold:

- for all $x \in \mathcal{C}$, $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or they both exceed c_x;
- for all $x, y \in \mathcal{C}$, with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fr(\nu(x)) \leq fr(\nu(y))$ iff $fr(\nu'(x)) \leq fr(\nu'(y))$;
- for all $x \in \mathcal{C}$ with $\nu(x) \leq c_x$, $fr(\nu(x)) = 0$ iff $fr(\nu'(x)) = 0$.

A *clock region* is an equivalence class induced by $\equiv_{\mathcal{C}}$, and the *region graph* can be thought of as the equivalence classes together with the transitions between the classes. Since all valuations in the same region agree on the integral parts of the clocks, it is clear that the same set of action transitions can be enabled from all states within a region. We denote the set of regions associated to a timed automaton $\mathcal{A}$ by $R_{\mathcal{A}}$, and we let $r, r_1, r_2 \ldots$ range over regions. If $\mathcal{A}$ is clear from the context, it will be elided and we will denote the set of regions as R. We often denote a region by the set of clock constraints that are met by the valuations in the region only. We denote by r_0 the starting region (all clocks set to 0) and by r_{max} the region for which $x > c_x$ for all clocks x. We say that a region r satisfies a condition on clocks $\phi \in \mathcal{B}(\mathcal{C})$ if the condition evaluates to true under all the valuations in r; in this case we write $r \models \phi$.

With passage of time, the automaton changes regions. We define the *successor* region as the next region that the automaton will move to by letting time elapse. Formally, we define a function $\mathit{succ} : R \longrightarrow R$ such that $\mathit{succ}(r) = r'$ if for all $\nu \in r$ there exists $d \in \mathbb{R}$ such that $\nu + d \in \mathit{succ}(r)$ and for all $d' < d$ either $\nu + d' \in r$ or $\nu + d' \in \mathit{succ}(r)$. succ is undefined for r_{max}.

The action of moving to the next region involves an increment of the value of all clocks, but only some of them actually cause the change of a region. For example, if we consider two clocks x and y, when going from the region $x = y = 0$ to the region $0 < x = y < 1$, both clocks change region. However when going from $(0 < x < 1) \wedge (y = 0)$ to $(0 < x < 1) \wedge (0 < y < 1) \wedge (y < x)$, it is only y that changes region. We

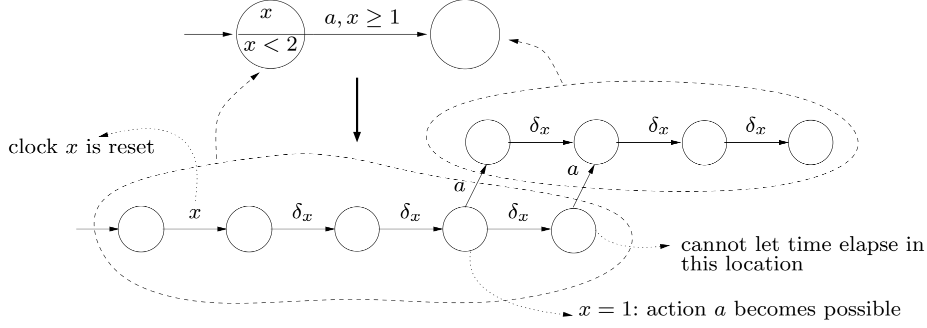

Fig. 1. A timed automaton (top) $\mathcal{A}$ and the corresponding region automaton $R(\mathcal{A})$ (bottom).

are interested in identifying the set of clocks that change their own region as it will be convenient in the following. We define $clocks : R \longrightarrow 2^{\mathcal{C}}$ as the set of clocks that change their own region at the next *succ* action; $clocks(r)$ is the smallest set X of clocks such that, for all valuations $\nu \in r$ and $\eta \in succ(r)$, we have $\nu \equiv_{\mathcal{C}\setminus X} \eta$. We also define Δ as the set of actions describing the change of region due to the passage of time; the elements of Δ are δ_X, where $X \in 2^{\mathcal{C}}$. We can now define the region automaton corresponding to a timed automaton $\mathcal{A}$.

Definition 2.2. Given a timed automaton $\mathcal{A}$, the corresponding **region automaton** $\mathcal{R}_{\mathcal{A}} = (Q_r, \Sigma_r, \overline{q}_r, \rightarrow_r)$ is defined as follows:

- $Q_r = \{(l, r) \mid l \in L \cup free(L),\ r \in R_{\mathcal{A}}\}$
- $\Sigma_r = \Sigma \cup \Delta \cup 2^{\mathcal{C}}$, with $\Delta = \{\delta_X | X \subseteq \mathcal{C}\}$
- $\overline{q}_r = (\bar{l}, r_0)$
- $\rightarrow_r \subseteq Q_r \times (\Sigma_r \cup \{\tau\}) \times Q_r$ such that:
  - $(l, r) \xrightarrow{a}_r (l', r)$, where $a \in \Sigma$ if, for all $\nu \in r$, $(l, \nu) \xrightarrow{a}_{lts} (l', \nu)$;
  - $(l, r) \xrightarrow{X}_r (l, r[X])$, where $X \subseteq \mathcal{C}$, if, for all $\nu \in r$, $(l, \nu) \xrightarrow{X}_{lts} (l, \nu[X])$;
  - $(l, r) \xrightarrow{\delta_X}_r (l, succ(r))$ if, for all $\nu \in r$, there exists d such that $(l, \nu) \xrightarrow{d}_{lts} (l, \nu')$ with $\nu' \in succ(r)$ and $X = clocks(r)$.

It is easy to observe that a transition $(l, r) \xrightarrow{a}_r (l', r)$ is enabled when there is an edge (a, ϕ) leaving location l such that $r \models \phi$. Likewise, a transition $(l, r) \xrightarrow{\delta_X}_r (l, succ(r))$ is enabled if it is possible to let time elapse in the location l, that is, $succ(r) \models \mathcal{I}(l)$. Finally, a clock reset action is enabled upon entering a location.

Region automata are the basis for any algorithm to model check timed automata, and they are an important technique to discretise the infinite state space of the induced transition system. Complexity is their main drawback, as the number of regions is exponential in the number of clocks and in the magnitude of the maximal constants. For this reason, more efficient representations have been devised (e.g. zones, see [Yov98] for an introduction).

Example 2.1. Figure 1 shows a small timed automaton and the corresponding region automaton constructed according to our definitions. The states of the automaton $R(\mathcal{A})$ in the bottom row correspond to the first location of $\mathcal{A}$, while those in the top row correspond to the second location. The first action of $R(\mathcal{A})$ is a reset action, corresponding to the reset of x in the initial location of $\mathcal{A}$. After such action, the value of x is 0, and, with each δ action, the value of x increases in order to reach the next region; so, in the next state we have $0 < x < 1$, $x = 1$ in the following one an so on. Before action a is enabled, at least two delay actions must be executed, reaching the condition $x \geq 1$ of the guarded action of $\mathcal{A}$. The two (maximal) traces of $R(\mathcal{A})$ are $\langle\, \{x\}\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x\, \delta_x \,\rangle$ and $\langle\, \{x\}\, \delta_x\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x \,\rangle$: together, they carry the information that a was possible only while $1 \leq x < 2$.

2.3. CSP

CSP is a process algebra introduced by Tony Hoare [Hoa85]. It describes concurrent systems in terms of their sequential components, characterised by the sequences of actions that they can perform. CSP processes with action alphabet Σ (not including the special silent action τ) are generated by the following syntax:

$$P \ ::= \ \mathit{STOP} \mid \mathit{SKIP} \mid a \rightarrow P \mid P \sqcap P \mid P \,\Box\, P \mid P \underset{A}{\|} P \mid P \setminus A \mid f[P] \mid Z \mid Z = P \mid P\,;\,P \tag{1}$$

where $a \in \Sigma$, $A \subseteq \Sigma$ and $f : \Sigma \rightarrow \Sigma$ is a bijective renaming function. The operators above represent, respectively: deadlock, successful termination, action prefix, internal choice, external choice, interface parallel, hiding, renaming, process name, recursion and sequential composition. We use only one type of parallel operator, as the others, e.g. interleaving, can be expressed in terms of interface parallel.

The semantics of a CSP term P is given by the set of actions that it can perform (*traces*), the set of pairs containing a trace and the actions that can be refused after it (*failures*) or the set of failures and the set of traces causing infinite executions of internal actions (*divergences*). Different relations are built upon these semantic models; for each of them, equivalences between processes are defined as set equalities. CSP also introduces the idea of *refinement*, which is the main focus of this paper: a process P_1 is refined by another process P_2 ($P_1 \sqsubseteq P_2$) if every behaviour of P_2 is a possible behaviour of P_1, that is, if it is “less deterministic”. This idea is formally defined as inverse set inclusion of traces, failures or divergences. For a detailed description, see [Ros98] or [Sch99].

3. Clocked CSP

3.1. Aims

When defining a timed process algebra, we are guided by the desire to combine the features of timed automata and those of CSP. The aim is to *extend CSP* with timing constructs to obtain a calculus that is *inspired* by timed automata and that describes them. We want to define a discrete *denotational semantics* on the language that extends CSP semantics, leading to *decidable refinement relations* that can be checked by using FDR2, and also ensuring that many algebraic properties of CSP can be inherited. We also often directly “import” results of CSP whenever we can find a direct correspondence between our algebra and CSP. In the next sections we describe how we achieved this, justifying the restrictions that we had to impose on the language.

