Compositional Verification of Timed Statecharts

Francesca Levi

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56100, Pisa, Italy
email:levifran@di.unipi.it

Abstract. We propose a compositional proof system for the verification of a dicrete timed process language $\mathcal{TSP}$ with minimal and maximal delays associated to actions. For expressing real-time properties we consider a discrete extension of μ-calculus with freeze quantification over clocks and clock constraints. The language $\mathcal{TSP}$ is parametric in the set of basic actions and it is characterized by an operator of process refinement, which permits a suitable translation in it of a discrete timed version of statecharts. The proof system is proved to be sound in general and complete for the class of regular finite state processes. It is indeed complete for processes corresponding to statecharts.

1 Introduction

Nowadays many applications are time-critical. Typical examples are transportation systems, railways control systems and signal-processing systems. The languages for programming real-time reactive systems are required to express timing contraints on the interaction of the system with the environment and their correctness depends also on the quantitative aspect of time. Hence, there is an increasing interest in the study of both the practical and the theoretical aspects of real-time logics [4, 3], that are capable of expressing quantitative temporal properties. In this framework compositional verification is very important for developing in a modular way complex specifications. As opposed to verification over the whole system, it makes it possible to leave undefined parts of the system and still to reason about it. Moreover, it has the advantage that, when a part is modified, only the verification concerning this particular component must be redone.

In this paper we propose a compositional verification method for a discrete timed process language called $\mathcal{TSP}$. It is a CCS-like language where actions can be constrained between a minimal and maximal delay and it is parametric in the set of basic actions and in two operations over actions of parallel composition and conflict. Moreover, it is characterized by an operator of process refinement and by explicit locations.

The design of $\mathcal{TSP}$ is motivated by the aim to have a quite general and simple process language that is adequate for modelling a discrete timed version of statecharts. Statecharts [9] is a synchronous formalism for the specification of reactive and real-time systems that overcome the limitations of state-transition diagrams by introducing notions of *parallelism*, *hierarchy* and *broadcast communication*. The language is based on the synchrony hypothesis [6]:"the reaction of the system to inputs is instantaneous". The instantaneous reaction of the system (a step) to inputs is given by a maximal set of parallel transitions *triggered* by the input. However, due to well known causality problems the formal definition of *step* is still quite controversial [10, 20, 17]. [15, 16] show a translation of classical untimed statecharts into an untimed version of $\mathcal{TSP}$ which agrees with the approach of [20], where a step is required to satisfy some additional requirements of consistency and causality. The basic idea is to build a process with locations corresponding to the basic states of the statechart and actions corresponding to transitions and representing the associated communication. However, the informations concerning the communication associated to transition *trigger/action*, where *trigger* expresses an enabling condition on input events and *action* gives the set of instantaneuosly generated events, is not sufficient to achieve a compositional semantics. Therefore, it is necessary to take into account in basic actions also the causal relation between inputs and outputs. The operator of refinement allows us to naturally represent the statecharts hierarchy.

The translation between statecharts and the process language can be extended to the timed discrete case [16]. However, the interest of the language is not limited to statecharts, since it can be instantiated to other process calculi with different communication mechanisms, such as Synchronous CCS [18] and the Calculus of Broadcasting Systems [21], by properly choosing the set of basic actions and the related operations.

In order to express quantitatively bounded temporal properties we consider a discrete real-time extension of propositional μ-calculus [13, 22] (RTL_μ) with freeze quantification over clocks and clock constraints. The logic is powerful enough to express the typical real-time properties of bounded response and bounded invariance. We define a compositional proof system for checking RTL_μ properties of $\mathcal{TSP}$ processes. In the syle of [5] local and compositional reasoning are combined toghether. There are two types of proof rules:

1. logical rules that work on the structure of the formula. Fixpoint formulas are treated by a generalization of the method of tagging fixpoints [24].
2. rules for the next modalities that work on the structure of the process and reduce a satisfaction problem for a compound process into satisfaction problems for its components.

In this approach the construction of a proof is driven by the syntax of the formula and of the process without reference to the underlying transition system which is typically infinite. The proof system is proved to be sound in general and complete for the class of regular finite state processes. It is indeed complete for processes corresponding to timed statecharts. A similar proof system for an untimed version version of $\mathcal{TSP}$ has been developed in [16, 14].

The paper is organized as follows. Section 2 presents the language $\mathcal{TSP}$ and its labelled transition system semantics. The logic RTL_μ is introduced in section 3. Finally, section 4 presents the compositional proof system.

2 The language $\mathcal{TSP}$

We introduce the process language $\mathcal{TSP}$. We assume a set *Act* of basic actions, a set *Loc* of locations and a set Var of variables. Two operations are assumed on basic actions, i.e. $\times : Act \times Act \to 2^{Act}$ (*parallel composition*) and $\# : Act \cup \{*\} \to 2^{Act}$ (*conflict*). We require $\times$ and $\#$ to satisfy the following properties:

1. for each $a_1, a_2, a_3 \in Act$, $a_1 \times (a_2 \times a_3) = (a_1 \times a_2) \times a_3$ (associativity), where $\times$ is the straightforward extension to sets;
2. for each $a_1, a_2, a_3, a_4 \in Act$, such that $a_2 \in \#(a_5)$ and $a_4 \in \#(a_6)$ for some $a_5, a_6 \in Act$, if $a \in a_1 \times a_2$ and $a \in a_3 \times a_4$, then $a_2 = a_4$.

Intuitively, if $\#(a) = \emptyset$ then a is autonomous and it requires no enabling (for instance inputs) from the environment. In contrast, an action $a' \in \#(a)$ espresses conditions on the environment which prevent the performance of a. For instance, if a requires an input i, then an action representing " i is not produced" is in conflict. In this case we require that a does not depend on timing constraints. The special action $*$ is used for obtaining a generic behaviour of the environment.

Definition 1. Processes p of $\mathcal{TSP}$ are defined as follows:

$$\begin{aligned} b &::= nil \mid b_1 + b_2 \mid [m, M]a.t \\ t &::= \ell :: b \mid rec\ x.t \mid t \bigtriangledown p \mid x \\ p &::= \ell :: t \mid \ell :: p_1 \times p_2 \end{aligned}$$

where $a \in Act$, $x \in Var$, $\ell \in Loc$, $m \in N$, $M \in N \cup \infty$ and $m \leq M$, $M \neq 0$. For $a \in Act$ with $\#(a) \neq \emptyset$, we require $m = 0$ and $M = \infty$.

The constructors $+$, rec, $\times$ stand as usual for *choice*, *recursion* and *parallel composition*. Timed prefixing is denoted by $[m, M]a.$. Roughly speaking, the action a can be performed only after its minimal delay m and cannot be delayed after its maximal delay M. The constructor of process refinement $\bigtriangledown$ is typical of our language and permits very compact and hierarchical specifications. It is inspired by the statecharts hierarchy. Locations are introduced as the counterparts of basic states in statecharts.

For $p = \ell :: t$ and $p = \ell :: p_1 \times p_2$, $main(p) = \{\ell\}$ and $loc(p) = \{\ell\} \cup loc(t)$, $loc(p) = \{\ell\} \cup loc(p_1) \times loc(p_2)$ respectively, where $loc(\ell :: b) = \{\ell\}$, $loc(rec\, x.t) = loc(t[rec\, x.t/x])$, $loc(t \bigtriangledown p) = loc(t) \cup loc(p)$ and $loc(x) = \emptyset$. Moreover, $rel(p) = loc(p) \backslash main(p)$. In other words $loc(p)$ represents the current configuration, namely the set of entered locations (basic states). In the following we consider well-formed and closed processes, where the locations of all subprocesses are distinct. We denote by $\mathcal{CTSP}$ the set of closed and well-formed processes.

2.1 The semantics of $\mathcal{TSP}$ processes

We define a labelled transition system semantics of closed $\mathcal{TSP}$ processes. Such a semantics is based on the idea of considering clocks corresponding to locations. The value of a clock represents the time that has been elapsed by the process in the corresponding location. States are given by a process together with an assignment of values to its clocks and transitions represent actions of the process together with an increment of one time unit in the values of clocks.

We need some preliminary concepts. For a set C of clocks, $\pi : C \to N$ is a *clock assignment* over C. For $\delta \in N$, we denote by $\pi + \delta$ the clock assignment, such that for each $x \in C$, $\pi(x) + \delta$ (and analogously for $\pi - \delta$). Moreover, for $C_1 \subseteq C$, $\pi[C_1 := 0]$ denotes the assignment such that, $\pi(y) = \pi[C_1 := 0](y)$, for $y \notin C_1$, and $\pi[C_1 := 0](x) = 0$ for $x \in C_1$. Let C be a set of clocks. A *clock constraint* is inductively defined by the grammar $\omega ::= x\mathcal{R}n \mid x + n\mathcal{R}y + m \mid \omega_1 \wedge \omega_2 \mid \neg\omega$ with $x, y \in C$ and $n, m \in N$, $\mathcal{R} \in \{\geq, \leq\}$. We denote by $\Omega(C)$ the set of clock constraints over clocks C. Clock constraints of $\Omega(C)$ are evaluated with respect to clock assignments over C ($\pi \models \omega$).

