A proven translation from a UML state machine subset to timed automata

Florent Peres, Mohamed Ghazel

A Proven Translation from a UML State Machine Subset to Timed Automata

Florent Peres and Mohamed Ghazel

Abstract

Although UML state machines constitute a convenient modeling formalism that is widely used in many applications, the lack of formal semantics impedes carrying out some automatic processing such as formal verification for instance. In this paper, we aim to achieve a proven translation from a subset of UML state machines to timed automata. A generic abstract syntax is defined for state machines, which allows us to specify state machines as a tree-like structure, thus explicitly illustrating the hierarchical relationships within the model. Based on this syntax, a formal asynchronous semantics for state machines and systems of state machines is established. Additionally, the semantics of timed automata is specified. Then, a translation relation from the considered set of state machines to timed automata is defined and a strong equivalence relation, namely a timed bisimulation between the source and target models, is formally proven. The proof is carried out inductively while considering continuous (time) and discrete transitions separately. Such a proof allows us to feature a strong similitude between these models.

Index Terms

Model Transformation, UML State machines, Timed Automata, Bisimulation Equivalence.

1 Introduction

Model transformation is a research domain that has garnered a lot of interest from the academic community over the last decade. It consists in generating target models from source models according to some translation rules. From a practical point of view, transforming models may serve as a support for specifications interoperability. It can also be useful when some specific mature tools with their own specific input language are targeted within the system engineering process, thus avoiding the tedious task of developing ad-hoc tools which will take years to achieve an acceptable level of efficiency. On the other hand, model transformation can theoretically be used to make comparisons between different notations, syntactically or semantically.

In this paper, we carry out a transformation from UML (Unified modeling Language) State Machines (SM) [36] to Timed Automata (TA) [2]. SM are a state/transition language that offers a view of the different states in which a given system can be, as well as the transitions between states, while expressing the causes and effects of each transition. SM notation allows an explicit representation of hierarchy, which helps model compactness and readability. Moreover, like the various UML diagrams, SM have good expressiveness capacities and remain quite intuitive, which explains their wide use in both academia and industrial communities. The SM syntax considered here offers some extensions to the standard one [36], namely through time guards and transition labels, while it does not consider the whole features in the specifications. In our work, we adopt the SM semantics proposed in [15], as will be explained later in the paper.

In the literature, several works propose model transformation between dynamic models, such as in [13], [24], [28], [30], [40]. However, most of them do not prove their transformation to be sound, or only demonstrate weak equivalence relations between the source and target models. As for UML diagrams which constitute a set of standard notations quite appropriate for design, documentation and communication between various stakeholders for system engineering [18], several works have tried to develop means allowing the integration of these diagrams with formal techniques, particularly for the design and verification of embedded systems as in [3], [7], [10], [22], [32], [34].

The main contribution of the present work is that the translation we establish from state machines to Timed Automata (TA) is formally proven through a strong equivalence relation, namely a timed bisimulation. In fact, one of the challenges to be addressed when it comes to defining a semantics for hierarchical languages, such as state machines, is to determine how the system status swiches from one state to another. In the present work, we firstly define an abstract syntax for state machines, while making explicit the various features

• *F. Peres and M. Ghazel are with the University Gustave Eiffel - IFSTTAR, COSYS/ESTAS team, 59666 Villeneuve d'Ascq, France (contact: mohamed.ghazel@univ-eiffel.fr).*

considered. This syntax is based on a tree-like structure where the successive levels denote alternation between the SM regions and nodes, while featuring hierarchical relations within the SM structure. In particular, the tree-like structure derived from the state machine abstract syntax has been advantageously used to explicitly illustrate the impact of transition triggering. It is worth noting that most of the artifacts defined within the proposed abstract syntax are generic enough to be easily adapted, or even used directly for several other state/transition hierarchical notations. Contrarily to the work in [28] where both synchronous and asynchronous communications are considered, in our work we consider an asynchronous semantics that we formally define for SMs and SM systems (sets of interconnected SMs). One argument in favour of using asynchronous semantics is that the synchrony hypothesis, stating that any system action takes more time than the minimal time grain, does not need to be considered. As a common point with existing works dealing with SM transformation, the present work also considers a SM subset for which a formal semantics is proposed. Besides, it is worth noting here that there does not exist a single "standard" semantics of the whole UML SMs at least for the time being [7], [16], [19], [35]. In fact, the lack of formal semantics was an incentive for the UML user community to propose its own formal semantics that resulted in many formalisms more or less conforming to the original state machine specification, depending on what the authors wanted to achieve. [9] reports 25 semantics proposals for UML state machines. However, only few works have considered providing an actual formal semantics, *i.e.*, which relies on formal notations, such as in [28] or in [10]. We should also mention that we also extend the SM subset we consider with timing features and labels. Regarding timing features, one can recall here that the MARTE UML profile allows an explicit specification of time in real time applications [31], and a language named CCSL (Clock Constraint Specification Language) has been elaborated in order to specifically express clock constraints [4]. MARTE profile offers good expressiveness and flexibility, but on the other side it is not appropriate for formal verification without significant restriction. In [10], Damm et al. develop a formal semantics for concurrency and communication hile considering a subset of UML called *krtUML*.

From a theoretical point of view, proving a time bisimilarity equivalence between asynchronous state machines and timed automata allows us to demonstrate a strong similarity between both languages. An additional aim motivating this work is to be able to use theorem provers to validate our transformation process. With this goal being overly ambitious, the aim of the present work is then to soundly formalize the translation and to manually prove its correctness. This work is then a first step toward providing all the necessary ingredients to state the problem using theorem prover theories, since our formalization entirely uses set theory (theorem provers can all handle set theory).

The remainder of the paper is organized as follows. In section 2, we establish an abstract syntax for SMs as bipartite arborescence, before introducing a formal semantics for the considered SMs and for systems of SMs systems in section 3. Systems of TA are presented in section 4, where the syntax and semantics of such systems are formally discussed. The translation from the subset of SM systems to TA is detailed in section 5, while the proof of bisimulation between both models is established in section 6. Finally, section 7 recalls the main contributions of the present work and outlines relevant future works.

2 STATE MACHINE: ABSTRACT SYNTAX

Prior to formally giving the semantics of state machines, we need to define their abstract syntax. Indeed, the main challenge of defining a semantics for UML state machines is related to the hierachical tree-like structure of the states and the corresponding modifications by taking a transition. In this section, we will mainly focus on the state structure while disregarding the transitions between nodes and regions, which will be discussed later on in Section 3. Namely, the abstract syntax will be given as a tree-like structure, more precisely as a *bipartite arborescence*, which will be defined in the sequel. In fact, the various definitions that will be given in the remainder of this section will allow us to explicitely represent the hierachical structure of SM states.

A state-machine is composed of two types of *elements* : *nodes* and *regions* which fit inside one another. This is translated into graph theory by a partition of the vertices along with the obligation for each vertex type to be alternated: the successor (in the hierarchical structure) of a node is a region and the successor (in the hierarchical structure) of a region is a node. Any graph possessing this feature is called *bipartite* (Petri nets are another example of bipartite graphs). The successor relation of the bipartite arborescence representation of a state machine actually gives a member relationship between its elements. However, we will prefer the term *father* (and its opposite, *child*) to state that an element belongs to another element. All these concepts will be formalized below.

Before that, we will define the notion of *tuple* that will be used in the sequel to introduce various relevant artifacts in a clear and precise way. We will also define the notion of reachability, which will be helpful when it comes to extracting and giving valid paths (realizable paths) over trees.

2.1 Generalized Cartesian Product and Tuples

In the sequel, we will heavily rely on the generalized Cartesian product. This is why clear and concise definitions are needed to handle tuples. In particular, the notations introduced in the following definition will be used in the remainder of this paper.

Definition 1 (Tuple)**.** *Let X be a given set of n elements, $n \in \mathbb{N}$, and a set of sets $S \subseteq 2^X$ such that $S = \{s_i \subseteq X, i = 1 \dots m\}$. Then let O be a surjective mapping from $[1,n]$ to S* (i.e., *$\forall s_i \in S, \exists j \in [1,n]$ such that $O(j) = s_i$, thus* Card$(S) = m \leqslant n$*). We can define the* generalized Cartesian product *of sets s_i w.r.t. O as the relation $P = \bigcup_{i=1}^{n}(\{i\} \times O(i))$ that can be written $\bigtimes_{i=1}^{n} O(i)$ or $\prod_O S$. A* tuple *t can therefore be seen as a relation, namely as a map from $[1,n]$ to $\bigcup_{i=1}^{m}\{s_i\}$. As such, we can write $t \subseteq P$.*

This definition is quite complex as it defines tuples as maps; however, it is very convenient since it allows us to use functional notations to handle the components of a tuple, in a rigorous way, without defining specific ad-hoc notations. For example, if t is a tuple then $t(i)$ is the i^{th} component of t.

Note that we require the simple binary Cartesian product to be defined. A binary relation can be defined as a subset of a Cartesian product. $\{i\} \times O(i)$ is just a binary relation relating i with each element of the set $O(i)$. If we take $O(i) = \{a_1, \dots, a_k\}$, then $\{i\} \times O(i) = \{(i,a_1), \dots (i,a_k)\}$. If we make the union of this relation for each i, we obtain P which is in fact the generalized Cartesian product of all the sets $O(i)$. Recall that a map is a particular relation, for which every element of the domain set is mapped to one value in the co-domain. This means that any relation t whose properties are to be simultaneously a mapping and a subset of P, is of the form $t = \{(1,a_1), \dots, (n,a_n)\}$, which is a functional representation of the tuple $(a_1, \dots, a_n)$.

When we write $\bigtimes_{i=1}^{n} s_i$, we implicitly specify the mapping O as $O(i) = s_i$, and in that case O is a bijective mapping and can be thought of as an ordering (or a permutation) of the sets s_i (under the condition that $(\forall i,j)(i \neq j \Rightarrow s_i \neq s_j)$). In the sequel, we will assume that all the implicit O mappings are bijective.