3.2. The Language

We define a language for describing timed automata, called Clocked CSP (CCSP), as an extension of CSP, thus retaining its choice operators, the hiding operator and the multi-way nature of the parallel composition. Clocked CSP terms with alphabet Σ and set of clocks $\mathcal{C}$ are obtained by the following syntax:

$$\begin{array}{lll} P & ::= & \{\!|X|\!\}P \mid \phi \rhd P \mid T \\ T & ::= & \mathit{STOP} \mid \mathit{SKIP} \mid \Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i \mid \bigsqcap_{i=1}^{N} P_i \mid \\ & & P \underset{A}{\|} P \mid Z \mid Z = P \mid P \setminus A \mid f[P] \mid P ; P \end{array} \tag{2}$$

where $a \in \Sigma$, $A \subseteq \Sigma$, $\phi \in \mathcal{B}_c(\mathcal{C})$, $\varphi \in \mathcal{B}(\mathcal{C})$ and $X \subseteq \mathcal{C}$. By convention, visible actions will be ranged over by $a, b, \ldots$, generic actions (including τ) by μ, clocks by $x, y, \ldots$, sets of clocks by $X, Y, \ldots$ and clock constraints by $\phi, \gamma, \cdots$. We denote the set of processes generated by the above grammar by **CCSP**. For simplicity, we write $\{\!|x, y \ldots|\!\}$ instead of $\{\!|\{x, y \ldots\}|\!\}$ when a list of clocks is given explicitly; we also omit the interface alphabet of parallel composition when implicit or not relevant. The syntax above is divided into two parts: the first part (the P terms) introduces the new constructs for handling clocks, that is, clock resets and invariants; in the second part we find the usual constructs of CSP, with some changes regarding internal choice. We use the following abbreviations: $(a, \varphi) \rightarrow P$ (guarded action) for external choice with $N = 1$, and

$|||$ for parallel composition with empty interface alphabet, that is, $\underset{\emptyset}{||}$. Let us consider the new constructs in detail:

- *reset*: $\{|X|\}P$ resets the clocks in X and then behaves as P.
- *invariant*: $\phi \triangleright P$ must evolve (i.e. perform an action) while ϕ is satisfied. This corresponds to invariants in timed automata.
- *guarded actions*: $(a, \varphi) \rightarrow P$ performs a only when the guard φ is satisfied and then behaves as P.
- *external choice*: this operator deserves some discussion. Our version differs from the definition in classical CSP as we allow choice only on visible actions; in this way we prevent components from executing internal actions before the choice is resolved. Even if this is one of the most characteristic traits of CSP's choice operators, we have chosen to avoid it because, in a clocked setting, defining external choice as $P \,\Box\, P$ would create problems with compositional semantics: it would be possible, for example, for a component to execute an internal action that triggers the reset of some clocks. Of course, changing the values of some clocks could change the behaviour of other components. A possible solution would have been to impose disjoint sets of clocks on the components (as we do for the parallel operator, see below), but this would rule out conditions on actions on the same clocks; for instance, it would be impossible to have common conditions, such as the following: do an a action if $x < 1$ and a b action otherwise. It would be possible to impose that any clock reset resolves the choice [CK03]; this would lead to a compositional semantics, but is counter-intuitive and the resulting semantics would be quite involved. We opted for this definition of external choice because it gives a clean, compositional semantics, while still respecting our intuition of what we want to model with external choice, especially in a setting with clocks that function as shared variables.

Internal and external clocks. External choice is not the only operator whose definition is made hard by the presence of clocks: the same difficulties arise for parallel composition if we want to obtain compositionality. Since the interaction between processes that modify the value of clocks creates problems, we make the following simplification: each process can handle only a subset of clocks, so that its behaviour depends only on them. The remaining clocks can be used by other processes that interact with it through the parallel operator.

More formally, given the global set of clocks $\mathcal{C}$, we define the internal set of clocks $\mathcal{C}_i(P)$ for a process P as the set of clocks that are explicitly referred to within P either inside clock reset or invariants. $\mathcal{C}_e(P)$ is defined as the complement $\mathcal{C} \setminus \mathcal{C}_i(P)$. We restrict the parallel operator to work only with pairs of processes P_1 and P_2 with disjoint sets of internal clocks, that is, such that $\mathcal{C}_i(P_1) \cap \mathcal{C}_i(P_2) = \emptyset$. The internal clocks of the parallel composition are defined as $\mathcal{C}_i(P_1 \,||\, P_2) = \mathcal{C}_i(P_1) \cup \mathcal{C}_i(P_2)$. When the process P is implicit from the context, we denote the set of its internal and external clocks simply by $\mathcal{C}_i$ and $\mathcal{C}_e$, respectively, instead of $\mathcal{C}_i(P)$ and $\mathcal{C}_e(P)$.

The idea behind this restriction is that, when defining the semantics of a process, we have to assume that, while the process has full control of its internal clocks, any action on external clocks is possible at any time, caused by any process running in parallel. A process must be willing to synchronise on any possible set of (external) clock resets at any moment. This is why we treat clock resets as visible actions: when a process resets a set of clocks X, the parallel processes are willing to synchronise on this reset action. In this way processes always agree on the value of clocks. We believe this is a reasonable restriction since most parallel composition constructs for timed automata use disjoint sets of clocks.

4. Operational Semantics

For the purpose of giving semantics to Clocked CSP, we introduce an extra operator, $\mathsf{free}(P)$, representing a process that behaves exactly like P but which does not perform any initial reset (i.e. the start state has been stripped of its resets). This performs a similar function to the operator *free* introduced to give semantics to timed automata. We denote the set of all CCSP processes extended with the $\mathsf{free}(\bullet)$ operator by $\mathbf{CCSP}^+$. Given a CCSP term P, we define the corresponding timed automaton $\mathcal{A}(P) = (L, \bar{l}, \Sigma \cup \{\tau\} \cup \{\checkmark\}, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, where $L = \mathbf{CCSP}^+$, $\bar{l} = P$, the sets of clocks and actions are the same and $\rightarrow$, κ and $\mathcal{I}$ are defined according to the rules of Table 2, Table 3 and Table 4, respectively. We have already introduced the silent action τ, which is the result of some internal computation, usually caused by internal choice or hiding. Following the

$$\frac{}{\mathit{SKIP} \xrightarrow{\checkmark,\mathtt{tt}} \Omega} \qquad \frac{}{Z = P \xrightarrow{\tau,\mathtt{tt}} P[Z = P\,/\,Z]}$$

$$\frac{P \xrightarrow{a,\varphi} P'}{\phi \rhd P \xrightarrow{a,\varphi} P'} \qquad \frac{P \xrightarrow{a,\varphi} P'}{\{\!|X|\!\}P \xrightarrow{a,\varphi} P'} \qquad \frac{P \xrightarrow{a,\varphi} P'}{\mathsf{free}(P) \xrightarrow{a,\varphi} P'}$$

$$\frac{}{(\,\Box_{i=1}^{N}(a_i,\varphi_i) \to P_i) \xrightarrow{a_j,\varphi_j} P_j} \quad j \in \{1..N\}$$

$$\frac{}{\sqcap_{i=1}^{N} P_i \xrightarrow{\tau,\mathtt{tt}} P_j} \quad j \in \{1..N\} \qquad \frac{P \xrightarrow{a,\varphi} P'}{f[P] \xrightarrow{f(a),\varphi} f[P']}$$

$$\frac{P \xrightarrow{a,\varphi} P'}{P \setminus A \xrightarrow{\tau,\varphi} P' \setminus A} \; a \in A \qquad \frac{P \xrightarrow{\mu,\varphi} P'}{P \setminus A \xrightarrow{\mu,\varphi} P' \setminus A} \; \mu \notin A$$

$$\frac{P \xrightarrow{\mu,\varphi} P'}{P \underset{A}{||} Q \xrightarrow{\mu,\varphi} P' \underset{A}{||} \mathsf{free}(Q)} \; \mu \notin A \qquad \frac{Q \xrightarrow{\mu,\varphi} Q'}{P \underset{A}{||} Q \xrightarrow{\mu,\varphi} \mathsf{free}(P) \underset{A}{||} Q'} \; \mu \notin A$$

$$\frac{P \xrightarrow{a,\varphi_1} P' \quad Q \xrightarrow{a,\varphi_2} Q'}{P \underset{A}{||} Q \xrightarrow{a,\varphi_1 \wedge \varphi_2} P' \underset{A}{||} Q'} \; a \in A \cup \{\checkmark\}$$

$$\frac{P \xrightarrow{\mu,\varphi} P'}{P;Q \xrightarrow{\mu,\varphi} P';Q} \; \mu \neq \checkmark \qquad \frac{P \xrightarrow{\checkmark} \Omega}{P;Q \xrightarrow{\tau,\mathtt{tt}} Q}$$

Table 2. Rules for transitions

$$\mathcal{I}(\mathit{STOP}) = \mathtt{tt} \qquad \mathcal{I}(\mathit{SKIP}) = \mathtt{tt} \qquad \mathcal{I}(\phi \rhd P) = \phi \wedge \mathcal{I}(P)$$

$$\mathcal{I}(\{\!|X|\!\}P) = \mathcal{I}(P) \qquad \mathcal{I}(\mathsf{free}(P)) = \mathcal{I}(P) \qquad \mathcal{I}(Z = P) = \mathtt{ff},$$

$$\mathcal{I}\left(\Box_{i=1}^{N}(a_i,\varphi_i) \to P_i\right) = \mathtt{tt} \qquad \mathcal{I}\left(\bigsqcap_{i=1}^{N} P_i\right) = \mathtt{ff} \qquad \mathcal{I}\left(P \underset{A}{||} Q\right) = \mathcal{I}(P) \wedge \mathcal{I}(Q)$$

$$\mathcal{I}(P \setminus A) = \mathcal{I}(P) \qquad \mathcal{I}(f[P]) = \mathcal{I}(P) \qquad \mathcal{I}(P;Q) = \mathcal{I}(P)$$

Table 3. Rules for invariants

CSP convention, we use an additional special label $\checkmark$, with a true guard, to denote successful termination and an additional process Ω that denotes the process that has successfully terminated.

Since Clocked CSP is designed to model timed automata, the operational semantics is intuitive and it extends CSP semantics in the obvious way for most operators. One design choice that we have made was to have the invariant of internal choice to be false; the reason for this is that internal choice is resolved immediately. In this way the semantics of internal choice follows that of internal choice in standard CSP exactly. Similarly, the invariant of $Z = P$ is false, since we force recursive calls to unfold immediately. We explain the introduction of the $\mathsf{free}(\bullet)$ operator by means of the following example [D'A99].

Example 4.1. Consider the process $\{\!|x|\!\}(a,\varphi_1) \to \mathit{STOP} \,|||\, \{\!|y|\!\}(b,\varphi_2) \to \mathit{STOP}$. At the beginning of the execution, both components reset their clocks; when one of the components executes, we do not want the other component to reset its clock again. The operational semantics of this term is given by the automaton of Figure 2; without the free operator, the clocks would be reset in the middle locations as well, even if they have both been reset at the start.

$$\kappa(STOP) = \emptyset \qquad \kappa(SKIP) = \emptyset \qquad \kappa(\phi \triangleright P) = \kappa(P)$$

$$\kappa(\{\!|X|\!\}P) = \{X\} \cup \kappa(P) \qquad \kappa(\mathsf{free}(P)) = \emptyset \qquad \kappa(Z = P) = \emptyset$$

$$\kappa\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i\right) = \emptyset \qquad \kappa\left(\sqcap_{i=1}^{N} P_i\right) = \emptyset \qquad \kappa\left(P \underset{A}{||} Q\right) = \kappa(P) \cup \kappa(Q)$$

$$\kappa(P \setminus A) = \kappa(P) \qquad \kappa(f[P]) = \kappa(P) \qquad \kappa(P;Q) = \kappa(P)$$

Table 4. Rules for clock resets

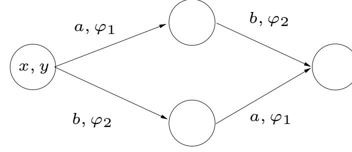

Fig. 2. Explanation of the $\mathsf{free}(\bullet)$ operator.