We define a function $delay : \mathcal{TSP} \to \Omega(rel(p))$ as follows:

$$
\begin{array}{ll}
delay(\ell :: t) \equiv delay(t) & delay(\ell :: p_1 \times p_2) \equiv delay(p_1) \wedge delay(p_2) \\
delay(rec\, x.t) \equiv delay(t[rec\, x.t/x]) & delay(t \bigtriangledown p) \equiv delay(t) \wedge delay(p) \\
delay(\ell :: b) \equiv \left\{ \begin{array}{ll} \ell \leq delay(b) & delay(b) \neq \infty \\ true & \text{otherwise} \end{array} \right\} & \\
delay(nil) \equiv \infty & delay(a.t) \equiv \infty \\
delay([m, M]a.t) \equiv M & delay(b_1 + b_2) \equiv min(delay(b_1), delay(b_2));
\end{array}
$$

Intuitively, $delay(p)$ gives the conditions that an assignment to clocks of p must satisfy. These conditions are used to force a timed action to be performed whenever its maximal delay is reached.

Let C be a set of clocks, $\pi : C \to N$ over C and $C_1 \subseteq C$. We define $pro_{C_1}(\pi) : C_1 \to N$, as $\pi(x) = pro_{C_1}(\pi)(x)$, for each $x \in C_1$. For C_1 and C_2 with $C_1 \cap C_2 = \emptyset$, and for $\pi_1 : C_1 \to N$ and $\pi_2 : C_2 \to N$, we define $\pi_1 \cup \pi_2 : C_1 \cup C_2 \to N$, such that $\pi_1 \cup \pi_2(x) = \pi_i(x)$, for $x \in C_i$ and $i \in \{1, 2\}$. Moreover, for $C_1 \subseteq C$ and C_2, such that $C_1 \cap C_2 = \emptyset$, and for $\pi : C \to N$, we denote by $reset(\pi, C_1, C_2) : C_1 \cup C_2 \to N$ the clock assignment $pro_{C_1}(\pi) \cup \{C_2 := 0\}$.

Transitions are labelled by actions obtained as follows.

Definition 2. Let Act be a set of basic actions. We define $\mathcal{S}(Act)$ as the set of *semantic actions* such that, for each $a \in Act$, $\tau \in \{\epsilon, \bar{\epsilon}\}$, $a : \tau \in \mathcal{S}(Act)$.

There are two types of actions: *non-idle* (ϵ) and idle ($\bar{\epsilon}$). A non-idle action represents a real action of the process, while an idle action represents the inability to do anything. This distinction is essential for obtaining maximal parallelism as required by the synchrony of statecharts. The idea is that idle actions represent an environment behaviour, which does not enable any non-idle action of the process.

The operation $\times$ can be trivially extended to semantic actions $\mathcal{S}(Act)$. For $\alpha_i = a_i : \tau_i \in \mathcal{S}(Act)$, for $i \in \{1,2\}$, we define $\alpha_1 \times \alpha_2$ as the set $\{a : \tau \in \mathcal{S}(Act)\}$, such that $a \in a_1 \times a_2$ and $\tau = \epsilon$ if $\tau_i = \epsilon$, for some $i \in \{1,2\}$, while $\tau = \bar{\epsilon}$ otherwise. For $\alpha = a : \tau$, we use the notation $type(\alpha) = \tau$.

Assume $p \in \mathcal{TSP}$ to be a closed process over actions Act and locations Loc. We define the labelled transition system semantics of p as $LTS(p) = (S, s_0, \mathcal{S}(Act), \mapsto)$ where

1. S is the set of timed states (p', π), such that p' is a closed process and π is an assignment over $rel(p)$, such that $\pi \models delay(p)$;
2. $s_0 = (p, \pi_0)$ with $\pi_0(\ell) = 0$, for each $\ell \in rel(p)$;
3. the transition relation $s_1 \overset{\alpha}{\mapsto} s_2$ is obtained by the following rules:

$$\frac{\pi_1 \models m \leq \ell_2 \leq M}{(\ell_1 :: \ell_2 :: [m,M]a.t, \pi_1) \overset{a:\epsilon}{\mapsto} (\ell_1 :: t, \{loc(t) := 1\})}\ non-idle$$

$$\frac{\pi_1 + 1 \models delay(\ell_1 :: \ell_2 :: [m,M]a.t)}{(\ell_1 :: \ell_2 :: [m,M]a.t, \pi_1) \overset{b:\bar{\epsilon}}{\mapsto} (\ell_1 :: \ell_2 :: [m,M]a.t, \pi_1 + 1)}\ idle\ (b \in \#(a) \vee (b \in \#(*) \wedge \#(a) = \emptyset)$$

$$\frac{b \in \#(*)}{(\ell_1 :: \ell_2 :: nil, \pi_1) \overset{b:\bar{\epsilon}}{\mapsto} (\ell_1 :: \ell_2 :: nil, \pi_1 + 1)}\ nil$$

$$\frac{\begin{array}{l}(\ell_1 :: \ell_2 :: b_i, \pi) \overset{\alpha}{\mapsto} (p, \pi') \\ type(\alpha) = \epsilon\end{array}}{(\ell_1 :: \ell_2 :: b_1 + b_2, \pi \overset{\alpha}{\mapsto} (p, \pi')}\ choice_1 \qquad \frac{\begin{array}{l}\{(\ell_1 :: \ell_2 :: b_i, \pi) \overset{\alpha}{\mapsto} (\ell_1 :: \ell_2 :: b_i, \pi_i')\}_{i \in \{1,2\}} \\ type(\alpha) = \bar{\epsilon}\end{array}}{(\ell_1 :: \ell_2 :: b_1 + b_2, \pi) \overset{\alpha}{\mapsto} (\ell_1 :: \ell_2 :: b_1, \pi_1' \cup \pi_2')}\ choice_2$$

$$\frac{\begin{array}{l}(\ell :: t, \pi_1) \overset{\alpha}{\mapsto} (p', \pi_1') \\ type(\alpha) = \epsilon\end{array}}{(\ell :: t \bigtriangledown p, \pi_1 \cup \pi_2) \overset{\alpha}{\mapsto} (p', \pi_1')}\ \bigtriangledown_1 \qquad \frac{\begin{array}{l}(p, \pi_1) \overset{\alpha}{\mapsto} (p', \pi_1') \\ type(\alpha) = \epsilon\ \pi_2 + 1 \models delay(\ell :: t)\end{array}}{(t \bigtriangledown p, \pi_1 \cup \pi_2) \overset{\alpha}{\mapsto} (t \bigtriangledown p', \pi_1' \cup (\pi_2 + 1))}\ \bigtriangledown_2$$

$$\frac{\begin{array}{l}(\ell :: t, \pi_1) \overset{\alpha}{\mapsto} (\ell :: t, \pi_1') \\ (p, \pi_2) \overset{\alpha}{\mapsto} (p, \pi_2') \\ type(\alpha) = \bar{\epsilon}\end{array}}{(\ell :: t \bigtriangledown p, \pi_1 \cup \pi_2) \overset{\alpha}{\mapsto} (\ell :: t \bigtriangledown p, \pi_1' \cup \pi_2')}\ \bigtriangledown_3$$

$$\frac{(\ell :: t[rec\ x.t/x], \pi) \overset{\alpha}{\mapsto} (p, \pi')}{(\ell :: rec\ x.t, \pi) \overset{\alpha}{\mapsto} (p, \pi')}\ rec \qquad \frac{\{(p_i, \pi_i) \overset{\alpha_i}{\mapsto} (p_i', \pi_i')\}_{i \in \{1,2\}}\ \alpha \in \alpha_1 \times \alpha_2}{(\ell :: p_1 \times p_2, \pi_2 \cup \pi_2) \overset{\alpha}{\mapsto} (\ell :: p_1' \times p_2', \pi_2' \cup \pi_2')}\ \times$$

Let us explain the semantics. The main feature is the ability to express maximal reactions, because of the synchrony of statecharts. By rule $non-idle$ action a can be performed nondeterministically whithin its minimal and maximal delay. The resulting assignment for $\ell_1 :: t$ is obtained by setting to one the value of the clocks of t, since the corresponding locations have been entered and time has advanced of one unit. If $\#(a) = \emptyset$, then a does not depend on the environment and can be delayed until M. This is expressed by rule $idle$ by considering a generic behaviour of the environment ($b \in \#(*)$). On the other hand, if $\#(a) \neq \emptyset$, then action a does not depend on temporal constraints by definition and the process can be idle only if the environment has a behaviour which does not enable a. This is obtained by rule $idle$ by taking $a' \in \#(a)$, which is in conflict with a. Note that it must be the case that $\pi_1 + 1 \models delay(\ell_1 :: \ell_2 :: [m,M]a.t)$ so that when the maximal delay is reached action a is forced to happen. Rule $choice_1$ is standard. On the other hand, by rule $choice_2$ an idle action must be an idle action of both processes so that the reaction is maximal. The constructor $\bigtriangledown$ is similar to the one of choice. However, the choice between the two processes is not symmetric as expressed by rules $\bigtriangledown_1$ and $\bigtriangledown_2$. The side condition of $\bigtriangledown_2$ guarantees that no timed action of $\ell :: t$ has reached its maximal delay. Rule $\times$ is the classical rule of parallel synchronous composition and is based on the operation $\times$ over actions, that intuitively realizes the communication between parallel components.