2.2 Reachability of a relation

Definition 2 (Reachability of a relation)**.** *Let $\mathcal{F}$ be a set of α-tuples* (i.e., *tuples of dimension α, $\mathcal{F}$ is also called an α-ary relation, or a relation of arity α) such that $\alpha \geqslant 2$. Let $\mathcal{F}^*$ be the set of words obtained by concatenating the elements of $\mathcal{F}$. The reachability of the relation $\mathcal{F}$ is given by:*

$$\mathcal{F}^{\downarrow}_{root} = \{s \mid s \in \mathcal{F}^* \wedge s^1(1) = root \wedge (\forall j \in [1, |s|-1])(s^j(\alpha) = s^{j+1}(1))\} \quad (1)$$

where for a given word s, s^j is a function returning the j^{th} letter (which is actually a tuple) of s and $|s|$ denotes the length of s.

This operator is general and can be applied to any α-ary relation as long as this relation is at least a binary one. Indeed, it is a generalization of the "join" operator used on binary elements (pairs) to tuples (i.e. to any α-ary relation with $\alpha \geqslant 2$). It can be used for instance to give actual execution paths in a transition system, but in our case it will be useful for extracting any branch starting from a given *root* element, that is any path in a given transition system, which is in general a graph. Indeed, in graph theory, vertices in V are linked by a relation $\mathcal{F} \subseteq V \times V$. Hence, any pair in $\mathcal{F}$ can be considered as a single letter, and $\mathcal{F}^*$ then indicates the various successions of edges. However, as those edges may not actually be connected by a common vertex, not all strings of $\mathcal{F}^*$ (which are also strings of pairs *i.e.*, elements of $(V \times V)^*$) are actual paths of the graph $(V, \mathcal{F})$. Finally, if $root \in V$, then $\mathcal{F}^{\downarrow}_{root}$ allows us to actually give all the paths starting from the *root* element, since it gives all the words of $\mathcal{F}^*$ using $\mathcal{F}$ in a connected manner from the *root* vertex.

Example 1. *Let $\mathcal{F}$ be a set holding four α-tuples: ρ_1, ρ_2, ρ_3 and ρ_4.*

$$\mathcal{F} = \{ \underbrace{(\overbrace{root}^{\rho_1(1)}, \dots, \overbrace{a}^{\rho_1(\alpha)})}_{\rho_1}, \underbrace{(\overbrace{a}^{\rho_2(1)}, \dots, \overbrace{b}^{\rho_2(\alpha)})}_{\rho_2},$$
$$\underbrace{(\overbrace{b}^{\rho_3(1)}, \dots, \overbrace{c}^{\rho_3(\alpha)})}_{\rho_3}, \underbrace{(\overbrace{a}^{\rho_4(1)}, \dots, \overbrace{d}^{\rho_4(\alpha)})}_{\rho_4} \}$$

By definition, we have $\mathcal{F}^{\downarrow}_{root} = \{ \rho_1, \rho_1{\cdot}\rho_2, \rho_1{\cdot}\rho_2{\cdot}\rho_3, \rho_1{\cdot}\rho_4 \}$. It is the set of words starting from root, whose letters are "connectable" by $\mathcal{F}$. It is plain that $\mathcal{F}^{\downarrow}_{root} \subseteq \mathcal{F}^*$

Finally, we want to be able to transform the words in $\mathcal{F}^{\downarrow}_{e}$ into words of Σ^*, where e is an element of Σ, $\mathcal{F} = \bigcup_{i=1}^{n} \bigtimes_{j=1}^{m} X_i^j$ while the X_i^j represent the components of the tuples in $\mathcal{F}$ as stated in Definition 2, and

$\Sigma = \bigcup_{i=1}^{n} X_i^1 \cup X_i^m$ with X_i^1 and X_i^m being respectively the sets of the first and last components of the tuples. In other words, we want to transform a "connectible" word of $\mathcal{F}_n^{\downarrow}$ into a "connected" one (belonging to Σ^*). As an example, let us take $\rho_1.\rho_2.\rho_3$ in the previous example. We want to be able to transform it into the string "$root.a.b.c$".

Definition 3 (Connector Operator $[\![w]\!]$)**.** *Let $\mathcal{F}$ be an α-ary relation, with $\alpha \geqslant 2$ and e an element of the sets used as components in the tuples of $\mathcal{F}$* (i.e., *$e \in \Sigma$ as stated in the previous paragraph). To any word w of $\mathcal{F}_e^{\downarrow}$, $[\![w]\!] = m$ is a word of length $|m| = |w| + 1$, where $(\forall i \in [1, |w|])(m^i = w^i(1))$ and $m^{|w|+1} = w^{|w|}(\alpha)$. This operator can be extended to sets: $[\![\mathcal{F}_e^{\downarrow}]\!] = \{\ [\![w]\!] \mid w \in \mathcal{F}_e^{\downarrow}\ \}$.*

Intuitively, $\mathcal{F}_e^{\downarrow}$ can be viewed as the set of accessible paths of the transition relation $\mathcal{F}$, starting from e. Traditionally, a transition relation will be allowed to hold much more information than just the source and target states. This is not the case here, as we deploy it on graph structures using the simplest transition relation possible: $\mathcal{F} \subseteq V \times V$. In a transition relation, one expects that each transition is encoded by a tuple whose extremum components are the source and target states (for example, $R \subseteq Q \times \Sigma \times Q$ can give a transition relation whose states are in Q and each transition bears a label in Σ). This is the way transition relations are defined traditionally.

The connector operator $[\![w]\!]$ operates on a path $w \in \mathcal{F}_e^{\downarrow}$. Still, using the previous intuition, it will allow us to remove any information that is not related to states. We already mentioned that we consider "meaningful" states to be at the tuple extremums, which is exactly what we do here with the connector operator: we only keep those states. For example, if $w = (q_0, a, q_1).(q_1, b, q_2).(q_2, c, q_3)$, then $[\![w]\!] = q_0.q_1.q_2.q_3$. It effectively "connects" the extremums to obtain a state path.

Another similar operator will also be introduced below, which will be used to decompose a word into the set of its symbols.

Definition 4 (Decomposition Operator $(\!|\sigma|\!)$)**.** *Let σ be a word of Σ^*, the set containing all the symbols in σ (called decomposition of σ), and written $(\!|\sigma|\!)$, is $(\!|\sigma|\!) = \{\lambda \in \Sigma \mid (\exists i)(\sigma^i = \lambda)\}$*

For example, for $a, b, c \in \Sigma$, $(\!|abc|\!) = \{a, b, c\}$.

2.3 Bipartite Arborescence

Definition 5. *A bipartite arborescence $\mathcal{A} = (N, R, H, root)$ is a tuple in which:*

- *N and R are two disjoint sets of vertices, $N \cap R = \varnothing$,*
- *$(N \cup R, H)$ is a directed graph,*
- *$H \subseteq (N \times R) \cup (R \times N)$ is the hierarchy relation. The pairs of H are specified as 2-tuples using the generalized Cartesian product,*
- *$root \in N$ designates the root node of the arborescence.*

It must also respect both the following constraints:

1) $(\forall v \in N \cup R)(\exists!\ p \in H_{root}^{\downarrow})(p^{|p|}(2) = v)$. *This constraint states that for all vertices v, there exists a* unique *path (remark the ! in $\exists!$) from root to v.*
2) $(\forall p \in H_{root}^{\downarrow})\ (\forall i \in [1, |p| - 1])\ ([\![p]\!]^i \in N \Rightarrow [\![p]\!]^{i+1} \in R \wedge [\![p]\!]^i \in R \Rightarrow [\![p]\!]^{i+1} \in N)$. *This constraint ensures alternation between nodes and regions.*

A bipartite arborescence is a tree-shaped graph. As such, it contains no cycles. Every bipartite arborescence is also a directed acyclic graph (but the converse is not true).

2.4 Arborescence Specific Notations

In this section, we will introduce some new operators and functions together with relevant notations which will be useful in the sequel to allow for a concise definition of state machines semantics and for establishing the translation into timed automata.

Let $\mathcal{A} = (N, R, H, root)$ be a bipartite arborescence. All of the following definitions will operate on $\mathcal{A}$. N stands for *Nodes*, while R refers to *Regions*. In order to illustrate the various concepts that will be introduced successively, let us consider the SM in Fig. 1, and its corresponding arborescence[1] that is given in Fig. 2. In this latter, the elements of R are the ones named r_i, while the other elements are in N. Moreover, the direction of the hierarchy relation is bottom-up. In particular the *root* node is the one that is at the highest level of the hierarchy.

1. The reader should ignore the black tokens tagging some of the nodes of the arborescence in Fig. 2 for the moment. This tagging will be used later in the paper