The timed automaton corresponding to a CCSP term might have an infinite number of locations, thus also leading to an infinite-state region automaton. For automatic verification purposes this should be avoided, and it is necessary to restrict to finite state processes, that is, processes whose operational model has a finite number of reachable states. The same restrictions as those applied to standard CSP apply in our case; see [Ros94] for a discussion of finite state processes.

Normal form. We need to impose several restrictions on the syntax of terms, so that we get terms whose structure makes the semantics more intuitive and which also allow for a simpler definition of the semantic rules. Consider the following term, $(x \leq 2) \triangleright \{\!|x|\!\}(a, x > 1) \to P$, and its corresponding timed automaton (Figure 3). From the timed automaton, we can see that the invariant $x \leq 2$ is bound by the reset of clock x, but this contradicts the intuition, since we use x before it is reset (this corresponds to the idea of *conflict* in [D'A99]), and we would like to rewrite the term as $\{\!|x|\!\}(x \leq 2) \triangleright (a, x > 1) \to P$.

Also, consider the term, $\{\!|x, y|\!\}((x \leq 2) \triangleright P_1 \,||\, (y < 1) \triangleright P_2)$, where x is a clock of the first component and y is a clock of the second. Both clock x and clock y are immediately reset (by applying the rules of Table 4). In order to simplify the semantics, and also to emphasise that each component handles its own clocks, we would like to rewrite this term as: $\{\!|x|\!\}((x \leq 2) \triangleright P_1 \,||\, \{\!|y|\!\}(y < 1) \triangleright P_2)$.

The two examples above explain our need for a *normal form* for CCSP terms by which the structure of CCSP terms reflects their semantics, and we prefix terms that can execute actions by resets and invariants all grouped together. We obtain the following alternative and equivalent (up to operational semantics) syntax:

$$\begin{array}{lcl} P & ::= & \{\!|X|\!\}\phi \triangleright T \mid P \underset{A}{||} P \mid P \setminus A \mid f[P] \mid P;P \mid Z = P \mid Z \\ T & ::= & STOP \mid SKIP \mid \Box_{i=1}^{N}(a_i, \varphi_i) \to P_i \mid \sqcap_{i=1}^{N} P_i \mid \end{array} \tag{3}$$

A CCSP term is either a T term prefixed by a (possibly empty) set of clock resets and a (possibly true) invariant, the parallel compositions of such terms without any leading clock operator, or some other term with some actions hidden or renamed.

Proposition 4.1. Every CCSP term can be rewritten to an equivalent (up to operational semantics) CCSP term generated by the syntax of Equation 3.

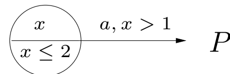

Fig. 3. Timed automaton induced by $(x \leq 2) \triangleright \{\!|x|\!\}(a, x > 1) \to P$. Clock x is bound by the initial reset.

$P = \{\mid\emptyset\mid\}P$	$P = \texttt{tt} \triangleright P$
$\phi_1 \triangleright \phi_2 \triangleright P = (\phi_1 \wedge \phi_2) \triangleright P$	$\{\mid X\mid\}\{\mid Y\mid\}P = \{\mid X \cup Y\mid\}P$
$\phi \triangleright \{\mid X\mid\}P = \{\mid X\mid\}\phi \triangleright P$	$\phi \triangleright (P_1 \mid\mid P_2) = (\phi \triangleright P_1) \mid\mid (\phi \triangleright P_2)$

$$\{\mid X_1 \cup X_2\mid\}(P_1 \mid\mid P_2) = (\{\mid X_1\mid\}P_1) \mid\mid (\{\mid X_2\mid\}P_2) \text{ with } X_1 \subseteq \mathcal{C}_i(P_1) \text{ and } X_2 \subseteq \mathcal{C}_i(P_2)$$

Table 5. Rewrite rules to obtain CCSP terms that are equivalent up to the operational semantics, i.e., they induce the same timed automaton.

Proof. Let us use structural induction on the terms generated by the syntax of Equation 2. Clearly, *STOP*, *SKIP* and all the T terms are already compatible with the alternative syntax if their components are. We need to prove that the result is true for all the P terms. This can be done by considering the sequence of resets and invariants and by applying the rules of Table 5 to rewrite the leftmost elements, until we are left with a unique reset operation followed by a unique invariant operation. By using induction, the result is proved if the P term is followed by any T term except for parallel composition. If the term is followed by parallel composition, the syntax of Equation 3 is not respected because the parallel operator cannot be preceded by resets and invariants. This case can be solved by using the last two rules of Table 5 "pushing" the resets and invariants inside the parallel. $\square$

Region automaton. In the following, we will use the region automaton obtained from the operational semantics as the model we have in mind for the denotational models for CCSP. Given a CCSP term P and the associated timed automaton $\mathcal{A}(P)$, we denote the region automaton corresponding to $\mathcal{A}(P)$ with initial region r by $R(P, r)$. If we want to add an extra invariant φ to the initial location, i.e. $\mathcal{I}(P) = \mathcal{I}(P) \wedge \varphi$, we denote the resulting region automaton by $R(P, r, \varphi)$. We use $R(P)$ as an abbreviation for $R(P, r_0, \texttt{tt})$.

Representation theorem. We state that CCSP is a complete language for the description of timed automata, that is, every timed automaton can be described by a CCSP term.

Theorem 4.1. For every (finitely branching) timed automaton $\mathcal{A}$, there exists a CCSP term P such that $\mathcal{A}(P)$ is isomorphic to the reachable part of $\mathcal{A}$, excluding the τ actions introduced by the operational rule for process definition (Table 2).

Proof. The proof is similar to the proof in [D'A99], but we also have to deal with internal τ actions. Given a timed automaton $\mathcal{A} = (L, \bar{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, we create a CCSP term with the same alphabet and set of clocks in the following way: for each location l, we define a process name P_l. A process is associated with each name, that is, for each $l \in L$, $P_l = \{\mid\kappa(l)\mid\}\mathcal{I}(l) \triangleright \left(\Box_{(l,a_i,\phi_i,l_i)\in\rightarrow}(a_i, \phi_i) \rightarrow P_{l_i}\right)$. This does not work if $\mathcal{A}$ contains internal actions. This problem can be solved by using an extra action b in the construction above whenever a τ is encountered. This action can then be hidden at the top level. Clearly, the function $f : L \rightarrow \mathbf{CCSP}$ defined as $f(l) = P_l$ is an isomorphism for the reachable part of the timed automaton, if we exclude the τ actions introduced by process definitions $Z = P$; such actions do not introduce non-determinism, so we can still consider the two timed automata as equivalent. $\square$

Example 4.2. Consider the timed automaton $\mathcal{A}$ of Example 2.1: the CCSP term corresponding to $\mathcal{A}$ is given by $\{\mid x\mid\}(x < 2) \triangleright (a, x \geq 1) \rightarrow \mathit{STOP}$.

5. Denotational Semantics

By working on the transition system induced by the operational model, one could use known equivalence relations for timed automata (e.g. timed bisimulation) or use traditional model checking techniques to verify whether a given process meets some temporal logic property. Since we are following the CSP approach, we want to give a denotational semantics to the language as an extension of the usual trace/failures semantics, together with *refinement* relations. As stated at the beginning of the previous section, we want the refinement

relations generated by the semantic model to be decidable, so that we can use or suitably extend FDR2 to automatically verify processes against specifications. This rules out the most natural model, that of timed traces (sequences of actions together with the time at which they were performed), as used in Timed CSP: it is known that timed trace inclusion and equivalence are generally undecidable problems for timed automata [AD94]. For this reason, we have to identify a suitable level of abstraction between the infinite and undecidable semantic level of the transition system and some appropriate high-level description. Our choice was to model the semantics on the region automaton. The idea is to record the clock constraints met when an action is taken, rather than the absolute time.

5.1. Trace Semantics

We give Clocked CSP processes a semantics in terms of region traces. A region trace is an element of $(\Sigma \cup \Delta \cup 2^{\mathcal{C}})^*$, where Σ is the process alphabet, Δ is the set of delay actions and $\mathcal{C}$ is the set of clocks. We denote the extended alphabet $\Sigma \cup \Delta \cup 2^{\mathcal{C}}$ by $\overline{\Sigma}$, $\Sigma \cup \{\checkmark\}$ by $\Sigma^{\checkmark}$ and $\overline{\Sigma} \cup \{\checkmark\}$ by $\overline{\Sigma}^{\checkmark}$. $\mathsf{RTraces}$ is the semantic model we study and it denotes the set of non-empty, prefix-closed subsets of $\overline{\Sigma}^{\checkmark}$. An element of a trace either denotes an action ($a \in \Sigma$), the passage of time (delay action $\delta_X \in \Delta$) or the reset of some set of clocks ($X \in 2^{\mathcal{C}}$). By following the sequences of delays and resets it is always possible to know which clock constraints are met when an action is taken, and trace synchronisation is also possible.