3 The logic RTL_μ

We introduce a discrete version of real-time μ-calculus which is negation-free and has freeze quantification over a set of specification clocks. We adopt a generalization of the tagged fixpoints method for local model checking [24, 23].

Assume a set of specification clocks C, a set of logical variables VAR, a set of propositional symbols $PROP$ and a set of actions $\mathcal{A}$. Moreover, assume a set of clocks C', with $C \cap C' = \emptyset$, a set of states S and a function $rel : S \to 2^{C'}$. Formulas of the real-time μ-calculus RTL_μ are defined as follows

$$A ::= X \mid P \mid \neg P \mid A_1 \vee A_2 \mid A_1 \wedge A_2 \mid \omega \mid < \alpha > A \mid [\alpha]A \mid x.A \mid \mu X\{U\}.A \mid \nu X\{U\}.A.$$

where $\omega \in \Omega(C)$ is a clock constraint, $x \in C$ is a specification clock, $\alpha \in \mathcal{A}$ is an action, $P \in PROP$ is a propositional symbol, $X \in VAR$ is a logical variable and $U = \{(s, \omega) \mid s \in S, \omega \in \Omega(C \cup rel(s))\}$.

The modal operator $< \alpha >$ is the next time, which is equivalent in this framework to next state, while $[\alpha]$ is its dual. The quantifier $x.$ is called *reset* and is inspired by the freeze quantification [2]. Formulas $\mu X\{\emptyset\}.A$ and $\nu X\{\emptyset\}.A$ stand respectively for the classical least and greatest fixpoint. In formulas $\mu X\{U\}.A$ and $\nu X\{U\}.A$, U is called the tag and represents a set of timed states over clocks $C' \cup C$. Note that the specification clocks of formulas and the clocks corresponding in timed states are required to be distinct for avoiding the reset of clock corresponding to locations.

Formulas are interpreted over a model $\mathcal{M} = ((TS, ts_o, \mathcal{A}, \mapsto), \gamma)$, where $\gamma : PROP \to 2^{TS}$ is the valuation function of propositional symbols and $(TS, ts_o, \mathcal{A}, \mapsto)$ is a labelled transition system, such that

- TS is a set of timed states (s, π), where $s \in S$ and $\pi : C \cup C' \to N \cup \{und\}$ is a partial assignment for the specification clocks $C' \cup C$, such that $\pi = \pi_1 \cup \pi_2$, where $\pi_1 : C \to N \cup \{und\}$ and $\pi_2 : C' \to N$;
- $ts_0 = (s_0, \pi_0)$, where $\pi_0(x) = und$, for each $x \in C$;
- $\mapsto \subseteq TS \times \mathcal{A} \times TS$ is the next state relation such that, for each $(s_1, \pi_1) \overset{\alpha}{\mapsto} (s_2, \pi_2)$, $\pi_2(x) = \pi_1(x) + 1$, for $x \in C$ $(und + 1 \equiv und)$.

The semantics of formulas with respect to a model $\mathcal{M} = ((TS, ts_o, \mathcal{A}, \mapsto), \gamma)$ and to a valuation $\rho : VAR \to 2^{TS}$ is defined as follows:

$$\begin{array}{l}
\| P \|_{\mathcal{M},\rho} = \gamma(P) \\
\| \neg P \|_{\mathcal{M},\rho} = TS \setminus \gamma(P) \\
\| \omega \|_{\mathcal{M},\rho} = \{(s,\pi) \in TS \mid \pi \models \omega\} \\
\| x.A \|_{\mathcal{M},\rho} = \{(s,\pi) \in TS \mid (s, \pi[x := 0]) \in \| A \|_{\mathcal{M},\rho}\} \\
\| < \alpha > A \|_{\mathcal{M},\rho} = \{(s,\pi) \in TS \mid (s,\pi) \overset{\alpha}{\mapsto} (s',\pi'), (s',\pi') \in \| A \|_{\mathcal{M},\rho}\} \\
\| [\alpha]A \|_{\mathcal{M},\rho} = \{(s,\pi) \in TS \mid \forall (s,\pi) \overset{\alpha}{\mapsto} (s',\pi'), (s',\pi') \in \| A \|_{\mathcal{M},\rho}\} \\
\| \mu X\{U\}.A \|_{\mathcal{M},\rho} = \mu V.(\| A \|_{\mathcal{M},\rho[V/X]} \setminus \| U \|_{\mathcal{M},\rho[V/X]}) \\
\| \nu X\{U\}.A \|_{\mathcal{M},\rho} = \nu V.(\| A \|_{\mathcal{M},\rho[V/X]} \cup \| U \|_{\mathcal{M},\rho[V/X]})
\end{array}$$

where $\| U \|_{\mathcal{M},\rho} = \{(s,\pi) \in TS \mid (s,\omega) \in U, \pi \models \omega\}$.

We use the notation $(s,\pi) \models_{\mathcal{M}} A$ for $(s,\pi) \in \| A \|_{\mathcal{M},\rho}$ for each valuation ρ. For a process p and an assignment π over $rel(p)$, we say that $(p,\pi) \models A$ iff $(p,\pi) \models_{(LTS(p),\gamma)} A$, where γ is the valuation $\gamma : Loc \to \mathcal{CTSP}$, such that $p \in \gamma(P)$ iff $P \in loc(p)$. Note that A must be a closed formula defined over propositional symbols Loc, actions $\mathcal{S}(Act)$, specification clocks C, with $C \cap Loc = \emptyset$, states $\mathcal{CTSP}$ and rel as defined in section 2. Moreover, for $\omega \in \Omega(C \cup rel(p))$, we say $p, \omega \models A$ iff, for each π with $\pi \models \omega$, $(p,\pi) \models A$.

It is important to stress that the logic RTL_μ subsumes the discrete real-time extension of CTL $RTCTL$ [8]. For instance the time-bounded property $E\Diamond_{\leq 3}A$ is expressed by the formula $x.\mu X.((A \wedge x \leq 3) \vee < - > X)$, while $A\Box_{\leq 3}A$ is expressed by the formula $x.\nu X.(((A \wedge x \leq 3) \vee (x > 3)) \wedge [-]X)$.

4 The proof system

We introduce the proof system. We consider judgments such as $p, \omega \vdash A$, where $p \in \mathcal{CTSP}$, $\omega \in \Omega(rel(p)) \cup C)$ and A is a closed RTL_μ formula. We require in addition that for each $x + m\mathcal{R}y + m \in \omega$, $x, y \in C$. The obvious interpretation of $p, \omega \vdash A$ is that (p, π) satisfies A for each assignment $\pi \models \omega$. There are two classes of rules: logical rules that do not involve process operators and rules that work on the structure of the process for the next modalities. The proof system is developed in a goal directed fashion (top down).

We need some preliminary definitions. Assume $\omega \in \Omega(C)$. We denote by $\omega + \delta$ $(\omega - \delta)$ for $\delta \in N$, the clock constraint $\omega[x+\delta/x]$ $(\omega[x-\delta/x])$, for each clock $x \in C$. For $C_1 \subseteq C$, we denote by $pro_{C_1}(\omega)$ the first order formula $\exists x_1, \ldots, x_n\ (\omega)$, where $C \setminus C_1 = \{x_1, \ldots x_n\}$. Let $\omega \in \Omega(C)$ and $C_1 \subseteq C$, $C_2 \cap C_1 = \emptyset$. We denote by $reset(\omega, C_1, C_2) \equiv pro_{C_1}(\omega) \wedge \bigwedge_{x \in C_2} x = 0$. It can be shown that $reset(\omega, C_1, C_2) \in \Omega(C_1 \cup C_2)$. Intuitively, $pro_{C_1}(\omega)$ and $reset(\omega, C_1, C_2)$ are the counterparts of the corresponding operations over clock assignments. For $\omega \in \Omega(C)$, we use the notation $\forall(\omega)$ for $\forall x_1, \ldots, \forall x_n(\omega)$, where $C = \{x_1, \ldots, x_n\}$.

4.1 Logical Rules

$$\frac{p, \omega \vdash P}{\quad}\, prop_1\ P \in loc(p) \qquad \frac{p, \omega \vdash \neg P}{\quad}\, prop_2\ P \notin loc(p)$$

$$\frac{p, \omega \vdash A_1 \wedge A_2}{p, \omega \vdash A_1 \quad p, \omega \vdash A_2}\, \wedge \qquad \frac{p, \omega \vdash A_1 \vee A_2}{p, \omega \vdash A_i}\, \vee$$

$$\frac{p, \omega_1 \vdash A}{p, \omega_2 \vdash A}\, thin \models \forall(\omega_1 \supset \omega_2) \qquad \frac{p, \omega_1 \vee \omega_2 \vdash A}{p, \omega_1 \vdash A \quad p, \omega_1 \vdash A}\, disj$$

$$\frac{p, \omega_1 \vdash \omega_2}{\quad}\, taut \models \forall(\omega_1 \supset \omega_2) \qquad \frac{p, \omega \vdash A}{\quad}\, empty \models \forall(\omega \wedge delay(p) \supset false)$$

$$\frac{p, \omega \vdash x.A}{p, reset(\omega, (rel(p) \cup C) \setminus \{x\}, x) \vdash A}\, reset \qquad \frac{p, \omega \vdash \nu X\{U\}.A}{\quad}\, \nu_1\ \begin{array}{l} \{(p, \omega_i) \subseteq U\}_{i \in \{1,n\}} \\ \models \forall(\omega \supset \bigvee_{i \in \{1,n\}} \omega_i) \end{array}$$

$$\frac{p, \omega \vdash \nu X\{U\}.A}{p, \omega \vdash A[\nu X\{U \cup \{(p, \omega)\}\}.A/X]}\, \nu_2 \qquad \frac{p, \omega \vdash \mu X\{U\}.A}{p, \omega \vdash A[\mu X\{U \cup \{(p, \omega)\}\}.A/X]}\, \mu\ \begin{array}{l} \{(p, \omega_i) \subseteq U\}_{i \in \{1,n\}} \\ \forall i \in \{1, n\} \models \forall(\omega \wedge \omega_i \supset false) \end{array}$$

Rule $reset$ is based on the semantics of quantifier $x.$ by exploiting the fact that $\pi \models \omega$ iff $\pi[x := 0] \models reset(\omega, (rel(p) \cup C) \setminus \{x\}, x)$. Rules ν_1, ν_2 and μ are obtained by generalizing to sets of timed states the rules for computing fixpoints locally [24].