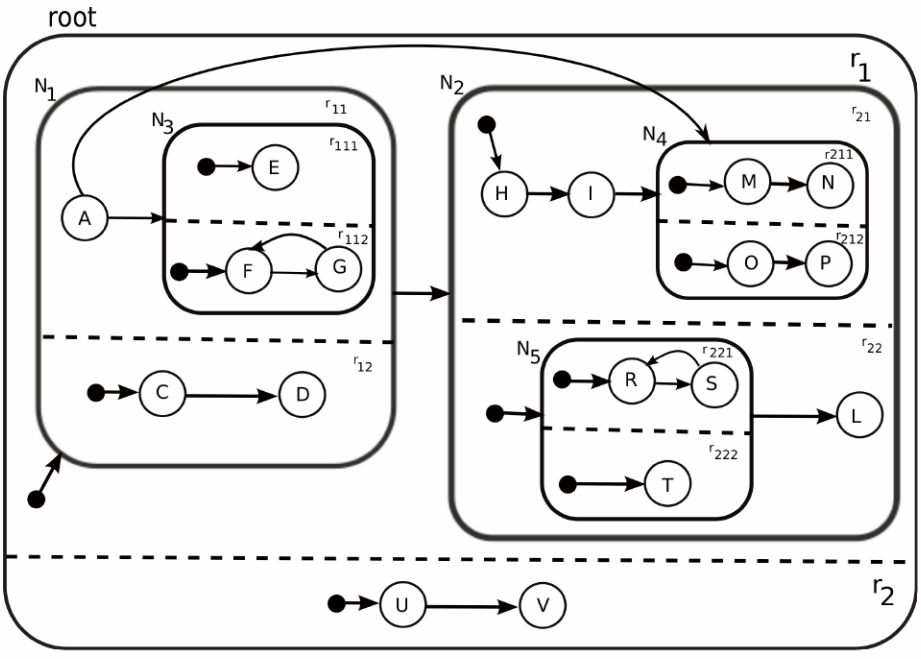

Fig. 1. A state machine example

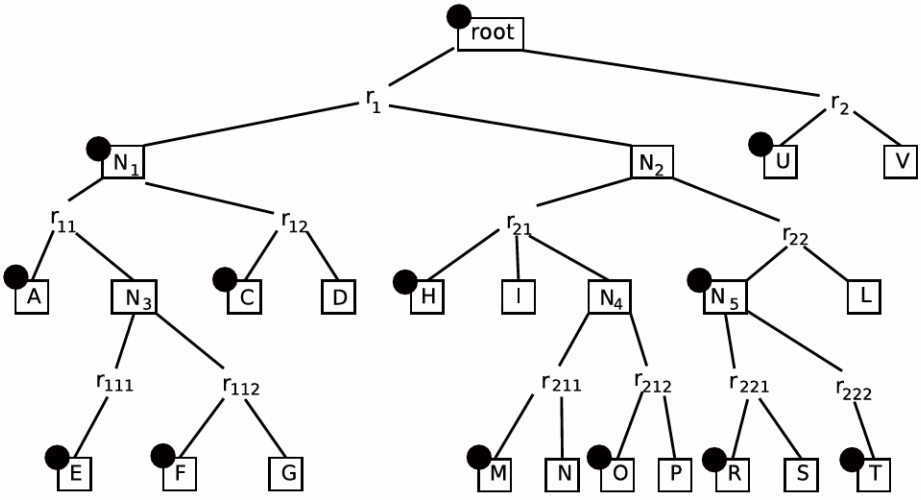

Fig. 2. The corresponding bipartite arborescence

Definition 6. *Let $n, m \in N \cup R$, we write $n \longrightarrow m$ to express that m is a son (or child) of n in the arborescence (and, thus, n is the father of m) : $n \longrightarrow m \Leftrightarrow (n, m) \in H$.*

Definition 7. *The transitive closure of $\rightarrow$ will be represented by the infix operator $>$. This means that for any $n, m \in N \cup R$, $n > m$ expresses that n is an ancestor of m,* i.e.*, there is a sequence $p \in H_n^{\downarrow}$, such that $p^{|p|}(2) = m$. Or equivalently, but in a recursive form: $(n > m)$ iff $((n \longrightarrow m) \vee (\exists q \in N \cup R, ((n \longrightarrow q) \wedge (q > m))))$*

Definition 8. *Two regions $r_1, r_2 \in R$ are* orthogonal *(denoted as $r_1 \perp r_2$) iff $\neg((r_1 > r_2) \vee (r_2 > r_1)) \wedge (\exists n \in N((n \longrightarrow r_1) \wedge (n \longrightarrow r_2)))$.*

For instance, by referring to the arborescence of Fig. 2, r_{11} and r_{12} are two orthogonal regions.

Definition 9. *Two nodes $n_1, n_2 \in N$ are* orthogonal *(denoted $n_1 \perp n_2$) iff $(\exists r_1, r_2 \in R)((r_1 \rightarrow n_1) \wedge (r_2 \rightarrow n_2) \wedge (r_1 \perp r_2))$,* i.e.*, n_1 and n_2 respectively belong to two orthogonal regions r_1 and r_2.*

As an example, in Figure 2, A and C are two orthogonal nodes.

Definition 10. *The sibling nodes of a node n form a set: $Siblings(n) = \{s \mid (\exists f \in R)((f \rightarrow n) \wedge (f \rightarrow s) \wedge (s \neq n))\}$.*

Example 2. *By referring to Fig. 2, $Siblings(H) = \{I, N_4\}$.*

Definition 11. *The strict descendants of a node n are given by the set* $\mathcal{D}^*(n) = \{m \in N |\ n > m\}$. *The non-strict version is defined as* $\mathcal{D} = \mathcal{D}^*(n) \cup \{n\}$. *Using a star* $(*)$ *is an abuse of notation, which is inspired from the use of* $\mathbb{N}^* = \mathbb{N}\backslash\{0\}$ *w.r.t.* $\mathbb{N}$, *to denote he set of strict positive integers.*

Example 3. *In the arborescence of Fig. 2,* $\mathcal{D}^*(N_4) = \{M, N, O, P\}$ *and* $\mathcal{D}(N_4) = \{N_4, M, N, O, P\}$.

Definition 12. *The set of* common ancestors *for a set* $S \subseteq N$ *of nodes is defined as:*
$CA(S) = \bigcap_{i \in S} \{e \in N \cup R \mid e > i\}$.

Example 4. *If we refer to Fig. 2, then* $CA(\{R, S, T, I\}) = \{N_2, r_1, root\}$.

Definition 13. *Let* x, y, w *be three words,* $w.x \cap w.y = w$ *iff* $x^1 \neq y^1$.

In other words, the intersection of two words returns the longest common prefix of both words.

Property 1. $(\forall S \subseteq N)(CA(S) = \varnothing \vee (\exists!\ p \in H^{\downarrow}_{root})(\Vert\!\llbracket p \rrbracket\!\Vert = CA(S))$

This property states that for a subset of nodes S, either they have no common ancestor, or all the common ancestors of S belong to a single path, and thus the set of the elements along that path is the set of the common ancestors of S. To prove this, we will introduce a new intersection operation on words. For that, we will reuse the set intersection symbol $\cap$.

Proof of Property 1: When $root \in S$, $CA(S) = \varnothing$. When $root \notin S$, let us suppose that $S = \{e_1, \ldots, e_n\}$. Each element e_i is reachable by a path $p_{e_i} \in H^{\downarrow}_{root}$, we then have $\Vert\!\llbracket p_{e_i} \rrbracket\!\Vert = \{n | n > e_i\}$. This means that $CA(S) = \bigcap_{i=1}^{n} \Vert\!\llbracket p_{e_i} \rrbracket\!\Vert$. Now, because H is tree-shaped, each path p_{e_i} must have a different ending (or else they all become a same path), or stated differently, as soon as two paths begin to differ, they will continue to differ (and the associated elements will not be included in $CA(S)$). This means that $CA(S) = \Vert\!\llbracket \bigcap_{i=1}^{n} p_{e_i} \rrbracket\!\Vert$. As H is a tree, there can only be one path p_{e_i} associated with each word $\llbracket p_{e_i} \rrbracket$, which means that $\bigcap_{i=1}^{n} \llbracket p_{e_i} \rrbracket = \llbracket \bigcap_{i=1}^{n} p_{e_i} \rrbracket$. This is also true for the intersection of words $\bigcap_{i=1}^{n} \llbracket p_{e_i} \rrbracket$. Let us suppose that $\bigcap_{i=1}^{n} \llbracket p_{e_i} \rrbracket = p_k$, we then have $CA(S) = \Vert\!\llbracket \bigcap_{i=1}^{n} p_{e_i} \rrbracket\!\Vert = \Vert\!\llbracket p_k \rrbracket\!\Vert$, which concludes the proof. □

Definition 14 (Min and Max)**.** *The* minimum *of a set* $S \subseteq N \cup R$ *is the subset formed by the elements in S which are located deepest in the arborescence. In other words, a node x is a minimum of S iff for any node y in S, either y is an ancestor of x, or y is x itself, or y is orthogonal to x.*
Formally, $Min(S) = \{x \mid x \in S \wedge (\forall y \in S)(\neg(x > y))\}$.
In a dual way, we have : $Max(S) = \{x \mid x \in S \wedge (\forall y \in S)(\neg(y > x))\}$.

Example 5. *By referring to Fig. 2, we have:*
- $Min(\{N_4, r_{211}, O, M, U, S, C, r_{111}\}) = \{O, M, U, S, C, r_{111}\}$, *and*
- $Max(\{N_4, r_{211}, O, M, U, S, C, r_{111}\}) = \{N_4, U, S, C, r_{111}\}$.

Definition 15. *The* Least Common Ancestor *of a set S is the minimum set of the common ancestors of S. Formally:* $LCA(S) = Min(CA(S) \cap (R \cup \{root\}))$.

Example 6. *In Figure 2,* $LCA(\{N, O, N_5, L\}) = \{r_1\}$.

Property 2. $(\forall S)(\text{Card}(LCA(S)) = 1)$

Proof of Property 2: This property comes from the fact that, by Property 1, all the common ancestors of a given node belong to a single path p originating from *root*. It is then clear that there is only one element which is minimal in $\Vert p \Vert$.

□

Remark 1. *When none of the elements in s corresponds to root, it is straightforward that* LCA(S) *is necessarily a region* $(\in R)$.

Definition 16 (Path)**.** *A* path *between s and t, denoted* $\mathcal{P}^t_s$, *is the set of nodes appearing on the path from s to t. Formally,* $\mathcal{P}^t_s = \{n \in N \mid s \leqslant n \leqslant t\}$. *In the sequel, we will write* $[s, t]$ *to denote* $\mathcal{P}^t_s$ *and* $[s, t[$ *to denote* $\mathcal{P}^t_s \backslash \{t\}$ (i.e., $[s, t[= \{n \in N \mid s \leqslant n < t\}$).
Here $<$ *and* $\leqslant$ *are, respectively, the dual operators of* $>$ *and* $\geqslant$ *defined earlier.*

Example 7. *In Fig. 2,* $\mathcal{P}^{root}_G = \{n \in N | G \leqslant n \leqslant root\} = \{G, N_3, N_1, root\}$.

2.5 State Machines as Bipartite Arborescences

As we have mentioned earlier, we do not consider the whole SM features w.r.t the specifications [36] or the more recent ones. For instance, we do not consider final states, *History* pseudo-states, entry/exit/do behaviors, forks/joins. Our goal through the present contribution is not to formalize the whole SM specifications. Instead, we aim to show how such a formalization can be defined on a SM subset, and to set the necessary artefacts to achieve a *sound* translation to a target formal notation. Naturally, the present contribution can serve as a basis for extensions to cope with further SM syntactical and/or semantic aspects. Another aspect which is worth mentioning in our work, is that the various syntactical choices we have made are guided by our primary objective stated above. Namely, some choices in terms of syntax representation may be deemed as complex. nevertheless, such choices can be explained by the fact that we essentially seek for facilitating the translation step and more importantly enabling to conduct the proof of the bisimilation relation between the source and the target models.
Now that all the necessary ingredients to present the abstract syntax of state machines have been introduced, we can give the definition of the subset of SM that we consider, while putting in light a number of extensions that we include.