We need to be able to define the semantics of a process from any possible starting region and under any possible initial invariant. Consider, for example, the process $(a, x \leq 1) \rightarrow P$: its semantics depends on the semantics of the process P starting from 3 possible regions ($x = 0$, $0 < x < 1$ and $x = 1$), depending on when a is taken. For this reason, the semantics of a process is the set of possible behaviours under any possible starting condition. The refinement relation is extended accordingly as inverse set inclusion under every starting condition. Formally, let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of regions corresponding to a term and $\mathcal{B}_c(\mathcal{C}) = \{\phi_1, \phi_2, \ldots, \phi_m\}$ the set of possible invariants (note that this set is finite as an invariant is the union of a set of regions). We define a function $\mathcal{RT} : \mathbf{CCSP} \times R \times \mathcal{B}_c(\mathcal{C}) \rightarrow \mathsf{RTraces}$ that returns the region traces of a process, assuming it starts from a particular region under a particular invariant. The semantics of a process is thus given by an ordered set of sets of traces:

$$\begin{aligned} \mathit{RegionTraces}(P) = (&\mathcal{RT}(P, r_1, \phi_1), \mathcal{RT}(P, r_1, \phi_2), \ldots, \mathcal{RT}(P, r_1, \phi_m), \\ &\ldots \\ &\mathcal{RT}(P, r_n, \phi_1), \mathcal{RT}(P, r_n, \phi_2), \ldots, \mathcal{RT}(P, r_n, \phi_m)) \end{aligned}$$

The function $\mathcal{RT}$ is defined inductively on the syntax of terms along the same lines as the derivation rules for traces for classical CSP. The definition of the function $\mathcal{RT}$ is given in the following subsection. The semantic domain for processes in the region trace model is given by $\mathsf{RTraces}^{mn}$, where m the the number of possible invariants and n is the number of possible regions. It is clear that this forms a *complete lattice* under inverse inclusion of components. We can then extend the refinement relation as inverse inclusion of behaviour:

$$\begin{aligned} P \sqsubseteq_{\mathcal{RT}} Q \quad &\text{iff} \quad \mathit{RegionTraces}(P) \supseteq \mathit{RegionTraces}(Q) \\ &\text{iff} \quad \forall r_i\ \forall \phi_j\ \mathcal{RT}(P, r_i, \phi_j) \supseteq \mathcal{RT}(Q, r_i, \phi_j) \end{aligned}$$

meaning that for every possible starting region, Q's behaviour is a subset of P's behaviour. It can be shown that the refinement relation is independent of the starting conditions for processes that reset all their clocks before referencing them; hence, only one set inclusion needs to be verified. Moreover, we usually consider refinement between processes that have no external clocks, in which case their behaviour is self-contained.

5.1.1. Rules for Region Traces

We give the definition of the function $\mathcal{RT}$ that identifies the set of region traces of a process. First, a few auxiliary definitions are needed. The following operator is necessary to concatenate two traces in the case of sequential composition.

Definition 5.1. Given a region trace t and starting region r, the region after the execution of t is defined as follows:

- $\mathsf{after}(\langle\rangle, r) = r$
- $\mathsf{after}(a\hat{\ }t', r) = \mathsf{after}(t', r)$ if $a \in \Sigma$
- $\mathsf{after}(\delta_X\hat{\ }t', r) = \mathsf{after}(t', succ(r))$ if $\delta_X \in \Delta$
- $\mathsf{after}(X\hat{\ }t', r) = \mathsf{after}(t', r[X])$ if $X \subseteq \mathcal{C}$

We need to be able to synchronise two traces in the case of the parallel operator. The synch_A operator is the same as for standard CSP (e.g. [Sch99]); the only difference is that traces are implicitly synchronised not only on the interface alphabet A, but also on delay actions Δ and on clock resets. This guarantees that action are synchronised only if they are performed under the same clock constraints, which is the very idea behind the region semantics of Clocked CSP. In the following definition *head* and *tail* are the standard operators on traces, returning the first element of a trace and the remainder of a trace, respectively.

Definition 5.2. synch_A is a relation describing when a trace tr can be a synchronisation of two traces tr_1 and tr_2 with interface alphabet A. It is defined as follows:

- $\langle\rangle\ \mathsf{synch}_A\ (tr_1, tr_2)$ iff $tr_1 = tr_2 = \langle\rangle$
- $\langle\checkmark\rangle\ \mathsf{synch}_A\ (tr_1, tr_2)$ iff $tr_1 = tr_2 = \langle\checkmark\rangle$
- $\langle a\rangle\hat{\ }tr\ \mathsf{synch}_A\ (tr_1, tr_2)$ with $a \neq \checkmark$ iff
  - $a \in (A \cup \Delta \cup \mathcal{C})\ \wedge\ head(tr_1) = head(tr_2) = a \wedge tr\ \mathsf{synch}_A\ (tail(tr_1), tail(tr_2))$ or
  - $a \notin (A \cup \Delta \cup \mathcal{C}) \wedge (head(tr_1) = a\ \wedge tr\ \mathsf{synch}_A\ (tail(tr_1), tr_2)$
    $\vee\ head(tr_2) = a\ \wedge tr\ \mathsf{synch}_A\ (tr_1, tail(tr_2)))$

We let $t \setminus A$, where t is a sequence of actions and A is a subset of actions, denote the sub-sequence of t that excludes all elements that are in A.

The function $\mathcal{RT}$ is now defined recursively on CCSP terms by the following rules (note that the rules for external clocks assume the syntactic restrictions just described). Recall that $clocks(r)$ denotes the set of clocks whose value is going to change from region r and that r_{max} denotes the region where the value of all clocks exceeds the maximal constant against which clocks are compared. In the following rules we often need to check whether time can elapse by means of a change of region; this is possible if both the current region r and its successor $succ(r)$ satisfy the current invariant, and if r is not the maximal region r_{max}.

- $\mathcal{RT}(\{\!|X_i|\!\}P, r, I) = \{\langle\rangle\} \cup$
  $\{(X_i \cup X_e)\hat{\ }t \mid t \in \mathcal{RT}(P, r[X_i \cup X_e], I) \text{ and } X_e \subseteq C_e\}$
- $\mathcal{RT}(\phi \triangleright P, r, I) = \mathcal{RT}(P, r, I \wedge \phi)$
- $\mathcal{RT}(STOP, r, I) = \{\langle\rangle\} \cup$
  $\{X_e\hat{\ }t \mid t \in \mathcal{RT}(STOP, r[X_e], I)\}$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\cup\ \{\delta_{clocks(r)}\hat{\ }t \mid t \in \mathcal{RT}(STOP, succ(r), I)\}$
- $\mathcal{RT}(SKIP, r, I) = \{\langle\rangle, \langle\checkmark\rangle\} \cup$
  $\{X_e\hat{\ }t \mid t \in \mathcal{RT}(SKIP, r[X_e], I)\}$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\cup\ \{\delta_{clocks(r)}\hat{\ }t \mid t \in \mathcal{RT}(SKIP, succ(r), I)\}$
- $\mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i, r, I\right) = \{\langle\rangle\} \cup$
  $\left\{X_e\hat{\ }t \mid t \in \mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i, r[X_e], I\right)\right\} \cup$
  $\{a_i\hat{\ }t \mid t \in \mathcal{RT}\ (P_i, r, \mathtt{tt})\ \wedge\ r \models \varphi_i\}$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\cup\ \left\{\delta_{clocks(r)}\hat{\ }t \mid t \in \mathcal{RT}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \to P_i, succ(r), I\right)\right\}$
- $\mathcal{RT}(\sqcap_{i=1}^{N} P_i, r, I) = \bigcup_{i=1}^{N} \mathcal{RT}(P_i, r, \mathtt{tt})$
- $\mathcal{RT}(P_1 \underset{A}{||} P_2, r, I) = \{t \mid \exists t_1, t_2\ .\ t_1 \in \mathcal{RT}(P_1, r, I) \wedge$
  $t_2 \in \mathcal{RT}(P_2, r, I) \wedge t\ \mathsf{synch}_A\ (t_1, t_2)\}$

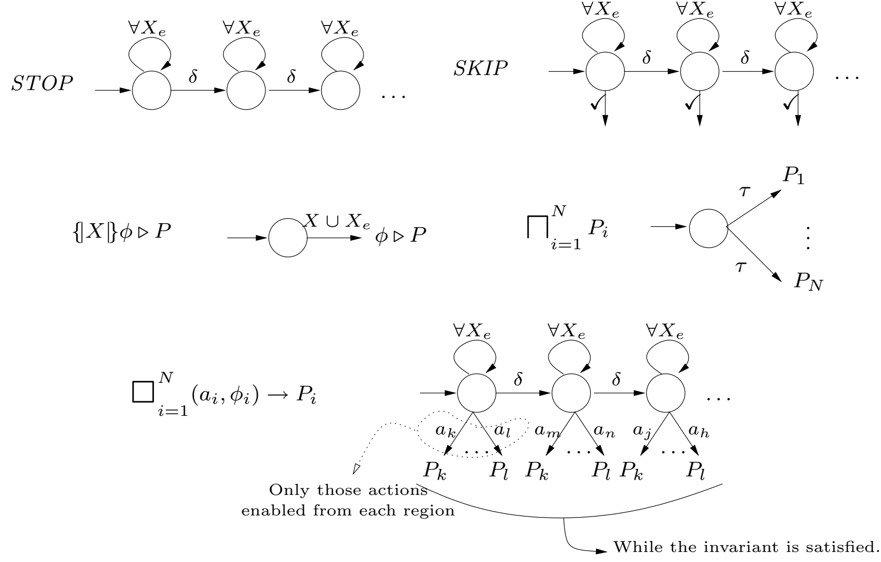

Fig. 4. The region automata corresponding to some operators.

- $\mathcal{RT}(P \setminus A, r, I) = \{t \setminus A \mid t \in \mathcal{RT}(P, r, I)\}$
- $\mathcal{RT}(f[P], r, I) = \{t \mid f^{-1}(t) \in \mathcal{RT}(P, r, I)\}$
- $\mathcal{RT}(P_1; P_2, r, I) = \{t \mid \exists t_1, t_2 \,.\, t = t_1 \hat{} t_2 \wedge t_1 \hat{} \checkmark \in RT(P_1, r, I)$
  $\wedge\, t_2 \in \mathcal{RT}(P_2, \mathsf{after}(t_1, r), \mathtt{tt})\} \cup \{t \mid t \in \mathcal{RT}(P_1, r, I)\}$

Figure 4 helps understand these rules by giving the intuition of the operational model underlying a term. Some CCSP operators behave exactly as in CSP. For example, internal choice is resolved immediately as its invariant is false and a process is chosen via a τ transition. The behaviour of clock reset is quite simple: the process $\{\!|X|\!\}\phi \triangleright P$ first resets the set of clocks X (this is a visible action and it appears in the trace, see Table 1), then it behaves like P under the invariant ϕ. This process must also be able to synchronise with other processes that want to reset their clocks; by performing the action $X \cup X_e$, for all $X_e \subseteq \mathcal{C}_e$, the process takes into account the possibility of external clocks being reset. The behaviour of external choice is quite involved if compared with the corresponding rule for standard CSP. The reason for this is that, in Clocked CSP, time can elapse (δ actions) before the choice is resolved. This explains why there are several states in the operational model to represent choice. The actions whose guards are satisfied are enabled from each region. Finally, from each state, the process must be willing to synchronise with external clocks of other processes. Reset actions of external clocks would lead to different states and would not in general be self loops, but we represent them this way for conciseness since they do not alter the values of internal clocks and the same actions are enabled after external clock resets. Since no delay action is possible from r_{max}, we cannot have an infinite number of delays without actions or clock resets taking place. The case for *STOP* and *SKIP* is similar; for example, these processes can let time elapse while their invariant are true and, in the case of *SKIP*, must successfully terminate by executing a $\checkmark$ action.