4.2 Rules for modal operators

We present the rules for establishing $< \alpha > A$ and $[\alpha]A$ in a compositional way. Let us explain the main ideas. Consider a process $p = \ell_1 :: \ell_2 :: [1,4]a.\ell_3 :: nil$ and a clock constraint $\omega \equiv \ell_2 = 3$. We have $p, \omega \models < a : \epsilon > \ell_3$, where $< a : \epsilon > \ell_3$ express the ability to perform the action $a : \epsilon$ and to reach a configuration containing location ℓ_3. Since p is not further decomposable we can immediately deduce from its syntax the transitions of its LTS. Therefore, we have the following proof.

$$\cfrac{p, \omega \vdash < a : \epsilon > \ell_3}{\cfrac{\ell_1 :: \ell_3 :: nil, \ell_3 = 1 \vdash \ell_3}{\quad}\, \ell_3 \in loc(\ell_1 :: \ell_3 :: nil)} \models (\omega \supset 1 \leq \ell_2 \leq 4)$$

The side condition $\models (\omega \supset 1 \leq \ell_2 \leq 4)$ ensures that rule $non-idle$ is indeed applicable. Note that we consider the clock constraint $\ell_3 = 1$, since the clock assignment corresponding to $\ell_1 :: \ell_3 :: nil$ by rule $non-idle$ is $\{loc(\ell_3 :: nil) := 1\}$.

Consider a process $\ell :: t \bigtriangledown p$, where $t = \ell_1 :: [1,3]a.\ell_2 :: nil$ and $p = \ell_3 :: \ell_4 :: [0,\infty]b.\ell_5 :: nil$. We have $\ell :: t \bigtriangledown p, \omega \models < a : \epsilon > true$, for $\omega \equiv 1 \leq \ell_1 \leq 2$, since by rule $\bigtriangledown_1$ $(\ell :: t \bigtriangledown p, \pi) \overset{a:\epsilon}{\mapsto} (\ell :: \ell_2 :: nil, \ell_2 = 1)$, for each $\pi \models \omega$. Rule $\bigtriangledown_1$ suggests how to decompose the structure of the process. We obtain the following proof:

$$\cfrac{\ell :: t \bigtriangledown p, \omega \vdash < a : \epsilon > true}{\cfrac{\ell :: t, \omega \vdash < a : \epsilon > true}{\ldots}} \; type(a:\epsilon) = \epsilon$$

Note that the side condition $type(a : \epsilon) = \epsilon$ is necessary to obtain a sound rule, namely for ensuring that $\bigtriangledown_1$ is applicable.

We have $\ell :: t \bigtriangledown p, \omega \models < b : \epsilon > true$, since by rule $\bigtriangledown_2$ $(\ell :: t \bigtriangledown p, \pi) \overset{b:\epsilon}{\mapsto} (\ell :: t \bigtriangledown p', \ell_5 = 1 \wedge 2 \leq \ell_1 \leq 3)$ with $p' = \ell_4 :: \ell_5 :: nil$, for each $\pi \models \omega$. However, it would not be correct to deduce $\ell :: t \bigtriangledown p, \omega \vdash < b : \epsilon > true$ from $p, \omega \vdash < b : \epsilon > true$ as before.

In order to achieve *compositionality* we adapt a technique proposed by [5], that permits to avoid the construction of the transition from $\ell :: t \bigtriangledown p$. The idea is that of using *extended assertions* such as $(A)_{(p,\omega)\bigtriangledown}$, with the semantics $\parallel (A)_{(p,\omega)\bigtriangledown} \parallel = \{(p', \pi') \mid (p \bigtriangledown p', \pi' \cup \pi \models A$, for each $\pi \models \omega\}$. Whit this understanding it can be obtained a proof as the following:

$$\cfrac{\ell :: t \bigtriangledown p, \omega \vdash < b : \epsilon > true}{\cfrac{p, \omega_1 \vdash < b : \epsilon > (true)_{(\ell::t,\omega_2)\bigtriangledown}}{\ldots}} \; \models \forall(\omega_2 \supset delay(\ell :: t))$$

where $\omega_1 \equiv true \equiv pro_{rel(p)}(\omega)$, $\omega_2 \equiv 2 \leq \ell_1 \leq 3 \equiv pro_{rel(\ell::t)}(\omega) - 1$. The constraint ω_2 is obtained by noting that $\pi \models pro_{rel(\ell::t)}(\omega)$ iff $\pi + 1 \models \omega_2$ as required in rule $\bigtriangledown_2$. The side condition $\models \forall(\omega_2 \supset delay(\ell :: t))$ guarantees that $\bigtriangledown_2$ is indeed applicable.

Now a proof of $p, \omega_1 \vdash < b : \epsilon > (true)_{(\ell::t,\omega_2)\bigtriangledown}$ can be obtained as follows.

$$\cfrac{p, \omega_1 \vdash < b : \epsilon > (true)_{(\ell::t,\omega_2)\bigtriangledown}}{\cfrac{p', \omega_1' \vdash (true)_{(\ell::t,\omega_2)\bigtriangledown}}{\cfrac{\ell :: t \bigtriangledown p', \omega_2 \wedge \omega_1' \vdash true}{}} \; shift}$$

where $\omega_1' \equiv \ell_5 = 1$.

A special rule called *shift rule* is applied for eliminating the operators from formulas. The resulting proof is compositional, namely, if we replace process t by t', this part of the proof is not affected except for the test concerning the delay. Similar techniques are applied for the other constructors.

We include in the syntax the following *extended assertions*:

$$(A)_{(\ell::t,\omega)\bigtriangledown} \mid (A)_{\ell\times(p,\omega)} \mid (A)_{\times+b} \mid (A)_{\times(\ell::t,\omega)\bigtriangledown} \mid (A)_{\times\bigtriangledown(p,\omega)} \mid (A)_{\times}$$

The semantics of these assertions is defined as follows:

$$\begin{array}{l}
\parallel (A)_{(\ell::t,\omega)\bigtriangledown} \parallel = \{(p,\pi) \mid (\ell :: t \bigtriangledown p, \pi \cup \pi') \models A, \forall \pi' \models \omega\} \\
\parallel (A)_{\ell\times(p,\omega)} \parallel = \{(p',\pi) \mid (\ell :: p' \times p, \pi \cup \pi') \models A, \forall \pi' \models \omega\} \\
\parallel (A)_{\times+b} \parallel = \{(\ell :: p_1 \times \ell_1 :: \ell_2 :: b', \pi) \mid (\ell :: p_1 \times \ell_1 :: \ell_2 :: b' + b, \pi) \models A\} \\
\parallel (A)_{\times(\ell::t,\omega)\bigtriangledown} \parallel = \{(\ell_1 :: p_1 \times p_2, \pi) \mid (\ell_1 :: p_1 \times \ell :: t \bigtriangledown p, \pi \cup \pi') \models A, \forall \pi' \models \omega\} \\
\parallel (A)_{\times\bigtriangledown(p,\omega)} \parallel = \{(\ell_1 :: p_1 \times \ell :: t, \pi) \mid (\ell_1 :: p_1 \times \ell :: t \bigtriangledown p, \pi \cup \pi') \models A, \forall \pi' \models \omega\} \\
\parallel (A)_{\times} \parallel = \{(\ell_1 :: \ell_2 :: (p_1 \times p_2) \times p_3, \pi) \mid (\ell_1 :: p_1 \times \ell_2 :: (p_2 \times p_3)) \models A\}.
\end{array}$$

4.3 Shift Rules

$$\frac{p, \omega_1 \vdash (A)_{(\ell::t,\omega_2)\bigtriangledown}}{\ell :: t \bigtriangledown p, \omega_1 \wedge \omega_2 \vdash A} \qquad \frac{p_1, \omega_1 \vdash (A)_{\ell\times(p_2,\omega_2)}}{\ell :: p_1 \times p_2, \omega_1 \wedge \omega_2 \vdash A}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1, \omega_1 \models (A)_{\times+b}}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b, \omega_1 \models A} \qquad \frac{\ell :: p_1 \times p_2, \omega_1 \vdash (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}}{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_2, \omega_1 \wedge \omega_2 \vdash A}$$

$$\frac{\ell :: p_2 \times \ell_1 :: t, \omega_1 \vdash (A)_{\times\bigtriangledown(p_1,\omega_2)}}{\ell :: p_2 \times \ell_1 :: t \bigtriangledown p_1, \omega_1 \wedge \omega_2 \vdash A} \qquad \frac{\ell :: \ell_1 :: (p_1 \times p_2) \times p_3, \omega_2 \vdash (A)_\times}{\ell :: p_1 \times \ell_1 :: (p_2 \times p_3), \omega_1 \wedge \omega_2 \vdash A}$$

These rules are trivially based on the previously defined semantics for extended assertions.