Definition 17 (State Machine)**.** *A state machine is a 12-tuple* $S = (N, R, H, Tr, I, O, C, root, Default, UpB, \Sigma, \mathcal{L})$ *such that:*

- N *is a set of nodes*
- R *is a set of regions*
- $root \in N$ *is the root node*
- $(N, R, H, root)$ *forms a* bipartite arborescence*,*
- I *is the set of* input *events,*
- O *is the set of* output *events,*
- $C \subseteq \mathbb{B} \times \mathbb{N} \times ((\mathbb{N} \times \mathbb{B}) \cup \{(\infty, \top)\})$ *is the set of* time guards*. Here* $\mathbb{B} = \{\top, \bot\}$ *is the Boolean set, where* $\top$ *stands for true and* $\bot$ *for false. In fact, a guard is an interval whose bounds* $(\mathbb{N} \cup \{\infty\})$ *and strictness (*$\top$*: bound excluded and* $\bot$*: bound included) are given as a 4-tuple in* C*.*
- $Tr \subseteq N \times 2^I \times C \times 2^O \times N$ *is the set of* transitions*,*
- $Default \subseteq N$ *is the set of* default *nodes, which correspond to the nodes that are target of a transition the source of which is an initial pseudo-state. While defining the Default nodes in a SM, the following must be fulfilled:* $((\forall r \in R)(\exists! n)((r \rightarrow n) \wedge (n \in Default))) \wedge (root \in Default)$*. Here* $(\exists! n)$ *means "there exists a unique n".*
- $UpB : N \rightarrow (\mathbb{N} \times \mathbb{B}) \cup \{(\infty, \top)\}$ *assigns an* upper bound *delay (possibly infinite) to each node,*
- Σ *is a set of symbols containing the empty letter* ϵ*,*
- $\mathcal{L} : Tr \rightarrow \Sigma$ *assigns a label to each transition,*
- *Finally, each region of S must contain at least one node:* $(\forall (s,t) \in H \cap (N \times R))(\exists u \in N)((t,u) \in H \cap (R \times N))$*.*

A transition holds a lot of information: the source and target nodes, the input and output events, and the clock guard which induces some implicit corresponding clocks. The input events will be required to be present in the environment to enable the transition. As the transition fires, each input is consumed and each output is produced. The precise semantics will be given shortly.

A clock guard is in fact an interval, representing the relative time instants between which the transition can trigger. These time intervals are specified using a tuple of $\mathbb{B} \times \mathbb{N} \times \{(\mathbb{N} \times \mathbb{B}) \cup \{(\infty, \top)\}\}$. The two Booleans represent the strictness of the bounds. The first bound (the lower bound) is given by a natural number, while the second one is either a natural number or ∞. The latter is a bit special, as it is not influenced by strictness, as we will see in details in the semantics description section.

The root element is a node, as UML specifications authorize many regions at the highest level. More precisely, a state machine is defined by the regions it contains, and those regions must hold some nodes which in turn can hold some regions, and so on. This means that the upper item defining the state machine must be a node in the data structure which we set up to represent the abstract syntax. The root node is initially activated ($root \in Default$) and will always, in any circumstances, remain so.

Each region contains one and only one *default* son node. Each time the region is entered by a transition, such that no particular node of this region (or a sub-region) is explicitly targeted, its default node is activated.

Each node can have an *upper bound* sejourn delay, which is an upper-bound condition on time elapsing that must not be violated. The time counting starts as the node is activated (relative time), and time elapsing is limited to the constraint in UpB, provided that the node is continuously activated.

The triggering of a transition is tracked using a label. Those labels do not constrain the behavior of the machine, but will be useful when it comes to stating some properties about its behavior. One particular

property which will be considered here is bisimulation equivalence. Bisimulation does not intrinsically require transitions to be labelled. Besides, it was originally defined on KRIPKE structures, but this labelling will strengthen the equivalence. Indeed, due to the duality between states and transitions, if an event-based bisimulation equivalence exists between two structures, then the corresponding underlying state based bisimulation also holds. It is worth mentioning that this mechanism is not specified by the UML specification, but as we have already taken some liberties regarding this specification (notably through time guards), and because this is a harmless extension, it should not be regarded as problematic.

The last constraint is related to the bipartite arborescence data structure. As a data structure, it allows for any kind of alternation between nodes and regions. But in the context of SMs, nodes and regions are not strictly equal objects. Indeed, only nodes have a substantial existence in SMs. Namely, nodes can be viewed as containers, while regions are only used to split those containers. Hence, there is no sense in splitting if there is no container to be split, but *also* (and maybe more importantly) if there is no true object inside the space that would be created by the split. The constraint then requires that no leaf in the arborescence can be a region.

Finally, we should mention that, in the literature, there exist various formalizations of hierarchy. As an example, in [29] the authors also use a tree-like structure to develop a formalization for hierarchical models.

3 STATE MACHINE SEMANTICS

3.1 Toward Semantics: Impact of transitions

One of the most important aspects regarding the semantics of hierarchical languages, such as state machines, is to know which states are exited and and which ones are entered upon a transition firing. When both the source and the target nodes of a transition belong to the same region, the expected behavior is straightforward: the source node is deactivated, and the target node is activated. However, this is only true if both nodes are leaves in the SM arborescence. More noticeably, this does not hold when the transition exits from one region and enters another one. In those cases, knowing which nodes are deactivated and which are activated is not straightforward any more.

We will solve this problem by defining, for each transition, what we will call *impacts*. There are two sorts of impacts: the *source impact* and the *target impact*. The source impact is the set of all the nodes which will be deactivated upon the transition firing, while the target impact represents the set of all the nodes which will be activated as a consequence of the transition firing.

To be more detailed, a state of a SM is given by its activated nodes, the value of its clocks and the events waiting to be *consumed*. The part of the state corresponding to the set of activated nodes will be called *nodal configuration*. When a transition triggers, its source (resp. target) impact will be removed (resp. added) from (resp. to) the current nodal configuration.

In order to illustrate the source impact, let us consider the state machine in Fig. 1, and its corresponding arborescence structure in Fig. 2. Also, recall that the nodes tagged with a black token correspond to the default nodes.

In order to establish the formal definition of the source and target impacts, let us introduce the following definition.

Definition 18 (Default Activated Nodes (DAN))**.**
Let n be a node in the state machine, we define $\mathrm{DAN}(n)$ *as:*

$$\mathrm{DAN}(n) = \bigcup_{p \in H_n^{\downarrow} \wedge (\forall i \in [1,|p|])([\![p]\!]^i \in (\{n\} \cup Default \cup R))\}} (\!|[\![p]\!]|\!) \cap N$$

$\mathrm{DAN}(n)$ is in fact the set of nodes that belong to any path in the state machine arborescence, starting in n and going through regions and default nodes only.

Example 8. *In the state machine in Fig. 1, and while referring to its corresponding arborescence in Fig. 2, if we consider node N_4 we can state that:*
$H_{N_4}^{\downarrow} = \{N_4.r_{211}.M, N_4.r_{211}.N, N_4.r_{212}.O, N_4.r_{212}.P\}$.
From $H_{N_4}^{\downarrow}$ we keep only paths of which all the elements are either regions ($\in R$), or default nodes ($\in Default$), or equal to N_4. Thus, we obtain the set of two paths $\{N_4.r_{211}.M, N_4.r_{212}.O\}$. Then the decomposition operator $(\!|.|\!)$ is applied on these two paths, giving the set $\{N_4, r_{211}, M, r_{212}, O\}$. Finally, we keep only nodes ($\cap N$): $\mathrm{DAN}(N_4) = \{N_4, M, O\}$.
Similarly, we obtain $\mathrm{DAN}(N_2) = \{N_2, H, N_5, R, T\}$.

Definition 19 (Source Impact)**.**
Let s_1 and s_2 be two nodes, and $\{h_{s_1}^{s_2}\} = Max([s_1, \mathrm{LCA}(\{s_1, s_2\})[)$. It is clear here that Max returns a singleton, since

all the elements of $[s_1, \mathrm{LCA}(\{s_1, s_2\})[$ *belong to the same path. The source impact of a transition going from* s_1 *to* s_2 *corresponds to the set of nodes to be deactivated as result of the firing of the transition, and is given by:*

$$^{\bullet}\mathcal{I}_{s_1}^{s_2} = \mathcal{D}(\{h_{s_1}^{s_2}\}) \setminus \bigcup_{i \in [s_1, \{h_{s_1}^{s_2}\}[} \mathcal{D}(Siblings(i))$$

The source impact first concerns every node in $[s_1, \mathrm{LCA}(\{s_1, s_2\})[$. If neither s_1 nor s_2 corresponds to the *root*, $\mathrm{LCA}(\{s_1, s_2\})$ is the lowest region of the common ancestors of s_1 and s_2 that is *not* impacted by the transition firing. In other words, the transition from s_1 to s_2 takes place entirely *within* $\mathrm{LCA}(\{s_1, s_2\})$. We say that $\mathrm{LCA}(\{s_1, s_2\})$ is the *scope* of the transition. This also means that $\{h_{s_1}^{s_2}\}$ is the highest node which has to be deactivated upon the firing. Moreover, each node which is a descendant of a node in $[s_1, \mathrm{LCA}(\{s_1, s_2\})[$ must also be deactivated. Each node in $[s_1, \mathrm{LCA}(\{s_1, s_2\})[$ is also a descendant of $h_{s_1}^{s_2}$, which implies that all the nodes that have to be deactivated belong to $\mathcal{D}(\{h_{s_1}^{s_2}\})$. However, in fact, not all the nodes in $\mathcal{D}(\{h_{s_1}^{s_2}\})$ are to be deactivated. Namely, the siblings of the nodes in $[s_1, \mathrm{LCA}(\{s_1, s_2\}[$ do not need to be deactivated, for the simple reason that it is not possible that they are currently activated. Nevertheless, deactivating a deactivated node would not be a problem for the semantics, but this would cause problems with respect to the translation to timed automata. Indeed, we will rely on both impacts in the translation, and one important hypothesis is that *only* nodes which *could* be activated (and then would have to be deactivated) must belong to the source impact.