Recursion. We treat recursion in the same way as standard CSP and the semantics of $Z = P$ is given by the least fixed point of the term P. Let $P^0(STOP) = STOP$ and $P^{n+1}(STOP) = P[P^n(STOP)/Z]$, that is, the $n+1$-th unwinding of P starting from the least process $STOP$; then the least fixed point is given by $RegionTraces(Z = P) = \bigcup_{n=0}^{\infty} RegionTraces(P^n(STOP))$ and $\mathcal{RT}(Z = P, r, I) = \bigcup_{n=0}^{\infty}(P^n(STOP), r, I)$. Since the trace domain is a complete lattice, we need to show that all operators are monotonic with respect to reverse set inclusion in order to prove that recursion is well-defined for guarded processes and there exists a least fixed point [Ros98].

Theorem 5.1. *RegionTraces* is a monotonic function with respect to all operators, and the least fixed point operator exists for guarded processes.

Proof. The result is straightforward for most operators following the standard arguments for CSP. Guardedness ensures the existence of a unique fixed point. Let us then prove monotonicity for the added operators:

- (Reset) $P_1 \sqsubseteq_{\mathcal{RT}} P_2 \implies \{\!|X|\!\}P_1 \sqsubseteq_{\mathcal{RT}} \{\!|X|\!\}P_2$: we have $\{\!|X|\!\}P_1 \sqsubseteq_{\mathcal{RT}} \{\!|X|\!\}P_2$ iff $\forall r, I\ \mathcal{RT}(\{\!|X|\!\}P_1, r, I) \supseteq \mathcal{RT}(\{\!|X|\!\}P_1, r, I)$. By applying the rule for reset, we get that this inclusion is true if and only if $\mathcal{RT}(P_1, r[X \cup X_e], I) \supseteq \mathcal{RT}(P_2, r[X \cup X_e], I)$, which is true by induction.
- (Invariant) $P_1 \sqsubseteq_{\mathcal{RT}} P_2 \implies \phi \triangleright P_1 \sqsubseteq_{\mathcal{RT}} \phi \triangleright P_2$. A similar argument to that above applies for this operator.
- (External choice) We can consider external choice with two terms as the argument easily extends to the finitary case, in which all the terms are equal except for one. Let us consider $P_1 = (a_1, \phi_1) \to Q_1 \,\Box\, (a_2, \phi_2) \to Q$ and $P_2 = (a_1, \phi_1) \to Q_2 \,\Box\, (a_2, \phi_2) \to Q$ such that $Q_1 \sqsubseteq_{\mathcal{RT}} Q_2$, then we have to prove that $P_1 \sqsubseteq_{\mathcal{RT}} P_2$.
  Note that the rule for external choice is defined in terms of itself: this is because, before taking an action, a process can allow to let time elapse for a finite number of times. Moreover, delay actions can be interleaved by a sequence of external clock resets. The idea is that both processes can execute the same sequence of delay and resets, and then an action is taken and the inductive step can be applied. Let us assume that such sequences are finite; if they are not, the two processes "diverge" at this point and have the same set of traces. So, before enabling an action, both processes can execute the same sequences of delays (they have the same invariant and the actions are guarded by the same condition) and external resets (neither process has control on this part) and they both end up in the same region. Assume some sequence of delays and resets $\delta X_1 \cdots \delta X_j$; the region after this sequence is given by $succ(succ(\cdots succ(r)[X_1] \cdots)[X_{j-1}])[X_j]$. If at this point a_1 is taken, then the inductive step can be applied, else if a_2 is taken, we trivially get the same trace.
  The proofs for *SKIP* and *STOP* proceed in the same way.

□

Example 5.1. Consider the process $P = \{\!|x|\!\}(x < 2) \triangleright (a, x \geq 1) \to STOP$ from Example 4.2. Assume $\mathcal{C}_e(P) = \emptyset$, then we can use the rules given above to find the region traces of this process from the initial region $r_0 = (x = 0)$:

$$\mathcal{RT}(P, x=0, \mathtt{tt}) = \{\langle\rangle\} \cup \{\{x\}\hat{\ }t \mid t \in \mathcal{RT}((x<2) \triangleright (a, x \geq 1) \to STOP, x=0, \mathtt{tt})\}$$
$$\mathcal{RT}((x<2) \triangleright (a, x \geq 1) \to STOP, x=0, \mathtt{tt}) = \mathcal{RT}((a, x \geq 1) \to STOP, x=0, x<2)$$
$$\mathcal{RT}((a, x \geq 1) \to STOP, x=0, x<2) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}((a, x \geq 1) \to STOP, 0<x<1, x<2)\}$$
$$\mathcal{RT}((a, x \geq 1) \to STOP, 0<x<1, x<2) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}((a, x \geq 1) \to STOP, x=1, x<2)\}$$
$$\mathcal{RT}((a, x \geq 1) \to STOP, x=1, x<2) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}((a, x \geq 1) \to STOP, 1<x<2, x<2)\} \cup \{a\hat{\ }t \mid t \in \mathcal{RT}(STOP, x=1, \mathtt{tt})\}$$
$$\mathcal{RT}((a, x \geq 1) \to STOP, 1<x<2, x<2) = \{\langle\rangle\} \cup \{a\hat{\ }t \mid t \in \mathcal{RT}(STOP, 1<x<2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x=1, \mathtt{tt}) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}(STOP, 1<x<2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, 1<x<2, \mathtt{tt}) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}(STOP, x=2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x=2, \mathtt{tt}) = \{\langle\rangle\} \cup \{\delta_x\hat{\ }t \mid t \in \mathcal{RT}(STOP, x>2, \mathtt{tt})\}$$
$$\mathcal{RT}(STOP, x>2, \mathtt{tt}) = \{\langle\rangle\}$$

since $r_{\max} = x > 2$. From the equations above we get that the two maximal traces $\langle \{x\}\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x\, \delta_x \rangle$ and $\langle \{x\}\, \delta_x\, \delta_x\, \delta_x\, a\, \delta_x\, \delta_x \rangle$ of Example 2.1 are in $\mathcal{RT}(P, r_0, \mathtt{tt})$, as expected. $\mathcal{RT}(P, r_0, \mathtt{tt})$ contains these two traces and all of their possible prefixes.

5.2. Region Failures

Having defined the semantic model for region traces, the next natural step is to extend it to a finer semantics that distinguishes between stable failures. We define *region failures*, again having in mind the operational model of region automata. A region refusal set F is a subset of $\overline{\Sigma}^{\checkmark}$ that describes the set of actions that a process can refuse after a specified trace. The most interesting case happens when a process refuses a delay action, as this is useful to describe timed liveness properties: refusing a delay action means refusing to let time

elapse, so one could specify that an action a must happen before some bound by refusing to let time elapse until a has been performed. A region failure is a pair (t, F), where t is a region trace and F is a refusal set. We denote the semantic domain of region failures by $\mathsf{RFailures}$. Following a line of reasoning similar to that for region traces, we obtain a new semantic model for Clocked CSP processes, given by the *RegionFailures* function which is once again congruent with the operational model, and a refinement relation $\sqsubseteq_{\mathcal{RF}}$.

5.2.1. Rules for Region Failures

RegionFailures is the function that denotes processes under the region failure model and is described as follows, where again each term of the tuple ranges over the possible regions and invariants:

$$\begin{aligned} \mathit{RegionFailures}(P) = (&\mathcal{RF}(P, r_1, \phi_1), \mathcal{RF}(P, r_1, \phi_2), \ldots, \mathcal{RF}(P, r_1, \phi_m), \\ &\ldots \\ &\mathcal{RF}(P, r_n, \phi_1), \mathcal{RF}(P, r_n, \phi_2), \ldots, \mathcal{RF}(P, r_n, \phi_m)) \end{aligned}$$

As for standard CSP, we introduce the maximal element for stable failures, DIV, that executes no action and does not refuse anything, whose semantics is given by $\mathcal{RF}(DIV, r, I) = \{(\langle\rangle, \{\})\}$. The semantic domain for region failures forms a complete lattice under reverse inclusion, like the stable failure domain of standard CSP. We define the function $\mathcal{RF}$ inductively on the structure of terms.

- $\mathcal{RF}(\{\!|X|\!\}P, r, I) =$
  $\left\{(\langle\rangle, F) \mid F \subseteq \Sigma^{\checkmark} \cup \Delta \cup 2^{C \setminus X}\right\} \cup$
  $\{(X \cup X_e)\hat{\ }t, F) \mid X_e \subseteq \mathcal{C}_e \wedge (t, F) \in \mathcal{RF}(P, r[X \cup X_e], I)\}$
- $\mathcal{RF}(\phi \triangleright P, r, I) = \mathcal{RF}(P, r, I \wedge \phi)$
- $\mathcal{RF}(STOP, r, I) =$
  $\{(X_e\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\,(STOP, r[X_e], I)\} \cup$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark} \setminus \left(2^{\mathcal{C}_e} \cup \{\delta_{clocks(r)}\}\right)\right\} \cup$
  $\cup \left\{(\delta_{clocks(r)}\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\,(STOP, succ(r), I)\right\}$
  else (time cannot elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark} \setminus 2^{\mathcal{C}_e}\right\}$
- $\mathcal{RF}(SKIP, r, I) = \{(\langle\checkmark\rangle, F) \mid F \subseteq \overline{\Sigma}^{\checkmark}\} \cup$
  $\{(X_e\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\,(SKIP, r[X_e], I)\} \cup$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma} \setminus \left(2^{\mathcal{C}_e} \cup \{\delta_{clocks(r)}\}\right)\right\} \cup$
  $\cup \left\{(\delta_{clocks(r)}\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\,(SKIP, succ(r), I)\right\}$
  else (time cannot elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \overline{\Sigma} \setminus 2^{\mathcal{C}_e}\right\}$
- $\mathcal{RF}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, r, I\right) =$
  $\left\{(X_e\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, r[X_e], I\right)\right\} \cup$
  $\cup \left\{(a_i\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\,(P_i, r, \mathtt{tt}) \wedge r \models \varphi_i\right\}$
  if $r \models I$ and $r \neq r_{max}$ and $succ(r) \models I$ (time can elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \left(\Sigma^{\checkmark} \setminus \{a_j \mid r \models \varphi_j\}\right) \cup 2^{\mathcal{C}_i} \cup \left(\Delta \setminus \{\delta_{clocks(r)}\}\right)\right\}$
  $\cup \left\{(\delta_{clocks(r)}\hat{\ }t, F) \mid (t, F) \in \mathcal{RF}\left(\Box_{i=1}^{N}(a_i, \varphi_i) \rightarrow P_i, succ(r), I\right)\right\}$
  else (time cannot elapse)
  $\left\{(\langle\rangle, F) \mid F \subseteq \left(\Sigma^{\checkmark} \setminus \{a_j \mid r \models \varphi_j\}\right) \cup 2^{\mathcal{C}_i} \cup \Delta\right\}$
- $\mathcal{RF}(\sqcap_{i=1}^{N} P_i, r, I) = \bigcup_{i=1}^{N} \mathcal{RF}(P_i, r, \mathtt{tt})$
- $\mathcal{RF}(P \underset{A}{\|} Q, r, I) = \{(t, F \cup G) \mid F \setminus (A \cup \{\checkmark\}) = G \setminus (A \cup \{\checkmark\})$
  $\wedge\ \exists t_1, t_2 \,.\, (t_1, F) \in \mathcal{RF}(P, r, I) \wedge (t_2, G) \in \mathcal{RF}(Q, r, I)$
  $\wedge\ t \ \mathsf{synch}_A\ (t_1, t_2)\}$