4.4 Rules for *nil*

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil, \omega_1 - 1 \vdash A} <> nil \; a \in \#(*) \qquad \frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \epsilon]A}{} []nil_1$$

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \bar{\epsilon}]A}{\ell :: \ell_1 :: nil, \omega_1 - 1 \vdash A} []nil_2 \; a \in \#(*) \qquad \frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \bar{\epsilon}]A}{} []nil_3 \; a \notin \#(*)$$

4.5 Rules for timed prefixing

Let $\omega_2 \equiv reset(\omega_1, C, loc(t)) - 1$ and $\omega_3 \equiv reset(\omega_1 \wedge m \leq \ell_1 \leq M, C, loc(t)) - 1$.

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash A} \tau <> \epsilon \qquad \models \forall(\omega_1 \supset m \leq \ell_1 \leq M)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash < a_2 : \bar{\epsilon} > A}{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 - 1 \vdash A} \tau <> \bar{\epsilon} \qquad \begin{array}{l}(a_2 \in \#(*) \wedge \#(a_1) = \emptyset) \vee (a_2 \in \#(a_1)) \\ \models \forall(\omega_1 - 1 \supset \ell_1 \leq M)\end{array}$$

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash [a : \epsilon]A}{\ell :: t, \omega_3 \vdash A} \tau[]\epsilon_1 \qquad \not\models \forall(\omega_1 \wedge m \leq \ell_1 \leq M \supset false)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 - 1 \vdash A} \tau[]\bar{\epsilon}_1 \qquad (a_2 \in \#(*) \wedge \#(a_1) = \emptyset) \vee a_2 \in \#(a_1)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \epsilon]A}{} \tau[]\epsilon_2 \qquad a_1 \neq a_2$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{} \tau[]\bar{\epsilon}_2 \qquad (a_2 \notin \#(*) \wedge \#(a_1) = \emptyset) \vee ((a_2 \notin \#(a_1) \wedge \#(a_1) \neq \emptyset)$$

$$x\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_1 : \epsilon]A}{} \tau[]\epsilon_3 \qquad \models \forall(\omega_1 \wedge m \leq \ell_1 \leq M \supset false)$$

These rules are based on rules $non-idle$ and $idle$ of the operational semantics by exploiting the fact that $\pi \models \omega_1$ iff $reset(\pi, C, rel(t)) + 1 \models \omega_2$ and analogously for ω_3.

4.6 Rules for choice

There are two sets of rules: rules applicable when the type of the action is non-idle and rules applicable when the type of the action is idle in order to distinguish whether either $choice_1$ or $choice_2$ is applicable.

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: \ell_1 :: b_i, \omega_1 \vdash < \alpha > A} + <> \epsilon \quad \frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: \ell_1 :: b_i, \omega_1 \vdash [\alpha]A\}_{i \in \{1,2\}}} + []\epsilon$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\{\ell :: \ell_1 :: b_i, \omega_1 \vdash < \alpha > true\}_{i \in \{1,2\}} \ \ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \vdash A} + <> \bar{\epsilon}$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \vdash A} + []\bar{\epsilon}_1 \quad \frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: \ell_1 :: b_i, \omega_1 \vdash [\alpha]false} + []\bar{\epsilon}_2$$

4.7 Rules for refinement

Rules for refinement are similar to the ones for choice. However, in this case $rel(\ell :: t \bigtriangledown p) = rel(\ell :: t) \cup rel(p)$ and $rel(p) \cap rel(\ell :: t) = \emptyset$, by definition of well-formed process. Therefore, we consider the projections with respect to $rel(p)$ and $rel(\ell :: t)$. Let $\omega_2 \equiv pro_{rel(\ell::t)}(\omega_1) - 1$, $\omega_p \equiv pro_{C \cup rel(p)}(\omega_1)$ and $\omega_t \equiv pro_{C \cup rel(\ell::t)}(\omega_1)$.

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: t, \omega_t \vdash < \alpha > A} \bigtriangledown <> \epsilon_1 \qquad \frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t, \omega_t \vdash [\alpha]A \ p, \omega_p \vdash [\alpha](A)_{(\ell::t,\omega_2)\bigtriangledown}} \bigtriangledown[]\epsilon$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{p, \omega_p \vdash < \alpha > (A)_{(\ell::t,\omega_2)\bigtriangledown}} \bigtriangledown <> \epsilon_2 \ \models \forall(\omega_2 \supset delay(\ell :: t))$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: t, \omega_t \vdash < \alpha > true \ p, \omega_p \vdash < \alpha > true \ \ell :: t \bigtriangledown p, \omega_1 - 1 \vdash A} \bigtriangledown <> \bar{\epsilon}$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t \bigtriangledown p, \omega_1 - 1 \vdash A} \bigtriangledown[]\bar{\epsilon}_1 \ \frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t, \omega_t \vdash [\alpha]false} \bigtriangledown[]\bar{\epsilon}_2 7 \frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{p, \omega_p \vdash [\alpha]false} \bigtriangledown[]\bar{\epsilon}_3$$

Rules for recursion

$$\frac{\ell :: rec\ x.t, \omega_1 \vdash \sigma A \ \sigma \in \{< \alpha >, [\alpha]\}}{\ell :: t[rec\ x.t/x], \omega_1 \vdash \sigma A} rec$$

4.8 Rules for parallel composition

The case of parallel composition is the most complex. We would like to give a set of rules for proving $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$ from $\{\ell :: p_1 \times p_{2,j}, \omega_j \vdash < \alpha_j > A\}_{j\in\{1,n\}}$, where $p_{2,j}$ are components of p_2 and ω_j are obtained from ω by projection depending on the structure of p_2.

Suppose that $p_2 = \ell_1 :: \ell_2 :: [m, M]a.t$ and that $(\ell :: p_1 \times p_2, \pi_1 \cup \pi_2) \overset{\alpha}{\mapsto} (\ell :: p'_1 \times p'_2, \pi'_1 \cup \pi'_2)$, where $p'_2 = \ell_1 :: t$ and $\pi'_2 = \{loc(t) := 1\}$, for each $\pi_1 \cup \pi_2 \models \omega$. By definition of $\times$, there exists $(p_1, \pi_1) \overset{\alpha_1}{\mapsto} (p'_1, \pi'_1)$ with $\alpha \in \alpha_1 \times (a : \epsilon)$. Following this construction, we can derive $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$ from $p_1, \omega_1 \vdash < \alpha_1 > (A)_{\ell\times(p'_2,\omega_2)}$, where $\omega_1 \equiv pro_{C\cup rel(p_1)}(\omega)$ and $\omega_2 \equiv loc(t) = 1$ under the condition that $\models \forall(\omega \supset m \leq \ell_1 \leq M)$. However, this method is not applicable in general. Suppose that $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$ and that one wants to prove $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$, where $type(\alpha) = \epsilon$. One might think of obtaining a proof from $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$ for some $i \in \{1, 2\}$. However, this kind of deduction is not correct in general. Actually, there exists $(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \pi) \overset{\alpha}{\mapsto} (\ell :: p'_1 \times p'_2, \pi')$, iff by rule $\times$, $(p_1, \pi_1) \overset{\alpha_1}{\mapsto} (p'_1, \pi'_1)$ and $(\ell_1 :: \ell_2 :: b_1 + b_2, \pi_2) \overset{\alpha_2}{\mapsto} (p'_2, \pi'_2)$ such that $\alpha \in \alpha_1 \times \alpha_2$ and $\pi = \pi_1 \cup \pi_2$, $\pi' = \pi'_1 \cup \pi'_2$. Now, if $type(\alpha_2) = \epsilon$, by rule $choice_1$, $(\ell_1 :: \ell_2 :: b_i, \pi_2) \overset{\alpha_2}{\mapsto} (p'_2, \pi'_2)$, for some $i \in \{1, 2\}$, and by rule $\times$, $(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \pi) \overset{\alpha}{\mapsto} (\ell :: p'_1 \times p'_2, \pi')$. In this case $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \models < \alpha > A$ so that the suggested deduction is correct. On the other hand, if $type(\alpha_2) = \bar{\epsilon}$ then rule $choice_1$ is not applicable. In conclusion, it is necessary to know the type of α_2. Since $type(\alpha_2)$ cannot be deduced from $type(\alpha)$, we must adopt formulas with modal operators over *extended actions*.