One can also notice that, as a consequence of the above definition, orthogonal regions of a deactivated node are left simultaneously, which is consistent with the OMG specifications [36].

As a side note, we use a strict upper bound for $[s_1, \{h_{s_1}^{s_2}\}[$ since $Siblings(\{h_{s_1}^{s_2}\}) \cap \mathcal{D}(\{h_{s_1}^{s_2}\}) = \varnothing$. It is then useless to remove the descendents of the siblings of $\{h_{s_1}^{s_2}\}$ from $\mathcal{D}(\{h_{s_1}^{s_2}\})$.

By referring to Fig. 1, let us consider the transition from node A to node N_4. Looking at the corresponding arborescence in Fig. 2, we can determine all the components in the $^{\bullet}\mathcal{I}_A^{N_4}$ formula:

- $\mathrm{LCA}(\{A, N_4\}) = r_1$, $[A, \mathrm{LCA}(\{A, N_4\})[= \{A, N_1\}$, then $h_A^{N_4} = N_1$
- $\mathcal{D}(h_A^{N_4}) = \mathcal{D}(N_1) = \{A, N_3, E, F, G, C, D\}$.
- On the other hand $[A, h_A^{N_4}[= \{A\}$, thus $\bigcup_{i \in [A, h_A^{N_4}[} \mathcal{D}(Siblings(i)) = \mathcal{D}(Siblings(A)) = \mathcal{D}(N_3) = \{N_3, E, F, G\}$.
- Finally, $^{\bullet}\mathcal{I}_A^{N_4} = \{A, N_3, E, F, G, C, D\} \setminus \{N_3, E, F, G\} = \{A, C, D\}$.

Definition 20 (Target impact)**.**
Let s_1 *and* s_2 *be two nodes, and let* $\rho = [s_2, \mathrm{LCA}(\{s_1, s_2\})[$. *The target impact of a transition going from* s_1 *to* s_2 *corresponds to the set of nodes to be activated as result of the firing of the transition, and is given by:*

$$\mathcal{I}^{\bullet s_2}_{s_1} = \bigcup_{i \in \rho} \mathrm{DAN}(i) \setminus \bigcup_{i \in \rho} \mathcal{D}(Siblings(i))$$

Using a symmetrical argument to the one used for the source impact, every node in $\rho = [s_2, \mathrm{LCA}(\{s_1, s_2\})[$ must be activated upon the triggering of the transition. In each node i in ρ, we must activate all the nodes in the set $\mathrm{DAN}(i)$ corresponding to its default activated nodes. Recall that $\mathrm{DAN}(i)$ contains i itself and all the sub-nodes of i which have to be activated as soon as i is activated. However, there are exceptions for that, namely we need to ensure that two sibling nodes cannot be simultaneously active, hence for all nodes $i \in \rho$, we have to prevent the activation of their siblings, and obviously the descendants of these latter as well $(\setminus \bigcup_{i \in \rho} \mathcal{D}(Siblings(i)))$.

One can also notice that, as a consequence of the above definition, orthogonal regions of an activated node are entered simultaneously, which is consistent with the OMG specifications [36].

To illustrate the concept of *target impact*, let us again consider the transition from node A to node N_4 in the state machine of Fig. 1. While referring to its corresponding arborescence given in Fig. 2, we obtain:

- $\mathrm{LCA}(\{A, N_4\}) = r_1$, $\rho = [N_4, \mathrm{LCA}(\{A, N_4\})[= \{N_4, N_2\}$
- $\mathrm{DAN}(N_4) = \{N_4, M, O\}$, $\mathrm{DAN}(N_2) = \{N_2, H, N_5, R, T\}$
- $\bigcup_{i \in \rho} \mathrm{DAN}(i) = \{N_4, M, O, N_2, H, N_5, R, T\}$
- $\bigcup_{i \in \rho} \mathcal{D}(Siblings(i)) = \mathcal{D}(Siblings(N_4)) \cup \mathcal{D}(Siblings(N_2)) = \{H, I\} \cup \{N_1, A, N_3, E, F, G, C, D\}$
  $= \{H, I, N_1, A, N_3, E, F, G, C, D\}$
- Finally, $\mathcal{I}^{\bullet N_4}_{A} = \{N_4, M, O, N_2, N_5, R, T\}$

3.2 State Machine Semantics

Our work is based on the semantics proposed in [15]. This choice is justified by the fact that this semantics proposes at the same time a somewhat sound formalisation using an extension of labelled Kripke structures (namely, Clocked Labelled Kripke Structure, or CLKS), but also includes most of the concepts of the state machine syntax. As will be discussed in the sequel, some aspects of the aforementioned semantics will not be included in our semantics, while some others will be different, like the synchronization mechanism and the fact that no difference will be made between stable and unstable states.

We have made the choice of asynchronous communication since our goal is to achieve a translation into timed automata, which can natively manipulate real-time information and can serve as a model for automatic formal verification. The other widely known formalism in that category is Time Petri nets (TPN) [33]. The choice between the two notations is more a matter of taste and tools availability than anything else. Indeed, we could have used the TPN formalism without much difficulty (see [6] regarding the transformation of Timed Automata into a Time Petri nets, or [8] which makes a structural translation from Time Petri nets to Timed Automata). One of the key issues here would be the composition of Time Petri nets, which can mostly be resolved using forbid/allow Petri nets as demonstrated in a previous work [37].

As mentioned earlier, no difference is made in the present work between unstable and stable states, since this simplifies the semantics of state machines and, as a consequence, the translation to timed automata is simpler (along with the proof of the preservation of the bisimulation equivalence).

3.2.1 Semantics: Informal Presentation

A state machine may be an *open* or a *closed* system. A closed system does not require another machine to work, while an open machine should be linked to at least one other machine, maybe waiting for some inputs, or just sending some outputs, or even both. This openness requirement is not explicitly expressed by the syntax: each machine is able to interact with another one through the use of a global set of events E. An open machine does know that an event was generated by another one, but cannot determine which one (except, of course, if the events are partitioned according to each machine). In the case of a closed system, a machine is in fact linked with itself, namely it is the only one that can *read* and *write* in E.

The set E can equivalently be viewed as a set of 1-bounded queues of events. The queue model behavior is such that each arrival of an event is stored in a separate queue for each event, with the size of the queue being arbitrarily fixed either by the semantics or by the specifications. This point of view is equivalently represented by a set when each queue has a size of 1, meaning that a queue only stores one instance of an event (the first one of a series). In a more abstract view, E is the medium of asynchronous communications, using 1-bounded buffers. This mechanism is quite similar to the token game of Petri nets: to be equivalent, we would only have to generalize the set E to a multi-set [14]. In fact, state machine formalisms (among which UML SMs) obey to the *run-to-completion* paradigm (*RTC*), assuming that a SM completes the processing of each event before it can start processing the next event. In other terms, events are processed in a discrete manner in RTC models, through "indivisible" steps. In other terms, new incoming events cannot interrupt the processing of the current event and must be stored (typically in an event queue) until the state machine becomes idle again. although we consider different queues for event handling, this should not impact the behavior of the SM w.r.t. the RTC paradigm.

In addition, the state machine formalisation we propose associates one clock with each node and allows any transition to be constrained (using clock guards) according to these clocks. Instead of using the timed automata approach which explicitly references clocks, a transition will only be able to be constrained (in terms of time constraints) by the clock associated with its *source* node, thus mimicking the time constraints of time Petri nets in that regard. The values of each clock at a given instant constitute what we call the *clock configuration* of the state machine at that instant.

From what has been presented so far, the reader should have a fairly clear impression of how the nodal part will behave. Initially, only the nodes in the default paths will be activated. When a transition triggers, it deactivates each active node in its source impact, and activates all those in its target impact. At any time, there can only be one node activated in each region. The activated nodes at a given time form the *nodal configuration* of the state machine. The nodal configuration can evolve upon transition firing. Namely, a transition is able to fire if and only if the following holds:

- its source node belongs to the current nodal configuration,
- its time guard respects the clock configuration,
- and finally if its event requirements are all included in the current state of the set E of global events.

3.2.2 Temporal Guards

Traditionally, state machines specify time constraints using two basic mechanisms: *after* and *when*. These two keywords state how much time has elapsed since some given reference time (*ex.* date of activation of a node) using *after*, or considering an absolute date since the start of the machine using *when*. Moreover, UML provides the opportunity to add some constraints using OCL (Object Constraint Language) [38]. However, this mechanism is either too powerful if we consider that clocks can be manipulated by OCL constraints, or too restrictive if we do not consider the existence of clocks. One easy, yet powerful, way of dealing with this issue is to completely redefine what clock constraints can be. In addition, as we already sketched it out, blackwe use a similar representation to that of Time Petri nets, where unbounded time intervals can be assigned to transitions.

3.2.3 State machine systems

A state machine system (SMS) composed of n SMs is a pair $\langle \mathbb{S}, \mathbb{E} \rangle$, where:

- $\mathbb{S} = \bigcup_{i=1}^{n} \{s_i\}$ is a set of state machines, with $s_i = (N_i, R_i, H_i, Tr_i, I_i, O_i, C_i, root_i, D_i, UpB_i, \Sigma_i, \mathcal{L}_i)$,
- $\mathbb{E} = \bigcup_{i=1}^{n} O_i \cup I_i$ is the global set of events

The state of a machine is given by the triple (a, b, c), in which:

- $a \in \bigtimes_{i=1}^{n} (2^{N_i} \backslash \{\varnothing\})$ is the *nodal configuration* holding all of the *local* nodal configurations, *i.e.*, the active nodes of each state machine. When referring explicitly to a machine, as in "the nodal configuration of SM s_i", s_i, $1 \leqslant i \leqslant n$, we mean the *local* nodal configuration of s_i, *i.e.*, a non-empty subset of N_i. We require every local configuration to be non empty. An element of $2^{N_i} \backslash \{\varnothing\}$ is then a set of nodes, that are active in the machine s_i.
- $b \in \bigtimes_{i=1}^{n} \mathbb{R}_+^{|N_i|}$ with $\mathbb{R}_+ = [0, \infty[$, is the *time configuration*, holding all the *local* time configurations, *i.e.*, the clock valuations for each machine. A local time configuration is a mapping assigning to each node in N_i a non-negative real value. For a given nodal configuration, a time configuration b is a tuple of size n. In turn, each i^{th} component in this tuple is a tuple of size $|N_i|$, holding a non-negative real value that corresponds to the value of the local clocks associated with each of the nodes of s_i.
- $c \in 2^{\mathbb{E}}$ is the *event configuration*: it is a global set which holds the events that are required by some transition to trigger, which are present.