- $\mathcal{RF}(P \setminus A, r, I) = \{(s \setminus A, F) \mid (s, A \cup F) \in \mathcal{RF}(P, r, I)\}$
- $\mathcal{RF}(f[P], r, I) = \{(t, F) \mid (f^{-1}(t), f^{-1}(F)) \in \mathcal{RF}(P, r, I)\}$
- $\mathcal{RF}(P; Q, r, I) = \{(s, F) \mid (s, F') \in \mathcal{RF}(P, r, I) \text{ with } F' \setminus \{\checkmark\} = F\}$
  $\cup\{(s\hat{\ }t, F) \mid s\hat{\ }\langle\checkmark\rangle \in \mathcal{RT}(P, r, I) \land (t, F) \in \mathcal{RF}(Q, \mathsf{after}(s), \mathtt{tt})\}$

The rules for region failures are similar to those for region traces; Figure 4 illustrates the correspondence with the operational model and Theorem 5.3 will relate the rules above with the operational model in detail.

Recursion. Again, we define the semantics of recursion $Z = P$ as the least fixed point of P, that is, $\mathcal{RF}(Z = P, r, I) = \bigcup_{n=0}^{\infty}(P^n(DIV), r, I)$. The following theorem states that recursion is well defined for guarded processes.

Theorem 5.2. *RegionFailures* is a monotonic function with respect to all operators and the least fixed point operator is well-defined for guarded processes.

Proof. Again, we consider only the novel operators as the result is clear for standard CSP operators.

- Reset: let us consider two processes, $\{\!|X|\!\}P_1$ and $\{\!|X|\!\}P_2$, such that $P_1 \sqsubseteq_{\mathcal{RF}} P_2$. From the rule for reset, the first set of region failures $(\{(\langle\rangle, F) \mid F \subseteq \Sigma^{\checkmark} \cup \Delta \cup (C_i \setminus X)\})$ is common between the two processes, while, clearly, we have inclusion of $\{((X \cup X_e)\hat{\ }t, F) \mid X_e \subseteq \mathcal{C}_e \land (t, F) \in \mathcal{RF}(P, r[X \cup X_e], I))\}$ if and only if we have inclusion of the failure sets of the components, which is true by induction.
- Invariant: this case is immediate from the definition of region refusals for the invariant operator.
- External choice: we can use the same arguments as for trace semantics; the operator is defined in terms of itself, but before using the recursive rule both processes can take the same actions and refuse the same sets. If we analyse the rule for external choice, both processes can execute a finite sequence of delays, possibly interleaved by external clock resets; moreover, they can refuse the same sets for each action. Then both processes can execute the same action, under the same starting conditions, and the recursive step can be applied, as we did for the trace semantics. The arguments used for region traces also apply in this case.

□

5.3. Congruence with the Operational Semantics

We show that the denotational semantics we have given in this section is congruent with the operational semantics of Section 4. This is not surprising since we have modelled the trace semantics on region automata, but it is nevertheless important as it permits model checking of refinement relations using the operational model as done for CSP [Ros94].

Region automata with external clocks. We will prove the congruence result by structural induction on the terms. We have to show that, for every process, the failure semantics of the region automaton induced by the timed automaton obtained with the operational semantics is the same as the failure semantics obtained by applying the denotational rules.

In order to do this, we have to add the notion of external clocks to the definition of region automata given in Section 2. As shown in Figure 4, the idea of external clocks is that, at any point in the computation, a component must be able to synchronise with an external clock reset, and thus unable to refuse such actions. Given a timed automaton $\mathcal{A} = (L, \bar{l}, \Sigma, \mathcal{C}, \mathcal{I}, \kappa, \rightarrow)$, we partition its set of clocks $\mathcal{C}$ into the set of internal clocks $\mathcal{C}_i$, that is, those clocks that are explicitly used by $\mathcal{A}$, and its complement, the external clocks $\mathcal{C}_e$. Then the rules for transitions of region automata of Definition 2.2 have to be modified as follows: from any state (l, r), $(l, r) \xrightarrow{X_e}_r (l, r[X_e])$, for all $X_e \subseteq \mathcal{C}_e$, must be enabled and the rule for reset of clocks $X \in \mathcal{C}_i$ must be modified by allowing external resets, that is, $(l, r) \xrightarrow{X \cup X_e}_r (l, r[X \cup X_e])$, for all $X_e \subseteq \mathcal{C}_e$. We kept explicit information about the reset of the empty set of clocks in order to be able to synchronise with other clocks. The other types of actions are not modified; moreover, note that the same actions are enabled from different regions if they disagree only on the values of external clocks.

Congruence. We can now prove the main result.

Theorem 5.3. The function $\mathcal{RF}$ is a congruence with respect to the operational semantics. That is, for all CCSP terms in normal form, regions r and invariants I, the following holds:

$$\mathcal{RF}(P, r, I) = \mathcal{F}(R(P, r, I))$$

where $\mathcal{F}(R(P, r, I))$ denotes the set of stable failures of the region automaton corresponding to process P under the given starting conditions.

Proof. We prove the result by structural induction on the syntax. In Figure 4 the region automata corresponding to some operators are shown, to highlight the correspondence between the operational model and the denotational rules for region failures. We start from the base cases:

- $STOP$: this process can only let time elapse or synchronise on external clock resets; these are the clauses of the rule for $\mathcal{RF}(STOP)$. All actions except for external clock resets or the next delay can be refused.
- $SKIP$: in this case the process can let time elapse (only while the invariant remains satisfied), synchronise on external clocks or successfully terminate by executing a $\checkmark$ action. All these are the clauses of the rules for failures. At each step the process can refuse any action except for $\checkmark$ and external clock resets; after the $\checkmark$, any set of actions is refused.

The inductive steps:

- $\{\![X]\!\}\phi \triangleright P$: we deal with these two operators together. These operators introduce the invariant and reset the clocks in X. From the initial state of the operational model, clocks $X \cup X_e$, for all $X_e \subseteq \mathcal{C}_e$ can be reset, and all other actions must be refused; this corresponds to the denotational rule. After the transition, the inductive step applies. Since we assume that we deal with processes in normal form, we know that the sub-term P has no clock reset; this shows the importance of having a normal form, so that we have a unique reset action.
- $P = \sqcap_{i=1}^{N} P_i$: in the operational model a τ action must be taken immediately, therefore the only traces allowed are those of the components and the process can refuse everything that the components can refuse.
- $P = \Box_{i=1}^{N} (a_i, \varphi_i) \to P_i$: Figure 4 helps understand the rule for external choice and why it is congruent to the operational semantics. Similarly to $SKIP$, before the choice is resolved and an action a_j is executed, thus reducing to the inductive step, the operational model can reset external clocks or let time elapse; for instance, from any state (P, r) of the figure, the automaton can either execute the actions whose guards are satisfied, or it can let time elapse and behave like $(P, succ(r))$, or reset some external clocks X_e and behave like $(P, r[X_e])$. All other actions are refused. All these cases correspond to the clauses of the denotational rule.
- $P = P \underset{A}{||} Q$: by induction, we know that $\mathcal{RF}(P, r, \texttt{tt}) = \mathcal{F}(R(P, r))$ and that $\mathcal{RF}(Q, r, \texttt{tt}) = \mathcal{F}(R(Q, r))$ (the invariants must be true for components of the parallel operator because of the structure of the syntax). Following standard CSP arguments, it can be shown that $\mathcal{RF}(P \,||\, Q, r, \texttt{tt})$ has the same traces obtained by synchronising $R(P, r)$ and $R(Q, r)$ on the actions in $A \cup \Delta \cup 2^{\mathcal{C}}$, that is, the traces of $R(P, r) \,||\, R(Q, r)$. We have to show that this region automaton corresponds to the region automaton obtained by operational semantics of the parallel composition, that is, $R(P \,||\, Q, r) = R(P, r) \,||\, R(Q, r)$. Recall that the state space for region automata corresponding to a CCSP term is given by the terms in $\mathbf{CCSP}^+$ extended with the operator $free$ of timed automata, which we denote by $\overline{\mathbf{CCSP}^+}$. Note the different role in this case of free, the operator on CCSP terms, and $free$, the operator on the locations of timed automata; for instance, from the operational semantics and the definition of region automata, the region automaton corresponding to the process $\mathsf{free}(P)$ has an initial reset action on the empty set of clocks. We get $R(P, r) \,||\, R(Q, r) = ((\overline{\mathbf{CCSP}^+} \times R) \times (\overline{\mathbf{CCSP}^+} \times R), \Sigma \cup \Delta \cup 2^{\mathcal{C}}, ((P, r), (Q, r)), \longrightarrow)$ and $R(P \,||\, Q, r) = ((\overline{\mathbf{CCSP}^+} \times R), \Sigma \cup \Delta \cup 2^{\mathcal{C}}, (P \,||\, Q, r), \to)$. Let us show that each transition of one is matched by a transition of the other.
  - $((P', r'), (Q', r')) \xrightarrow{a} ((P'', r'), (Q'', r'))$, with $a \in A$, is a transition for the parallel if both components can make this transition, which is true if both $P' \xrightarrow{a,\varphi_1} P''$ and $Q' \xrightarrow{a,\varphi_2} Q''$ with $r' \models \varphi_1$ and $r' \models \varphi_2$.