Definition 3. Let Act be a set of basic actions. We define $\mathcal{E}(Act)$ as the set of *extended actions* $a : \tau$, where $a \in Act$ and τ is an *extended* type, such that: $\epsilon, \bar{\epsilon}$ are basic extended types and, if τ_1 is an extended type and τ_2 is a basic extended type, then $\tau_1 \times \tau_2$ is an extended type. For $a : \tau \in \mathcal{E}(Act)$, we define $type^*(a : \tau) = \tau$, if τ is basic, and $type^*(a : \tau_1 \times \tau_2) = \epsilon$ if $type^*(a : \tau_i) = \epsilon$, for some $i \in \{1, 2\}$, and $type^*(a : \tau_1 \times \tau_2) = \bar{\epsilon}$, otherwise.

For $\alpha \in \mathcal{E}(Act)$, we say that $(p, \pi) \overset{\alpha'}{\mapsto} (p', \pi')$ is *compatible* with α iff

1. $\alpha = a : \tau$, $\alpha' = a : \tau'$ and $type(\alpha') = type^*(\alpha)$;
2. If $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$, then $(p, \pi) \overset{\alpha'}{\mapsto} (p', \pi')$ has been been obtained by rule $\times$ from $(p_i, \pi_i) \overset{\alpha'_i}{\mapsto} (p'_i, \pi'_i)$, such that $(p_i, \pi_i) \overset{\alpha'_i}{\mapsto} (p'_i, \pi'_i)$ is compatible with $\alpha_i = a_i : \tau_i$ and $\alpha'_i = a_i : \tau'_i$.

   We can now define the semantics of formulas with extended actions.

$$\begin{array}{ll} \| < \alpha > A \| = & \left\{ (p,\pi) \mid \exists (p,\pi) \overset{\alpha'}{\mapsto} (p',\pi'), (p',\pi') \in \| A \|, (p,\pi) \overset{\alpha'}{\mapsto} (p',\pi') \text{ is compatible with } \alpha \right\} \\ \| [\alpha] A \| = & \left\{ \{(p,\pi) \mid \forall (p,\pi) \overset{\alpha}{\mapsto} (p',\pi') \text{ that is compatible with } \alpha, (p',\pi') \in \| A \| \right\} \end{array}$$

Intuitively, $(p, \pi) \models < \alpha > A$ with $\alpha \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$ iff there exists $(p, \pi) \overset{\alpha'}{\mapsto} (p', \pi')$, where $(p', \pi') \models A$, such that $\alpha = a : \tau$, $\alpha' = a : \tau'$ and this transition is constructed consistently with τ. For instance, if $\tau = \epsilon \times \bar{\epsilon}$, then it must be obtained by rule $\times$ applied to premises $\{(p_i, \pi'_i) \overset{\alpha'_i}{\mapsto} (p'_i, \pi'_i)\}_{i\in\{1,2\}}$, where $type(\alpha'_1) = \epsilon$ and $type(\alpha'_2) = \bar{\epsilon}$.

We extend the operation $\times$ to extended actions. For $a_1 : \tau_1 \in \mathcal{E}(Act)$, $a_2 : \tau_2 \in \mathcal{S}(Act)$, $(a : \tau_1) \otimes (a : \tau_2) = \{a : \tau_1 \times \tau_2 \mid a \in a_1 \times a_2\} \subseteq \mathcal{E}(Act)$. For $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act)$, we define $proj_i(\alpha) = a : \tau_i$.

Expansion rules Extended actions are introduced by the following rules.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times p_2, \omega_1 \vdash < \beta > A}\ \mathcal{E} <> \ \beta \in \mathcal{E}(\alpha) \qquad \frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha] A}{\{\ell :: p_1 \times p_2, \omega_1 \vdash [\beta_i] A\}_{i\in\{1,n\}}}\ \mathcal{E}[] \ \{\beta_i\}_{i\in\{1,n\}} = \mathcal{E}(\alpha)$$

where for $a : \tau \in \mathcal{S}(Act)$, $\mathcal{E}(a : \tau) = \{a : \tau' \mid \tau \in \mathcal{E}(\tau)\}$, with $\mathcal{E}(\tau) = \{\epsilon \times \epsilon, \bar{\epsilon} \times \epsilon, \epsilon \times \bar{\epsilon}\}$, for $\tau = \epsilon$ and $\mathcal{E}(\tau) = \{\bar{\epsilon} \times \bar{\epsilon}\}$ otherwise. In other words $\mathcal{E}(\tau)$ realizes the inverse of $type^*$ and produces extended types consistent with the basic type τ.

Rules for *nil* Let $p_2 = \ell_1 :: \ell_2 :: nil$ and $\omega_2 \equiv pro_{\{\ell_2\}}(\omega_1) - 1$, $\omega_{p_1} \equiv pro_{C \cup rel(p_1)}(\omega_1)$. Moreover, let $\{\beta_i\}_{i\in\{1,k\}} = \{\beta \mid \alpha \in \beta \otimes (b : \bar{\epsilon}), b \in \#(*)\}$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, \omega_{p_1} \vdash < \beta > (A)_{\ell\times(p_2,\omega_2)}} \times <> nil \;\; \begin{array}{l}\alpha \in \beta \otimes (b : \bar{\epsilon})\\ b \in \#(*)\end{array} \quad \frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\{p_1, \omega_{p_1} \vdash [\beta_i](A)_{\ell\times(p_2,\omega_2)}\}_{i\in\{1,k\}}} \times [] nil$$

Rules for prefixing Let $p_2 = \ell_1 :: \ell_2 :: [m, M]a.t$, $\omega_2 \equiv rel(t) = 1$, $\omega_3 \equiv pro_{rel(p_2)}(\omega_1) - 1$ and $\omega_{p_1} \equiv pro_{C\cup loc(p_1)}(\omega_1)$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, \omega_{p_1} \vdash < \beta > (A)_{\ell\times(\ell_1::t,\omega_2)}} \tau\times <>_1 \;\; \begin{array}{l}\alpha \in \beta \otimes (a : \epsilon)\\ \models \forall(\omega_1 \supset m \leq \ell_2 \leq M)\end{array}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, \omega_{p_1} \vdash < \beta > (A)_{\ell\times(p_2,\omega_3)}} \tau\times <>_2 \;\; \begin{array}{l}\alpha \in \beta \times (b : \bar{\epsilon})\\ (b \in \#(*) \wedge \#(a) = \emptyset) \vee b \in \#(a)\\ \models \forall(\omega_3 \supset \ell_2 \leq M)\end{array}$$

Let $\{\alpha_i\}_{i\in\{1,n\}} = \{\alpha_i \mid \alpha \in \alpha_i \otimes (a : \epsilon)\}$ and $\{\beta_i\}_{i\in\{1,k\}} = \{\beta_i \mid \alpha \in \beta_i \otimes (b : \bar{\epsilon}), (b \in \#(*) \wedge \#(a) = \emptyset) \vee b \in \#(a)\}$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\begin{array}{l}\{p_1, \omega_{p_1} \vdash [\alpha_i](A)_{\ell\times(\ell_1::t,\omega_2)}\}_{i\in\{1,n\}}\\ \{p_1, \omega_{p_1} \vdash [\beta_i](A)_{\ell\times(p_2,\omega_3)}\}_{i\in\{1,k\}}\end{array}} \tau \times []_1 \;\; \not\models \forall(\omega_1 \wedge m \leq \ell \leq M \supset false)$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\{p_1, \omega_{p_1} \vdash [\beta_i](A)_{\ell\times(p_2,\omega_3)}\}_{i\in\{1,k\}}} \tau \times []_2 \quad \models \forall(\omega_1 \wedge m \leq \ell \leq M \supset false)$$

Rules for choice

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash < \alpha > A} \times <> +\epsilon \; \frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha]A\}_{i\in\{1,2\}}} \times [] + \epsilon$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash < \alpha > (A)_{\times + b_j}\}_{i,j\in\{1,2\}, i\neq j}} \times <> +\bar{\epsilon}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha](A)_{\times + b_j}\}_{i,j\in\{1,2\}, i\neq j}} \times [] + \bar{\epsilon}_1 \; \frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha]false} \times [] + \bar{\epsilon}_2$$

Rules for refinement Let $\omega_p \equiv pro_{C\cup(rel(\ell::p_1\times p))}(\omega_1)$, $\omega_t \equiv pro_{C\cup rel(\ell::p_1\times\ell_1::t)}(\omega_1)$, $\omega_2 \equiv pro_{rel(\ell_1::t)}(\omega_1)$ -1 and $\omega_3 \equiv pro_{rel(p)}(\omega_1) - 1$.