In the sequel, when we write s_i (or $n_j \in N_i$), we implicitly specify that s_i is the machine associated with index i, *i.e.*, there exists a mapping $O(i) = s_i$.

3.3 Semantics of a State Machine System

To define the semantics of our system of state machines, we will use the standard formalism for languages manipulating dense time, namely Timed Transition Systems.

Definition 21. *A* Timed Transition System *(denoted* TTS *in the sequel) is a tuple* $(Q, q_0, \rightarrow, \Sigma)$*, in which:*

- Q *is a set of* states,
- q_0 *is the* initial *state,*
- $\rightarrow = D \cup C$*, where:*
  - $D \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ *is the underlying* discrete *transition relation*
  - $C \subseteq Q \times \mathbb{R}_+ \times Q$ *is the underlying* continuous *transition relation*
- Σ *is an alphabet of actions, it does* not *contain the silent action* ϵ.

As usual, a discrete transition $(q, a, q') \in \rightarrow$ is also denoted $q \xrightarrow{a} q'$.
As for continuous transitions, they obey the following rules. Let $\theta, \theta' \in \mathbb{R}^+$:

- *null delay*: $q \xrightarrow{0} q' \Leftrightarrow q = q'$
- *additivity*: $q \xrightarrow{\theta} q' \wedge q' \xrightarrow{\theta'} q'' \Rightarrow q \xrightarrow{\theta+\theta'} q''$
- *continuity*: $q \xrightarrow{\theta+\theta'} q' \Rightarrow (\exists q'')(q \xrightarrow{\theta} q'' \wedge q'' \xrightarrow{\theta'} q')$
- *temporal determinism*: $q \xrightarrow{\theta} q' \wedge q \xrightarrow{\theta} q'' \Rightarrow q' = q''$

As one may guess, the transition relation will be the major part of the semantics. This is why we split the definition into two distinct parts: one will present the state (and events) related properties of the TTS, and the other will be devoted to giving the transition relation separately.

$$
\begin{array}{l}
(C,\gamma,\lambda) \xrightarrow{a\in\Sigma} (C',\gamma',\lambda') \Leftrightarrow \\
(\exists!\ s_j \in \mathbb{S})(\exists t \in Tr_j) \\
\left(\begin{array}{l}
t = (n_k, i, c, o, n_l) \Rightarrow \\
\left(\begin{array}{lll}
\mathcal{L}_j(t) = a & \wedge & (1) \\
n_k \in C(j) & \wedge & (2) \\
(\exists l_s, u_s \in \mathbb{B})(\exists l_b \in \mathbb{N})(\exists u_b \in \mathbb{N}\cup\{\infty\}) & & \\
\left(\begin{array}{l}
c = (l_b, l_s, u_b, u_s) \Rightarrow \\
\left(\begin{array}{lll}
l_s = \top & \Rightarrow & \gamma(j)(k) > l_b \quad \wedge \\
l_s = \bot & \Rightarrow & \gamma(j)(k) \geqslant l_b \quad \wedge \\
u_s = \top & \Rightarrow & \gamma(j)(k) < u_b \quad \wedge \\
u_s = \bot & \Rightarrow & \gamma(j)(k) \leqslant u_b
\end{array}\right)
\end{array}\right) & \wedge & (3) \\
i \subseteq \lambda & \wedge & (4) \\
\lambda' = (\lambda \backslash i) \cup o & \wedge & (5) \\
C'(j) = (C(j) \backslash^{\bullet} \mathcal{I}^{n_l}_{n_k}) \cup (\mathcal{I}^{\bullet n_l}_{n_k}) & \wedge & (6) \\
(\forall n_p \in \mathcal{I}^{\bullet n_l}_{n_k})(\gamma'(j)(p) = 0) & \wedge & \\
(\forall n_p \notin \mathcal{I}^{\bullet n_l}_{n_k})(\gamma'(j)(p) = \gamma(j)(p)) & & (7)
\end{array}\right)
\end{array}\right)
\end{array}
$$

Fig. 3. SMS: Action transition semantics

3.3.1 States and Events

Let $S = \langle \mathbb{S}, \mathbb{E} \rangle$ be a state machine system in which $\mathbb{S} = \bigcup_{i=1}^{n} \{s_i\}$ with:
$s_i = (N_i, R_i, H_i, Tr_i, I_i, O_i, C_i, root_i, D_i, UpB_i, \Sigma_i, L_i)$. Recall that $\mathbb{E}$ is the global set of input and output events
The semantics of an SMS is given by the timed transition system $T_{sm} = (Q_{sm}, q^0_{sm}, \rightarrow_{sm}, \Sigma_{sm})$, where:

- $Q_{sm} \subseteq (\bigtimes_{i=1}^{n} 2^{N_i} \backslash \{\varnothing\}) \times (\bigtimes_{i=1}^{n} \mathbb{R}_+^{|N_i|}) \times (2^{\mathbb{E}})$ is the set of states that are reachable from q^0_{sm} by the transition relation $\rightarrow_{sm}$,
- $q^0_{sm} = (\bigtimes_{i=1}^{n} \{\Delta_i\}, \bigtimes_{i=1}^{n} \{\{0\}^{|N_i|}\}, \varnothing)$ is the initial configuration of S, where $\bigtimes_{i=1}^{n} \{\Delta_i\}$ is the initial nodal configuration, where the initial local configuration Δ_i of state machine s_i is defined as follows:

$$\Delta_i = \mathrm{DAN}(root_i) \tag{2}$$

  In fact, we consider one (local) clock per node; this will be useful to perform the translation and explain the bisimilation relation later on in the paper. Finally, $\varnothing$ is the initial event configuration. We can emphasize that the local configuration is similar to the target impact of a transition with *root* as a target node (such a transition is in principle possible).
- $\Sigma_{sm} = \bigcup_{i=1}^{n} \Sigma_i$.

3.3.2 Transition Relation

Defining the semantics of (dense) time formalism is often accomplished in two parts: the untimed behavior (related to discrete action transitions) on one hand, and the timed aspects (related to continuous time transitions) on the other hand. We will not derogate from this good practice.

3.3.2.1 Action Transitions: this first type of transition is untimed, and can also be referred to as *discrete* transitions. It is labelled with an *action* (also called event, but to avoid confusion with state machine events, we prefer the term *action*) which is considered visible (or accessible) to the user.

Let $t \in \rightarrow_{sm}$ and $a = \mathcal{L}(t)$ with $a \in \Sigma_{sm}$. The transition t, labelled by a, can trigger from a TTS state corresponding to configuration (C, γ, λ), to a state (C', γ', λ'), iff it respects the formula given by Fig. 3.

We should notice that TTS states and state machines configurations are in fact similar, but we will reserve the term *configuration* for the context of state machine systems (*i.e.*, when talking about their semantics), and instead use the term *state* for more technical, TTS-level details.

The initial configuration (C, γ, λ) is composed of a nodal configuration C, a time configuration γ and of an event configuration λ. From this configuration, a transition t, matching the tuple (n_k, i, c, o, n_l) and belongs to machine s_j, can trigger if and only if all the following seven conditions hold:
• (1) t is labelled by a;
• (2) the source node of t, namely n_k, belongs to the nodal configuration of s_j;
• (3) the clock guard c, defines the time interval in which t can fire. l_s and u_s respectively define the strictness of the bounds, *i.e.*, whether the bound is included in the interval. Recall that a bound is strict when it is true ($\top$).

As stated earlier, to each node of the SMS, we asssign a clock which serves as a counter for the time that has elapsed since the last activation of that node. The value of a clock associated with a node n_k is stored in γ, the global clock configuration of the system. To get the local clock configuration of the system s_j $(1 \leqslant j \leqslant n)$ we need to project γ on its j^{th} dimension, *i.e.*, $\gamma(j)$. This local clock configuration holds the value of each clock associated with every node of the state machine s_j. The value of the clock associated with the node n_k can then be retrieved, as it is stored in the k^{th} component of the local clock configuration $\gamma(j)$, that is the value of the clock associated with n_k is given by $\gamma(j)(k)$.

The last important remark about that part of the definition, is related to the manipulation of ∞. Indeed, u_b can be unbounded, which can be expressed simply by $\gamma(j)(k) < \infty$. Moreover, there is an equally unconstrained form for the lower bound, namely if $l_s = \bot$ and $l_b = 0$, then $0 \leqslant \gamma(j)(k)$, which is always true for any clock. Hence, clock constraint $(\bot, 0, \infty, \top)$ can be used to set no constraint on the transition;

- (4) the input event set i must be a subset of the current event configuration λ.
- (5) the resulting event configuration λ', after the firing of t, is λ in which all the input events have been removed $(\lambda \backslash i)$, which emulates their *consumption* (or *reception*, if the events are viewed as communication messages) by the transition; and the output events in o are added to the remaining set of events, emulating their *generation* (or *emission*, from a communication point of view) by the transition t $((\lambda \backslash i) \cup o)$. Note that if $o \cap (\lambda \backslash i) = s \neq \varnothing$ (*i.e.*, at least one of the generated events is already in the set of events, even after removing the inputs), then the effect of any event in s is lost. This emulates the fact that we try to send a message in a buffer which is already full;
- (6) the nodal configuration $C(j)$ of machine s_j is transformed into the nodal configuration $C'(j)$ by removing all the nodes of the source impact of t (*i.e.*, ${}^{\bullet}\mathcal{I}^{n_l}_{n_k}$) and by adding every node of the target impact of t (*i.e.*, $\mathcal{I}^{\bullet n_l}_{n_k}$);
- (7) finally, the clocks associated with the nodes n_p in the target impact of t are reinitialized $(\gamma'(j)(p) = 0)$, while the other clocks remain unchanged.