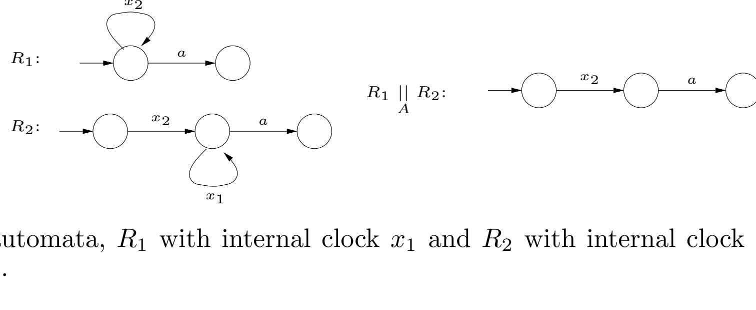

Fig. 5. Two region automata, R_1 with internal clock x_1 and R_2 with internal clock x_2, and their parallel composition $R_1 \underset{A}{||} R_2$.

This is also the condition for synchronisation for the parallel: $P' \,||\, Q' \xrightarrow{a,\varphi_1 \wedge \varphi_2} P'' \,||\, Q''$ allows an a action in the correposponding region automaton if both guards are satisfied and if both components are willingto take an a action.

- $((P', r'), (Q', r')) \xrightarrow{a} ((P'', r'), (Q', r'))$, with $a \in \overline{A} \cup \{\tau\}$, is a transition for the parallel if (P', r') can independently execute this action, which is true if $P' \xrightarrow{a,\varphi} P''$. After executing a, P'' can reset its set of clocks, and Q' will be able to synchronise on any external set of clocks. The parallel composition, $P \,||\, Q$, can make the same transition under the same conditions, that is, $P' \,||\, Q' \xrightarrow{a,\varphi} P'' \,||\, \mathsf{free}(Q')$. At this point P'' can reset its set of clocks, thus behaving in the same way as above.
- $((P', r'), (Q', r')) \xrightarrow{X} ((P'', r'[X]), (Q'', r'[X]))$ is a transition for the product if both components can reset the clocks in X; suppose that $\kappa(P') = X_1$, $\kappa(Q') = X_2$ and that we can partition X into three sets $X_1 \cup X_2 \cup X_3$, such that X_1 and X_2 are internal clocks for P' and Q', respectively, and X_3 are external to both. Then $(P', r) \xrightarrow{X} (P', r[X])$ because P resets its clocks X_1 and it is willing to synchronise on any external clock reset; the same applies to Q'. Therefore, they can synchronise on this action.
  On the other hand, the composition $P' \,||\, Q'$ will have $X_1 \cup X_2$ as the set of internal clocks to be reset ($\kappa(P' \,||\, Q') = X_1 \cup X_2$), and X_3 as the set of external clocks on which it is willing to synchronise. Therefore, $(P' \,||\, Q', r')$ will reset its own clocks $X_1 \cup X_2$, and it will also be able to synchronise on all other clocks, thus is must be able to execute the reset X.
- $((P', r'), (Q', r')) \xrightarrow{\delta} ((P', succ(r')), (Q', succ(r')))$, for $\delta \in \Delta$, if both components can execute such action, that is if they can both let time elapse and $succ(r') \models I(P')$ and $succ(r') \models I(Q')$, which is the condition for the $P' \,||\, Q'$ to let time elapse. Therefore also $(P' \,||\, Q', r') \xrightarrow{\delta} (P' \,||\, Q', succ(r))$.

Note that it is not possible to define a transition through which the two components end up in states with different regions. Therefore, we can consider the state spaces of the two automata as equivalent. The inverse inclusion is shown using exactly the same arguments.
We point out that after two terms are combined by parallel composition, they join their internal clocks into a new set of internal clocks. For this reason, they synchronise on the reset operations and they discard all the reset operations concerning internal clocks that have not been used; if they have no more external clocks, then there are no more “dangling” external clock reset actions (see Figure 5). Also note that, from the structure of the syntax, the first actions that both processes execute are reset actions, on which they immediately synchronise.

- $P; Q$: this case is trivial as the failures of the sequential composition are those of the first component (excluding those ending in $\checkmark$) and those of the second concatenated to the first. This is exactly what happens in the operational model. Note we assume that Q will start the execution under a true invariant: this is the case because the syntax of CCSP allows only P terms after the ; operator and all P terms have no invariant.
- $f[P]$: this case is trivial as the same renaming function is applied both to the failures of the semantic model and to the transitions in the operational model.
- $P \setminus A$: consider a trace $t \in \mathcal{RF}(P, r, I)$. It easy to see that if t is a trace of $R(P, r, I)$ then $t \setminus A$ is a trace of $\mathcal{RF}(P \setminus A, r, I)$ as all actions in A have become τ actions. Moreover, since $(t, F \cup A)$ is a refusal of

$\mathcal{RF}(P, r, I)$, no actions in A are enabled after t, therefore no τ action is introduced from the state after t, which therefore remains a stable state and refuses all the actions in F that have not been hidden.

Finally, we prove the result for recursion. The proof follows very closely the proof for the congruence theorem for CSP as described in [Ros98]. The main difference with our proof is that we have to rearrange the original one in order to consider tuples (CCSP process, starting region, starting invariant) as our basic object. These are the states of the labelled transition system of the operational semantics and also the arguments of the semantic function $\mathcal{RF}$. Once this correspondence is defined, the proof follows easily. So far in this proof we have ignored that a given CCSP term P might contain free process identifiers (*Ide*). Therefore what we need is a semantics that keeps track of these identifiers and introduce *environments*, that is, functions that associate a set of failures to an identifier. Formally,

$$Env_{\mathcal{RF}} : Ide \times R \times \mathcal{B}(\mathcal{C}) \longrightarrow \mathsf{RFailures}$$

So, we obtain the following semantic function for region failures:

$$S_{\mathcal{RF}} : (\mathbf{CCSP} \times R \times \mathcal{B}(\mathcal{C})) \longrightarrow Env_{\mathcal{RF}} \longrightarrow \mathsf{RFailures}$$

We define the equivalent operator for the operational semantics:

$$\xi : Ide \longrightarrow \overline{\mathbf{CCSP}}$$

where $\overline{\mathbf{CCSP}}$ is the set of CCSP terms with no free identifiers. We need to prove that for all substitutions ξ and all CCSP terms P,

$$\mathcal{F}(subst(R(P, r, \phi), \xi)) = S_{\mathcal{RF}}(P, r, \phi)\overline{\xi}$$

where $subst(R(P, r, \phi), \xi)$ is the term obtained by replacing each state with free identifier (Z, r, ϕ) by $(\xi(Z), r, \phi)$ and $\overline{\xi}(Z, r, \phi) = \mathcal{F}(R(\xi(Z), r, \phi))$. The proof we have given for all operators but recursion still holds in this case; the same arguments we have used still apply if we use this new semantic function instead, but we preferred to keep the notation simple where possible. We use the λ notation for recursion in this proof, that is, write $\mu Z.P$ for $Z = P$.

We prove that $\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi))$ is the least fixed point for the function Υ, mapping $\alpha \in \mathsf{RFailures}$, for a given ξ, to $\Upsilon(\alpha) = S_{\mathcal{RF}}(P, r, \phi)\overline{\xi}[\alpha/(Z, r, \phi)]$. The fixed point of this function gives the semantics of recursion. For any ξ, let $\mu Z.P'$ the term where all free identifiers other than Z have been substituted according to ξ, so that $subst(R(\mu Z.P, r, \phi), \xi) = R(\mu Z.P', r, \phi)$, since Z is not free in P.

$$
\begin{aligned}
&\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi)) = \\
&\qquad = \mathcal{F}(R(\mu Z.P', r, \phi)) && \text{by definition of } P' \\
&\qquad = \mathcal{F}(subst(R(P, r, \phi), \xi[\mu Z.P/Z])) && \text{by semantics of recursion} \\
&\qquad = S_{\mathcal{RF}}(P, r, \phi)\overline{\xi[\mu Z.P/Z]} && \text{inductive step } (P \text{ is simpler than } \mu Z.P) \\
&\qquad = S_{\mathcal{RF}}(P, r, \phi)(\overline{\xi}[\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi))/(Z, r, \phi)]) && \text{by definition of } \overline{\xi} \\
&\qquad = \Upsilon(\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi))) && \text{by definition of } \Upsilon
\end{aligned}
$$

This proves that $\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi))$ is a fixed point for Υ. We now have to prove that it is the least fixed point. The proof to show this follows exactly the proof of [Ros98], as we work with guarded recursion.

To show that it is the *least* fixed point, we have to show that every failure (t, F) of $\mathcal{F}(subst(R(\mu Z.P, r, \phi), \xi))$ is a failure of $\Upsilon^n(\{(\langle\rangle, \{\})\})$, for some n, as $\{(\langle\rangle, \{\})\}$ is the minimal element in the failures model. Consider such failure (t, F), then there must be some $t' \in (\overline{\Sigma}^{\checkmark})^*$ such that $t = t' \setminus \{\tau\}$ (t with its τ actions removed) and $subst(R(\mu Z.P, r, \phi), \xi) \xrightarrow{t'} Q$, with Q some other state of the automaton (the region and invariant are not relevant). Let $N = \#t$, then the recursion cannot be unfolded more than N times during the derivation of (t, F) if the process is guarded. Define the following processes:

$$
\begin{array}{rcll}
Z_n & = & P'[Z_{n+1}/Z] & n \in \mathbb{N} \\
Y_n & = & P'[Y_{n+1}/Z] & n \in \{0..N-1\} \\
Y_N & = & DIV &
\end{array}
$$

Clearly, $R(Z_0, r, \phi)$ has the same operational semantics as $subst(R(\mu Z.P, r, \phi), \xi)$; it just keeps track of the number of times recursion has been unfolded. Y_n behaves like Z_n for the first $N-1$ unfoldings of recursion,

then it diverges. Since $\#t = N$, (t, F) must also be a behaviour of Y_0, that is, $(t, F) \in \mathcal{F}(Y_0, r, \phi)$. We also have $S_{\mathcal{RF}}(Y_N, r_N, \phi_N) = \{(\langle\rangle, \{\})\}$, $S_{\mathcal{RF}}(Y_{N-1}, r_{N-1}, \phi_{N-1}) = \Upsilon(\{(\langle\rangle, \{\})\})$, ... and $S_{\mathcal{RF}}(Y_0, r, \phi) = \Upsilon^N(\{(\langle\rangle, \{\})\})$, because $\mathcal{F}(subst(R(P, r, \phi), \xi[Q/Z])) = S_{\mathcal{RF}}(P, r, \phi)\bar{\xi}[\forall r', \phi'\ \mathcal{F}(Q, r', \phi')/(Z, r', \phi')]$ for every closed term Q. It follows that $(t, F) \in \Upsilon^N(\{(\langle\rangle), \{\}\})$, which is what we had to prove. $\square$

This result is of course also true for trace semantics, since stable failure semantics is strictly stronger than region failure semantics.