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash < \alpha > A} \times <> \bigtriangledown\epsilon_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times p, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}} \times <> \bigtriangledown\epsilon_2 \qquad \models \forall(\omega_2 \supset delay(\ell_1 :: t))$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times p, \omega_p \vdash [\alpha](A)_{\times(\ell_1::t,\omega_2)\bigtriangledown} \;\; \ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha]A} \times [] \bigtriangledown \epsilon_2$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\begin{array}{l}\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash < \alpha > (A)_{\times \bigtriangledown (p,\omega_3)} \\ \ell :: p_1 \times p, \omega_p \vdash < \alpha > (A)_{\times (\ell_1 :: t, \omega_2)\bigtriangledown}\end{array}} \times <> \bigtriangledown \bar{\epsilon} \quad \frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\begin{array}{l}\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha](A)_{\times \bigtriangledown (p,\omega_3)} \\ \ell :: p_1 \times p, \omega_p \vdash [\alpha](A)_{\times (\ell_1 :: t, \omega_2)\bigtriangledown}\end{array}} \times [] \bigtriangledown \bar{\epsilon}_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha]false} \times [] \bigtriangledown \bar{\epsilon}_2 \qquad \frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times p, \omega_p \vdash [\alpha]false} \times [] \bigtriangledown \bar{\epsilon}_3$$

Rules for recursion

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t, \omega_1 \vdash \sigma A \ \sigma \in \{< \alpha >, [\alpha]\}}{\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x], \omega_1 \vdash \sigma A} \times rec$$

Rules for parallel composition

$$\frac{\ell :: p_1 \times \ell_2 :: (p_3 \times p_4), \omega_1 \vdash < \alpha > A}{\ell :: \ell_2 :: (p_1 \times p_3) \times p_4, \omega_1 \vdash < \beta > (A)_\times} \times\times <> \ \beta \in left(\alpha)$$

$$\frac{\ell :: p_1 \times \ell_2 :: (p_3 \times p_4), \omega_1 \vdash [\alpha]A}{\{\ell :: \ell_1 :: (p_1 \times p_3) \times p_4, \omega_1 \vdash [\beta_i](A)_\times\}_{i \in \{1,n\}}} \times \times [] \ \{\beta_1, \ldots \beta_n\} = left(\alpha)$$

where, for $a :: \tau_1 \times \tau_2 \in \mathcal{E}(Act)$, $left(a : \tau_1 \times \tau_2) = \{a : \tau\}$, such that $\tau = (\tau_1 \times \tau_{2,1}) \times \tau_{2,2}$, for $\tau_{2,1} \times \tau_{2,2} \in \mathcal{E}(\tau_2)$. The idea is that it is sufficient to associate on the left by the associative property of parallel composition.

5 Soundness and Completeness

The proof system is sound in general, while it is complete for the class of *regular* processes (finite-state). A closed process p is *regular* iff it has guarded recursion and for each variable x, x does not occur free in a subprocess $\ell :: p_1 \times p_2$ and in process p_1, such that $\ell :: t \bigtriangledown p_1$ is a subprocess of p.

We exploit the following properties concerning the correspondence between assignments and constraints.

Proposition 4. *Let C be a set of clocks and $C_1 \subseteq C$, $C_1 \cap C_2 = \emptyset$. For each assignment π over C and for each $\omega \in \Omega(C)$:*

1. *if $\pi \models \omega$ then $pro_{C_1}(\pi) \models pro_{C_1}(\omega)$. Moreover, if ω is satisfiable, then if $\pi' \models pro_{C_1}(\omega)$, there exists $\pi \models \omega$ and $\pi' = pro_{C_1}(\pi)$.*
2. *if $\pi \models \omega$, then $reset(\pi, C_1, C_2) \models reset(\omega, C_1 , C_2)$. Moreover, if ω is satisfiable and $\pi' \models reset(\omega, C_1, C_2)$, then there exists $\pi \models \omega$ and $reset(\pi, C_1, C_2,) = \pi'$.*
3. *$\pi \models \omega + \delta$ iff $\pi + \delta \models \omega$ and $\pi \models \omega - \delta$ iff $\pi - \delta \models \omega$.*

Proposition 5. *Let $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C$ and $C_{i,1}, C_{i,2}$, such that $C_{j,i} \cup C_{k,i} = C_i$, for each $i, j, k \in \{1, 2\}$ with $j \neq k$ and $C_{1,i} \cap C_{2,i} = \emptyset$. Let $\omega \in \Omega(C)$ such that, for each $x + n\mathcal{R}y + m \in \omega$, either $x, y \in C_{1,1}$ or $x, y \in C_{2,1}$. We have that $reset(\omega, C_1, C_2) \equiv reset(\omega, C_{1,1}, C_{1,2}) \wedge reset(\omega, C_{2,1}, C_{2,2})$.*

Theorem 6. *Let p be a closed process, A be a closed RTL_μ formula and $\omega \in \Omega(C \cup rel(p))$. If $p, \omega \vdash A$, then $p, \omega \models A$.*

The proof of soundness is based on showing that each rule is backward sound. The soundness of fixpoints rules follows from the following reduction lemma that justifies the use of tagging fixpoints.

Lemma 7. *Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$. For $P \in \mathcal{P}(D)$,*

1. $P \subseteq \psi(\mu V.(\psi(V) \setminus P)) \Rightarrow P \subseteq \mu V.(\psi(V))$;
2. $P \subseteq \psi(\nu V.(\psi(V) \cup P)) \Leftrightarrow P \subseteq \nu V.(\psi(V))$.
3. *If $\mu V.\psi(V) = \psi^k(\emptyset)$ for some integer k, then there exists $\{P_i\}_{i\in\{1,n\}}$, such that $P = \bigcup_{i\in\{1,n\}} P_i$, and, for each $i \in \{1,n\}$, $P_i \subseteq \mu V.\psi(V) \Rightarrow P_i \subseteq \psi(\mu V.(\psi(V) \setminus P_i))$.*

The details of proofs can be found in [16].

Theorem 8. *Let p be a regular process, A be a closed RTL_μ formula and $\omega \in \Omega(C \cup rel(p))$. If $p, \omega \models A$, then $p, \omega \vdash A$.*

A sketch of the proof of completeness can be found in the appendix A.

6 Conclusions

In this paper we have considered a quite general discrete timed process language $\mathcal{TSP}$ which is parametric in the set of basic actions. An operator of process refinement and the use of locations permit to model in its framework a discrete version of timed statecharts with minimal and maximal delays associated to transitions. We have defined a compositional proof system for $\mathcal{TSP}$ based on a non-trivial adaptation of the techniques proposed in [5] for a simple CCS-like language. The logic for expressing real-time properties is an extension of μ-calculus with freeze quantification and clock constraints over a discrete time domain.

We are currently interested in the realization of a tool for providing the user with mechanized assistance during the construction of the proofs. We are studying the embedding of our formal system in the proof assistant *Coq* [7]. This is very important since very simple proofs only can be done by hand.

One of the goals of a future research it would be to extend our approach to a dense time framework. The proposed model of timed statecharts is not adequate for modelling applications working whithin a dense environment, that may produce signals at arbitrary close instants of time. Dense timed versions of statecharts have been proposed in [12, 19]. However, it is not clear if the synchrony hypothesis is still reasonable in this dense framework. For the verification we could for instance consider an extension of μ-calculus with dense time such as the one proposed by [11].

References

1. R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, University of Stanford, 1991.
2. R. Alur and T.A. Henzinger. A really temporal logic. In *Proceeding of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.
3. R. Alur and T.A. Henzinger. Real-time logics: complexity and expressivness. In *Proc. Fifth IEEE Symp. on Logic In Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.
4. R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, Berlin, 1992.
5. H.R. Andersen, C. Stirling, and G. Winskel. A Compositional Proof System for the Modal μ-Calculus. In *Proc. Ninth IEEE Symp. on Logic In Computer Science*, pages 144–153, 1994.
6. G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.

7. C. Cornes, J. Courant, J. Fillatre, G. Huet, P. Manoury, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The Coq Proof Assistant Reference Manual-Version 5.10. Technical report, INRIA, Rocquenfort, 1995.
8. E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *First Annual Workshop on Computer-Aided Verification*, 1989.
9. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
10. D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. In *Proc. Second IEEE Symp. on Logic In Computer Science*, pages 54–64. IEEE Computer Society Press, 1987.
11. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111:193–244, 1994.
12. Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. In *Formal Techniques in Real-time and Fault Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 591–620. Springer-Verlag, Berlin, 1992.
13. D. Kozen. Results on the Propositional mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. F. Levi. A Compositional μ-Calculus Proof System for Statecharts. Submitted to Theoretical Computer Science, 1996.
15. F. Levi. A Process Language for Statecharts. In *Proc. of the LOMAPS Workshop on "Analysis and Verification of Multiple-Agent Languages"*, volume 1192 of *Lecture Notes in Computer Science*, pages 388–403, 1996.
16. F. Levi. *Verification of Temporal and Real-Time Properties of Statecharts.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1997.
17. A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *Proc. of CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 687–702. Springer-Verlag, Berlin, 1996.
18. R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
19. A. Peron and A. Maggiolo-Schettini. Transitions as Interrupts: A New Semantics for Timed Statecharts. In *Proceedings of International Symposium on Theoretical Aspects of Computer Software*, number 789 in Lecture Notes in Computer Science, pages 806–821. Springer-Verlag, Berlin, 1994.
20. A. Pnueli and M. Shalev. What is in a Step. In *Theoretical Aspects of Computer Science*, volume 526 of *Lecture Notes in Computer Science*, pages 244–464. Springer-Verlag, Berlin, 1991.
21. K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.
22. V. Pratt. A decidable mu-calculus. In *Proceedings of 22nd. FOCS*, pages 421–427, 1981.
23. C. Stirling and D. Walker. Local Model Checking in the Modal mu-Calculus. *Theoretical Computer Science*, 89:161–177, 1991.
24. G. Winskel. A Note on Model Checking the Modal Nu-calculus. *Theoretical Computer Science*, 83:157–187, 1991.