3.3.2.2 Time Transitions: Similarly to the case of discrete transitions, we will show the condition for a continuous transition t to trigger. This transition t transits from the configuration (or TTS state) (C, γ, λ) to the configuration (C', γ', λ'), after a positive delay $\theta \in \mathbb{R}^*_+ =]0, \infty[$.

$$(C, \gamma, \lambda) \xrightarrow{\theta \in \mathbb{R}^*_+} (C', \gamma', \lambda') \Leftrightarrow (\forall s_i \in \mathbb{S})$$
$$\left(\begin{array}{ll} C' = C & \wedge \\ \lambda' = \lambda & \wedge \\ (\forall n_j \in N_i)(\gamma'(i)(j) = \gamma(i)(j) + \theta) & \wedge \\ (\forall n_j \in C(i))(\exists (u_b, u_s) \in \{\mathbb{N} \times \mathbb{B}\} \cup \{\{\infty\} \times \top\}) & \\ \left(\begin{array}{l} UpB(n_j) = (u_b, u_s) \Rightarrow \\ \left(\begin{array}{lll} u_s = \top & \Rightarrow & \gamma'(i)(j) < u_b \quad \wedge \\ u_s = \bot & \Rightarrow & \gamma'(i)(j) \leqslant u_b \end{array} \right) \end{array} \right) & \end{array} \right)$$

The transition is taken *after* the time delay has elapsed. The nodal configurations C and the event configuration λ stay unaffected by this continuous transition ($C' = C$ and $\lambda = \lambda'$), as a continuous transition in the TTS is not related to any transition of any state machine.

The only state machine feature which actually influences time elapsing is the node upper bound delays (UpB). Indeed, time elapsing influences every clock $\gamma(i)(j)$ of every node $n_j \in N_i$ (activated or not) in every state machine $s_i \in \mathbb{S}$. The clocks are shifted according to the value of θ (*i.e.*, $\gamma'(i)(j) = \gamma(i)(j) + \theta$), which means that time elapsing is *not* absolute w.r.t the start of the system. It is instead related to the last elapsing of time.

An arguably cleaner way of handling clock elapsing would have been to only advance the clocks of the activated nodes, *i.e.*, nodes in C, but this would also mean adding another condition on clocks of inactivated nodes (their values stay unchanged). Simply shifting every single clock, without caring about the activation of their associated node, will make things easier when it comes to the translation and its proof (see Section 6). Using this "trick" is not a problem because the clocks associated with the target impact nodes of an action transition — those that become activated — are reset by the transition anyway (cf. first line of condition (7) in Figure 3).

Last, but not least, comes the core of the constraints on time elapsing regarding upper bounds. namely, for each active node n_j in every machine (*i.e.*, all nodes of the current nodal configuration (written $\forall n_j \in C(i)$, where i ranges over all machines), the upper bound delay of n_j restricts the advance of time, as it requires that the time shift θ does not make the new clock value (remember that the new clock value $\gamma'(j)(k) = \gamma(i)(j) + \theta$, see the third line in the semantics of time transitions above) exceed the upper bound delay u_b (strictly or not, depending on u_s).

4 TIMED AUTOMATA

Timed automata have been proposed for the first time by *Alur and Dill* [1] in 1990. Those automata have been extended with location invariants by *Henzinger (et al.)* [20]. Invariants are very helpful for a formalism like Timed Automata, because they allow urgency of actions. Indeed, the original semantics of timed automata proposes a so-called *weak* semantics of time, in the sense that in the basic definition, time can indefinitely elapse by any amount in any location. This means that timed automata could not easily specify important features of real-time systems like deadlines of transitions.

Moreover, we will use broadcast channels, to model the hierarchy of state machines [21], [27]. More specifically, we will use the UPPAAL [27] semantics of broadcast communication channels. Finally, we use variables [5], [23], [25], [26], to specify asynchronous communication. Variables would not have been mandatory if we did not want to use the semantics of timed automata as defined by UPPAAL. Indeed, to model message generation and consumption, we would also require synchronization channels. However, atomically, only one message can be received/sent at some given time, due to the communication semantics of UPPAAL. Therefore, not using variables would leave us only two alternatives: either to redefine our own communication semantics, or to use interleaving to model communication.

Interleaving would be theoretically possible but is pragmatically out of the question. Indeed, one of the goals of the transformation is to allow verification of state machines. Using an interleaving of actions to model an atomic transition would add many more states (and transitions). As model-checking already suffers from state explosion, using such a solution would then contribute to render the verification of our generated timed automata even more difficult.

On the other hand, defining our own communication semantics would not be a trivial work. Doing multiple way communication with broadcast in a single step is actually hard to manage in a general way. So, we would have certainly fallen back to a simpler *ad-hoc* communication semantics, more specific to our problem. That would be only the easy part. In fact, to actually verify our Timed Automata, we would have to implement a tool like UPPAAL, which is definitely *not* a simple task (and would decidedly go against one of our goals: using powerful *mature* tools).

Finally, in terms of verification, targeting UPPAAL TA allows us to take advantage of the various facilities offered by this mature tool, such as model-checking, simulation, etc.

4.1 Systems of Timed Automata Extended with Boolean Variables and Broadcast Synchronization

A timed automata system with Boolean variables, synchronizable using broadcast communication channels is a tuple $(\mathbb{A}, \mathbb{C}, \mathbb{V})$, in which:

- $\mathbb{A} = \bigcup_{i=1}^{n} \{A_i\}$ is the set of communicating timed automata, where: $A_i = (L_i, \ell_0^i, X_i, T_i, Inv_i, G_i, C_i, R_i, \mathbb{T}_i, \mathbb{F}_i, Syn_i, \Sigma_i)$,
- $\mathbb{C}$ is the set of communication channels,
- $\mathbb{V}$ is a set of Boolean variables.

A timed automaton is a tuple $A = (L, \ell_0, X, T, Inv, G, C, R, \mathbb{T}, \mathbb{F}, Sync, \Sigma)$, in which:

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the initial location,
- X is a finite set of clocks,
- $T \subseteq L \times \Sigma \times L$ is the set of transitions, $(\ell, a, \ell') \in T$ is also written $\ell \xrightarrow{a} \ell'$,
- $Inv : L \to 2^{X \times \{<, \leqslant\} \times \mathbb{R}^+}$ maps an *invariant* (possibly empty) to each location,
- $G : T \to 2^{X \times \{<, \leqslant, =, \geqslant, >\} \times \mathbb{R}^+}$ maps a set of *clock guards* possibly empty to each transition,
- $C : T \to 2^{\mathbb{V}}$ maps a *Boolean condition* (possibly empty) to each transition,
- $R : T \to 2^{X}$ maps a set of clocks (possibly empty) to be *reset*, to each transition,
- $\mathbb{T} : T \to 2^{\mathbb{V}}$ maps a set of variables possibly empty that must be set to true, to each transition,
- $\mathbb{F} : T \to 2^{\mathbb{V}}$ maps a set of variables (possibly empty) that must be set to false, to each transition, with $\mathbb{T} \cap \mathbb{F} = \varnothing$. Intersection is used here since mappings ($\mathbb{T}$ and $\mathbb{F}$ here) are also relations, thereby sets,
- $Syn : T \to (\mathbb{C} \times \{!, ?\}) \cup \{\varnothing\}$ maps a synchronization label (possibly empty) to each transition,
- Σ is a finite alphabet of actions.

A clock guard assigned to a clock t is, in fact, a triple (c, o, v), where c is a clock variable, $o \in \{<, \leqslant, =, \geqslant, >\}$ is a comparison operator and v a non-negative real value. For example, expression "$c \leqslant 3$", stating that the clock c must be lower than or equal to 3, is formally written as the triple $(c, \leqslant, 3)$. It works in the same way for invariants except that, for the latter, guards are mapped to locations, and use a restricted set of comparison operators (only $<$ and $\leqslant$).

Mappings C, R, $\mathbb{T}$ and $\mathbb{F}$ all work in the same way: they map to a transition a set of Boolean variables or clocks. In the case of C, the variables associated with a transition t must all be true for t to be able to trigger.

As for R, when t triggers, every clock in R is reset. Finally, $\mathbb{T}$ and $\mathbb{F}$ respectively specify which Boolean variable must be set to true, and which ones to false.

Synchronization labels are of two sorts. Those ending with a "*!*" specify a sending (or *master*) transition which initiates the communication. As it is a broadcast communication, it does not expect any listener to be ready to listen, which means that a master transition can always fire, provided that the other firing conditions are fulfilled. The transition holding a synchronization label ending with "*?*" are receiving (or *slave*) transitions. A slave transition is blocked until a master transition sending on the same channel is ready to fire. When a master and some slaves are ready to fire, they are all taken synchronously*i.e.*, they all fire in a single step (atomically). No slave ready to fire can be left out of the synchronous step when a master on the same channel fires.

4.2 Semantics of Timed Automata Systems

The semantics of a Timed Automata System $(\mathbb{A}, \mathbb{C}, \mathbb{V})$ is the Timed Transition System $(Q, q_0, \rightarrow, \Sigma)$, in which:

- $\mathbb{A} = \bigcup_{i=1}^{n} \{\ (L_i, \ell_0^i, X_i, T_i, Inv_i, G_i, C_i, R_i, \mathbb{T}_i, \mathbb{F}_i, Syn_i, \Sigma_i)\ \}$,
- $Q \subseteq \left(\bigtimes_{i=1}^{n} L_i\right) \times \left(\bigtimes_{i=1}^{n} \mathbb{R}_+^{|X_i|}\right) \times 2^{\mathbb{V}}$,
- $q_0 = \left(\bigtimes_{i=1}^{n} \{\ell_0^i\}, \bigtimes_{i=1}^{n} \{\{0\}^{|X_i|}\}, \varnothing\right)$,
- $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ is the alphabet of actions,
- $\rightarrow\ \subseteq Q \times (\Sigma \cup \mathbb{R}_+^*) \times Q$ is the transition relation.

This semantics uses the same internal TTS representation: it is a triple, whose first component holds the current location of each automaton ($\bigtimes_{i=1}^{n} L_i$), the second component holds the clock valuation of each automaton ($\bigtimes_{i=1}^{n} \mathbb{R}_+^{|X_i|}$) and the third one holds the subset of the Boolean variables that are true in the state ($2^{\mathbb{V}}$).