6. Model Checking Clocked CSP

In this paper we give the theoretical foundations for Clocked CSP, guided by the desire to formulate the foundations of automatic verification of timed properties for CSP processes extended with time. CCSP cannot be model checked directly and further work is necessary to achieve this. In this section we outline how this can be done and describe possible future research.

6.1. The Main Idea

We would like to be able to automatically verify the refinement relations described above by using the FDR2 model checker for CSP [For93]. This is possible for finite-state systems thanks to the congruence result. We also consider only processes with no external clocks. In this section we investigate the usefulness of this result, that is, what kind of properties can be verified by using the region refinement.

Our approach is limited, the main reason being that our semantics makes explicit references to clock names: we argued that this was necessary in order to obtain compositionality and decidability, but this turns out to limit the power of region refinement. Consider the following example:

$P_1 = \{\!|x|\!\}(a, x = 1) \rightarrow \{\!|x|\!\}(a, x = 1) \rightarrow SKIP$
$P_2 = \{\!|x|\!\}(a, x = 1) \rightarrow (a, x = 2) \rightarrow SKIP$

P_1 and P_2 would be distinguished as P_1 resets clock x twice, while P_2 does it only once. Clearly, we would like to identify them (they are timed bisimilar).

A first step to overcome this problem is to define refinement "up to" clock renaming. Next, we need to find a way to use the refinement in a meaningful way. The main idea is to distinguish between functional properties (dependent only on actions) and timed properties (dependent on the clocks). Then we can highlight only the timed properties we are interested in verifying by using auxiliary processes, and use refinement in such a way that a process P refines a process Q if it preserves some timed properties and if it functionally refines it in the traditional sense. This is similar to the idea of timewise refinement [Sch97], where system analysis distinguishes between timed and functional aspects.

To explain this, let us consider the traces model and note that if we hid all clock actions (either resets or delays) from traces we would be able to verify functional (untimed) refinement. If we hid only those clock actions pertaining to a given subset of clocks, then we would be able to verify the refinement with respect to only that subset of clocks, possibly the subset that describes the timed behaviour that we are interested in.

The hiding of clocks can be easily defined for $\mathcal{RT}$ and $\mathcal{RF}$ in the same way as we have defined hiding and renaming: if we want to hide a set of clocks X, then all clock reset actions Y are renamed to $Y \setminus X$ if $Y \setminus X \neq \emptyset$ and hidden otherwise, while a delay action δ_Y is renamed to $\delta_{Y \setminus X}$ if $Y \setminus X \neq \emptyset$ and hidden otherwise. The hiding of clocks can be defined only as a top level operator for processes with no external clocks, as otherwise most states would become unstable and different components would not be able to synchronise. This is why we talk about refinement between two processes P and Q with respect to a certain set of clocks X, meaning refinement in which all clocks not in X are hidden for each of the processes.

This approach is made clear by the following example.

6.2. A Model Checking Example

We give an example (also used in [OW03]) to illustrate model checking with the region semantics approach. Assume that we want to check if a process respects the bounded invariance property $\Box(a \Rightarrow \Box_I \neg b)$ where

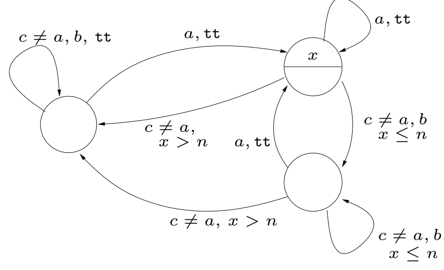

Fig. 6. The timed automaton corresponding to the bounded invariance property.

$I = [0, n]$ is a closed interval starting from 0. This means that, whenever an action a is performed, the process cannot execute a b for n successive time units. This is a safety property that can be specified as a trace property. The most non-deterministic process that respects this property is:

$$S = (a, \mathtt{tt}) \to \{\!|x|\!\}S_1 \,\Box\, \left(\Box_{c \neq a}(c, \mathtt{tt}) \to S\right)$$
$$S_1 = (a, \mathtt{tt}) \to \{\!|x|\!\}S_1 \,\Box\, \left(\Box_{c \neq a, b}(c, x \leq n) \to S_1\right) \,\Box\, \left(\Box_{c \neq a}(c, x > n) \to S\right)$$

The timed automaton corresponding to this process is pictured in Figure 6 (τ actions from process definition are omitted for simplicity). Every time an a action is performed, the system moves to the top-right location, where clock x is reset. The control stays on the right side of the system until the value of x exceeds n and an action which is not a is performed. No b action can be executed while $x \leq n$, that is, within n time units after the last a. This is reflected in the corresponding region automaton, where no b action is allowed until enough δ actions have occurred. The process S serves as the specification process, against which implementation processes are checked. Given a process P, in order to check whether P refines the specification S, we need to reset the clock x each time an action a is performed. We can do this by defining a process $T = (a, \mathtt{tt}) \to \{\!|x|\!\}T$, where x is a clock used only by T, i.e. it is an external clock for P, that performs this function. The refinement that we need is the following:

$$S \sqsubseteq_{\mathcal{RT}} \quad P \underset{\{a\}}{||} T$$

where the verification is with respect to clock x only. This works because the specification S does not allow the execution of any b before the value of clock x is greater than n, and the passage of time is recorded by delay actions. Clock hiding is defined as a combination of hiding and renaming on the set of traces (and the operational model): if we hide a clock x, then we need to hide all the reset or delay actions that involve it (or rename the action if other clocks are reset or change region at the same time). This is why we encode the information as to which clocks cause a region change in delay actions instead of having a single generic delay action that would not permit the hiding of clocks. Finally, our approach admits successive refinements: considering the example above, if the following two refinements hold

$$S \sqsubseteq_{\mathcal{RT}} \quad P_1 \underset{\{a\}}{||} T \sqsubseteq_{\mathcal{RT}} \quad P_2 \underset{\{a\}}{||} T$$

then both P_1 and P_2 respect the bounded invariance property and, in addition, there is functional refinement between P_1 and P_2.

The traces model suffices to verify safety properties, since one can consider all the traces that do not contain undesired behaviour. If we want to verify liveness properties, we need to use the failures model. The idea is the same as described above, that is, we verify refinement with respect to a subset of clocks only. It is possible to verify properties such as strong bounded response using a similar idea.

One could attempt the above examples with FDR2, but the complexity of manually translating CCSP processes into the equivalent CSP processes makes this task error prone and tedious. Given the complexity of building a region automaton, and its resulting size, it would be desirable to make the process above automatic, for which more research is needed. We tested our ideas with small toy examples in the following way: firstly, we defined small CCSP processes, then we translated them into the equivalent timed automaton

and, from here, into the resulting region automaton. At this point we manually specified CSP processes equivalent to the region automata and verified refinements between them with FDR2.

6.3. Considerations

We have shown how the semantics based on regions can be used to verify refinement relations between processes; by using appropriate additional processes and by checking refinement only with respect to some clocks, we were able to verify both functional properties of processes and also some timed properties, though, admittedly, a subclass of the latter. One advantage of our approach is that we have been able to obtain a decidable semantics that also allows chains of refinements in the same model.

It is worth pointing out that in this paper we are not concerned with efficiency: region automata can be exponential in size, thus making the checking of refinement exponential. Instead, the main objective was to find a decidable semantics that could be used to automatically verify certain timed properties of processes through refinement. It would be interesting to find out whether more efficient techniques, e.g. zones, can be applied to improve our approach. Another drawback is the fact that we have to manually write process specifications for each timed property, as we did in the example above. A possible line of research would be to automatically generate such processes from some appropriate timed logic.

7. Conclusions

In this paper we have described a proposal for a timed extension to CSP, called Clocked CSP. We have defined its semantics and the corresponding refinement relations, demonstrating how one could use the model checker FDR2 to verify such relations, and also certain timed properties of systems. Future work will include improving the complexity of model checking, and investigating whether it is possible to use known efficient techniques for timed automata in our case. It would also be interesting to define a logic that could be verified with our technique. Finally, we plan to compare our approach with other similar approaches to extending CSP with real time.

References

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):1–34, 1993.

[AD92] R. Alur and D. L. Dill. The theory of timed automata. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 45–73. Springer-Verlag, 1992.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH99] R. Alur, L. Fix, and T.A. Henzinger. Event clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.

[CK03] S. Cattani and M. Kwiatkowska. CSP + Clocks: a process algebra for timed automata. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proc. 3rd Workshop on Automated Verification of Critical Systems (AVoCS'03)*, Technical Report DSSE-TR-2003-2, University of Southampton, pages 50–63, April 2003.

[D'A99] P. R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, Department of Computer Science, University of Twente, November 1999.

[DJR+92] J. Davies, D. Jackson, G. Reed, J. Reed, A. Roscoe, and S. Schneider. Timed CSP: Theory and practice. In *Proceedings of REX Workshop, Nijmegen, LNCS 600, Springer-Verlag*, 1992.

[For93] Formal Systems (Europe) Ltd. Failures Divergence Refinement – User manual and tutorial, 1993. Version 1.3.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *ICALP '92: Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 545–558. Springer-Verlag, 1992.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Jac92] D. M. Jackson. *Logical Verification of Reactive Software Systems.* PhD thesis, University of Oxford, 1992.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Oua01] J. Ouaknine. *Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems.* PhD thesis, Oxford University, 2001.

[OW03] J. Ouaknine and J. Worrell. Timed CSP = closed timed ϵ-automata. *Nordic Journal of Computing*, 10(2):99–133, 2003.

[OW04] J. Ouaknine and J. B. Worrel. On the language inclusion problem for timed automata: Closing a decidability gap.

In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland*, pages 54–63. IEEE Computer Society, 2004.

[Ros94] A.W. Roscoe. Model-checking CSP. In *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall, 1994.

[Ros98] A.W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.

[RR88] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.

[Sch97] S. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28(1):43–90, 1997.

[Sch99] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley and Sons, 1999.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. (Kronos user's manual release 2.2), 1997.

[Yov98] S. Yovine. Model checking timed automata. In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 114–152. Springer-Verlag, 1998.

[YPD95] W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII*, pages 243–258. Chapman & Hall, Ltd., 1995.