A Completeness of the proof system

In this appendix we show the main ideas underlying the proof of theorem 8. The proof is by induction on a relation $\prec$ over formulas that is well-founded for regular processes. For $p \in CTSP$, we denote by $\mathcal{R}_p$ the minimal subset of closed processes such that $p \in \mathcal{R}_p$, and if $p' \in \mathcal{R}_p$ and $(p', \pi') \overset{\alpha}{\mapsto} (p'', \pi'')$, then $p'' \in \mathcal{R}_p$. Let $\mathcal{S}_p$ be the minimal subset of closed processes such that, if p' is a subprocess of p, then $\mathcal{R}_{p'} \subseteq \mathcal{S}_p$. Regular processes are characterized by the fact that both $\mathcal{R}_p$ and $\mathcal{S}_p$ are finite. We show that there exists a finite quotient of $LTS(p)$ that is sufficient for establishing A by using a clock region technique [11, 1].

Definition 9. For a closed process p and $\omega \in \Omega(rel(p) \cup C)$, we say that ω is a (p, A)-*constraint* iff the greatest constant of ω is k, where k is the greatest constant that occurs in p or in A. A set of timed states $\mathcal{R}$ of $LTS(p)$ is a (p, A)-*region* iff, $(p', \pi') \in \mathcal{R}$ iff $p' \in \mathcal{S}_p$ and $\pi' \models \omega$ for some (p, A)-constraint ω.

The basic observation is that, if p is regular, there are only finitely many (p, A)-regions. Moreover, the set of states satisfying a formula A is a (p, A)-region.

Lemma 10. *For a closed process p and for a closed formula A, $\| A \|$ is a (p, A)-region.*

Definition 11. Let p be a closed process and let A, A' be closed formulas with tags (p', ω), where ω is a (p, A)-constraint. We define $A' \prec A$ iff either A' is a proper subformula of A or $A = \sigma X\{U\}B$, for $\sigma \in \{\mu, \nu\}$, and $A' = B[\sigma X\{U \cup \{(p', \omega)\}\}B/X]$ where $\omega \in \Omega(rel(p') \cup C)$ is a (p, A)-constraint, such that $\| (p', \omega) \| \cap \| U \| = \emptyset$ and $p' \in \mathcal{S}_p$.

For a regular process p the set of (p, A)-regions is finite and by lemma 10 fixpoints can be computed whithin a finite number of steps so that $\prec$ is well-founded. Moreover, we exploit the following property.

Lemma 12. *Let p be a closed process, A be a closed formula and $\omega \in \Omega(rel(p) \cup C)$. For $\alpha \in \mathcal{E}(Act)$, if $p, \omega \models < \alpha > A$ (resp. $p, \omega \models [\alpha]A$*

1. *there exist sets $\{p_i \in \mathcal{S}_p\}_{i \in \{1,n\}}$ and $\{p'_j \in \mathcal{S}_p\}_{j \in \{1,k\}}$, such that $\{p_i, \omega_i \models A\}_{i \in \{1,n\}}$ and $\{p'_j, \omega'_j \models true\}_{j \in \{1,k\}}$;*
2. *$p, \omega \vdash < \alpha > A$ (resp. $p, \omega \vdash [\alpha]A$)) can be derived from $\{p_i, \omega_i \vdash A\}_{i \in \{1,n\}}$ and $\{p'_j, \omega'_j \vdash true\}_{j \in \{1,k\}}$.*

Moreover, for each $i \in \{1, n\}$ and $\pi \models \omega$, there exists $(p, \pi) \overset{\alpha_i}{\mapsto} (p_i, \pi')$, where $\pi' \models \omega_i$, such that it is consistent with α.

Proof. (Theorem 8) If $p, \omega \models A$ and $\omega \wedge delay(p)$ is not satisfiable, by rule *empty*, $p, \omega \vdash A$. If $\omega \wedge delay(p)$ is satisfiable, it is sufficient to consider the case of $\models \forall(\omega \supset delay(p))$. The proof is by induction on $\prec$, that is well-founded for finite-state timed graphs.

1. The cases of $A \in \{P, \neg P, \omega, A_1 \vee A_2, A_1 \wedge A_2\}$ are trivial.
2. $A \equiv x.A_1$. By definition $p, \omega \models x.A_1$ iff, for each $\pi \models \omega$, $(s, \pi[x := 0]) \models A_1$. Since ω is satisfiable, by proposition 4 for each $\pi' \models reset(\omega, (C \cup rel(p)) \setminus \{x\}, x)$, there exists $\pi \models \omega$, such that $\pi' = pro_{(C \cup rel(p)) \setminus \{x\}}(\pi) \cup \{x := 0\}$. Since $x \in C$, then $\pi[x := 0] = \pi'$. Hence, $(s, \pi') \models A_1$, for each $\pi' \models reset(\omega, (C \cup rel(p)) \setminus \{x\}, x)$, i.e. $s, reset(\omega, (C \cup rel(p)) \setminus \{x\}, x) \models A_1$. Since $A_1 \prec A$, by induction hypothesis, $p, reset(\omega, (C \cup rel(p)) \setminus \{x\}, x) \vdash A_1$ and by rule *reset*, $p, \omega \vdash A$.
3. $A \equiv < \alpha > A_1$. By lemma 12 there exists $\{p_i, \omega_i \models A_1\}_{i \in \{1,n\}}$ and $\{p'_j, \omega'_j \models true\}_{j \in \{1,k\}}$, such that $p, \omega \vdash < \alpha > A_1$ is derivable from $\{p_i, \omega_i \vdash A_1\}_{i \in \{1,n\}}$ and $\{p'_j, \omega'_j \vdash true\}_{j \in \{1,k\}}$. By induction hypothesis proofs $\{p_i, \omega_i \vdash A_1\}_{i \in \{1,n\}}$ and $\{p'_j, \omega'_j \vdash true\}_{j \in \{1,k\}}$ indeed exist.
4. $A = [\alpha]A_1$. Similar to the previous case.
5. $A = \nu X\{U\}.B$. If $\| (p, \omega) \| \subseteq \| U \|$, then by ν_1, $p, \omega \vdash \nu X\{U\}.B$. Otherwise, by lemma 10, $\| \nu X\{U\}.B \|$ is a (p, A)-region, namely there exists a (p, A)-constraint $\omega' \in \Omega(rel(p) \cup C)$, such that $\models \forall(\omega \supset \omega')$ and $p, \omega' \models A$. We have $p, \omega \vdash A$ from $p, \omega' \vdash A$ by rule *thin*. By lemma 7, $(p, \omega') \in \| B[\nu X\{U \cup \{(p, \omega')\}.B/X] \cup U \|$. There are two cases:
   (a) If $\| (p, \omega') \| \cap \| U \| = \emptyset$, then $p, \omega' \models B[\nu X\{U \cup \{(p, \omega')\}.B/X]$. Moreover, since $B[\nu X\{U \cup \{(p, \omega')\}.B/X] \prec A$, by induction hypothesis $p, \omega' \vdash B[\nu X\{U \cup \{(p, \omega')\}.B/X]$. We have $p, \omega' \vdash A$ by applying rule ν_2.
   (b) If $\| (p, \omega') \| \cap \| U \| \neq \emptyset$, then there exists $\{(p, \omega_i) \in U\}_{i \in \{1,n\}}$, such that $\omega' \wedge \omega_i$ is satisfiable, for each $i \in \{1, n\}$. Consider $\omega'_1 \equiv \omega' \wedge (\bigvee_{i \in \{1,n\}} \omega_i)$ and $\omega'_2 \equiv \omega' \wedge \neg(\bigvee_{i \in \{1,n\}} \omega_i)$. It is obvious that $\omega' \equiv \omega'_1 \vee \omega'_2$. Hence, $p, \omega' \vdash A$ can be obtained by rule *disj* from $p, \omega'_1 \vdash A$ and $p, \omega'_2 \vdash A$. Moreover, $\models \forall(\omega'_1 \supset \bigvee_{i \in \{1,n\}} \omega_i)$. Hence, by rule ν_1, $p, \omega'_1 \vdash A$. On the other hand, $\| (p, \omega'_2) \| \cap \| U \| = \emptyset$. Therefore, $p, \omega'_2 \models B[\nu X\{U \cup \{(p, \omega'_2)\}.B/X]$ and, since $B[\nu X\{U \cup \{(p, \omega'_2)\}.B/X] \prec A$, by induction hypothesis $p, \omega'_2 \vdash B[\nu X\{U \cup \{(p, \omega'_2)\}.B/X]$. We have $p, \omega'_2 \vdash A$ by applying rule ν_2.
6. $A = \mu X\{U\}.B$. By lemma 7 and by lemma 10, there exist (p, A)-constraints $\omega_1, \ldots, \omega_k$, such that $\omega \equiv \bigvee_{i \in \{1,k\}} \omega_i$ and $p, \omega_i \models B[\mu X.\{U \cup \{(p, \omega_i)\}B/X]$. Moreover, by the semantics of tagged least fixpoints $\| (p, \omega_i) \| \cap \| U \| = \emptyset$. Therefore, $B[\mu X.\{U \cup \{(p, \omega_i)\}B/X] \prec A$ and by induction hypothesis, $p, \omega_i \vdash B[\mu X.\{U \cup \{(p, \omega_i)\}B/X]$, for each $i \in \{1, k\}$. By applying rules *disj* and rule μ, we have $p, \omega \vdash \mu X\{U\}.B$.