Initially, the location control is set to each of the initial locations of all the automata ($\bigtimes_{i=1}^{n} \{\ell_0^i\}$), all the clocks are set to zero ($\bigtimes_{i=1}^{n} \{\{0\}^{|X_i|}\}$) and no Boolean variable is true ($\varnothing$).

As before, we will split the definition of the timed automata semantics into discrete and continuous parts.

4.2.1 Discrete Transition

A discrete step concerns the actual firing of one automaton transition or one synchronous set of transitions.

We will first define some useful functions regarding enabledness and synchronizations.

Definition 22 (Enabledness)**.** *A transition t is said to be* enabled *w.r.t a configuration $(\mathfrak{L}, C, \mathcal{V})$ of a system $(\mathbb{A}, \mathbb{C}, \mathbb{V})$, if it is firable without taking synchronizations into account:*
$enabled(t, \mathfrak{L}, C, \mathcal{V}) = (\exists i, A_i \in \mathbb{A})(\exists \ell, \ell' \in L_i)(\exists a \in \Sigma_i)$

$$\left(\begin{array}{ll} t \in T_i & \wedge \\ t = \ell \xrightarrow{a} \ell' & \wedge \\ \ell = \mathfrak{L}(i) & \wedge \\ C(i) \models G_i(t) & \wedge \\ C(i) \models Inv_i(\ell) & \wedge \\ C_i(t) \subseteq \mathcal{V} & \end{array}\right)$$

The above definition refers to local firings, *i.e.*, to transitions that are fired by a single TA. In fact, the definition states that transition t is enabled if its source location ℓ is active, *i.e.*, $\ell = \mathfrak{L}(i)$. Here, the "$\models$" operator is defined as follows:

Definition 23. *Let $C \in \mathbb{R}^{|X|}$, $G \subseteq X \times \{<, \leqslant, =, \geqslant, >\} \times \mathbb{R}^+$, then $C \models G \Leftrightarrow (\forall g \in G)(g = (c_i, o, v) \Rightarrow \quad C(i) \quad o \quad v)$. This means that every guard $g \in G$ must be satisfied by the clock valuation vector C. To check that, we extract the value $C(i)$ of clock c_i and apply it to the comparison operator o. Here, "$C(i)\ o\ v$" must be understood as the application of the infix operator o to its operands $C(i)$ and v. For example, if $C(i) = 1.2$, $v = 3$ and $o = \leqslant$, then this amounts to checking whether "$1.2 \leqslant 3$".*

Returning to the definition, we check that the current clock values respect the guard of t, $G_i(t)$ and the invariant of ℓ, $Inv_i(\ell)$.

Finally, the Boolean guard of t, stated as a set $C_i(t)$ must be included in $\mathcal{V}$, as it means that all the variables in $C_i(t)$ are currently true.

Semantics 1 Timed Automata System

$$
\begin{array}{l}
(\mathfrak{L},C,\mathcal{V}) \overset{a\in\Sigma}{\rightarrow} (\mathfrak{L}',C',\mathcal{V}') \Leftrightarrow \\
(\exists i, A_i \in \mathbb{A})(\exists t \in T_i)(\exists \ell,\ell' \in L_i)(\exists a \in \Sigma_i) \\
\left(\begin{array}{lll}
t = \ell \overset{a}{\rightarrow} \ell' & \wedge & \\
\ell = \mathfrak{L}(i) & \wedge & \\
\ell' = \mathfrak{L}'(i) & \wedge & \\
Enabled(t) & \wedge & \\
\left(\begin{array}{l}
(\exists c \in \mathbb{C})(\exists y \in \underset{A_j\in \textsc{SynA}_i^c}{\times} \textsc{SynT}_j^c) \\
\left(\begin{array}{lll}
Syn_i(t) = (c,!) & \wedge & \\
\textsc{SynA}_i^c \neq \varnothing & \wedge & \\
(\forall A_j \in \textsc{SynA}_i^c) & & \\
\left(\begin{array}{l}
y(j) = \ell_1 \xrightarrow{(c,?)} \ell_2 \Rightarrow \mathfrak{L}(j) = \ell_1 \wedge \mathfrak{L}'(j) = \ell_2 \quad \wedge \\
(\forall x_k \in X_j) \\
\left(\begin{array}{lll}
x_k \in R_j(y(j)) & \Rightarrow & C'(j)(k) = 0 \\
x_k \notin R_j(y(j)) & \Rightarrow & C'(j)(k) = C(j)(k)
\end{array}\right)
\end{array}\right) & \wedge & \\
(\forall A_j \in \mathbb{A}\backslash \textsc{SynA}_i^c) & & \\
\left(\begin{array}{l}
\mathfrak{L}(j) = \mathfrak{L}'(j) \quad \wedge \\
C(j) = C'(j)
\end{array}\right) & \wedge & \\
\mathcal{V}' = \left(\mathcal{V} \setminus \left[\left(\underset{A_j\in \textsc{SynA}_i^c}{\cup} \mathbb{F}_j(y(j))\right) \cup \mathbb{F}_i(t)\right]\right) \cup \\
\qquad \left(\underset{A_j\in \textsc{SynA}_i^c}{\cup} \mathbb{T}_j(y(j))\right) \cup \mathbb{T}_i(t) & &
\end{array}\right) \\
\vee \\
\left(\begin{array}{ll}
\left(\begin{array}{l} Syn_i(t) = \varnothing \\ (Syn_i(t) = (c,!) \wedge \textsc{SynA}_i^c = \varnothing) \end{array}\right) & \wedge \\
(\forall x_k \in X_j) & \\
\left(\begin{array}{ll}
x_k \in R_i(t) \Rightarrow C'(i)(k) = 0 & \wedge \\
x_k \notin R_i(t) \Rightarrow C'(i)(k) = C(i)(k) &
\end{array}\right) & \wedge \\
(\forall A_m \in \mathbb{A})\left(\begin{array}{l}
A_m \neq A_i \Rightarrow \\
\left(\begin{array}{ll}
\mathfrak{L}(m) = \mathfrak{L}'(m) & \wedge \\
C(m) = C'(m) &
\end{array}\right)
\end{array}\right) & \wedge \\
\mathcal{V}' = (\mathcal{V} \setminus \mathbb{F}(t)) \cup \mathbb{T}(t) &
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
\begin{array}{l}
\left.\begin{array}{l}\\\\\\\end{array}\right\} (1) \\
\left.\begin{array}{l}\\\end{array}\right\} (2) \\
\left.\begin{array}{l}\\\\\\\end{array}\right\} (3) \\
\left.\begin{array}{l}\\\end{array}\right\} (4) \\
\left.\begin{array}{l}\\\\\end{array}\right\} (5) \\
(6) \\
(7) \\
\left.\begin{array}{l}\\\end{array}\right\} (8) \\
\left.\begin{array}{l}\\\end{array}\right\} (9)
\end{array}
$$

Definition 24 (Synchronization sets)**.** *Let $(\mathbb{A},\mathbb{C},\mathbb{V})$ be a timed automata system. The set of transitions of automaton A_j which are synchronizable on the channel c with a transition $t \in T_i$ of automaton A_i at the state $(\mathfrak{L},C,\mathcal{V})$, with $Syn_i(t) = (c,!)$, is given by:*
$\textsc{SynT}_j^c(\mathfrak{L},C,\mathcal{V}) = \{t \in T_j \mid enabled(t,\mathfrak{L},C,\mathcal{V}) \wedge Syn_j(t) = (c,?)\}$.

The set of automata which do not participate in the synchronization on channel c with transition t at state $(\mathfrak{L},C,\mathcal{V})$, is given by the set $\textsc{NoSyn}^c(\mathfrak{L},C,\mathcal{V}) = \{A_j \in \mathbb{A} \mid (\forall t \in T_j)(Syn_j(t) = (c,?) \Rightarrow \neg(enabled(t,\mathfrak{L},C,\mathcal{V})))\}$.

The set of automata participating in the synchronization on the channel c with the transition $t \in T_i$ of automaton A_i at the state $(\mathfrak{L},C,\mathcal{V})$ is given by the set $\textsc{SynA}_i^c(\mathfrak{L},C,\mathcal{V}) = \mathbb{A}\backslash\{\textsc{NoSyn}^c(\mathfrak{L},C,\mathcal{V}) \cup \{A_i\}\}$.

In the following, the state context will be omitted when it is clear from the context. In particular, in the definition below, all the synchronization sets and the enabledness will obviously be stated with regard to the source state $(\mathfrak{L},C,\mathcal{V})$. We will then write $Enabled(t)$, $\textsc{SynA}_i^c$, $\textsc{SynT}_i^c$.

Semantics 1 defined below may be hard to read at a first glance. The reader should consider firstly reading the subsequent comments and getting a better idea as to how the definition is structured, and secondly checking, along the reading, the formal definition as soon as what is being defined is understood.

Semantics 1 can be interpreted as follows. In the Timed Transition System, a step labelled with a must originate from a transition t labelled with a. Transition t must be enabled and when it fires, it gives the control to the target location ℓ'. This is specified by $\ell' = \mathfrak{L}'(i)$ which can be read as "the next location of automaton A_i is ℓ'" (cf. condition (1) in Semantics 1).

We then arrive at a point from which we check if there will be a set of transitions or only one transition (t) to trigger. In any case, t is either a master transition (whether or not any other slave transition fires with it), or it bears no synchronization label.

In the first case, a channel c serves as the synchronization medium (this is specified as $Syn_i(t) = (c,!)$, which

means that t is labelled by the synchronization label $(c, !)$, which in turn specifies the channel that is used, here c, and the direction of the communication, here "!" specifies a broadcast *emission*), which means that there is at least one slave transition which has to be synchronized with t on c, and this last fact is given by $\text{SYNA}_i^c \neq \varnothing$ (cf. condition (2) in Semantics 1).

Then, product $\bigtimes_{A_j \in \text{SYNA}_i^c} \text{SYNT}_j^c$ gives the possibilities in terms of synchronization. Indeed, only one transition per automaton can be synchronized with t (*cf.* UPPAAL's semantics). This means that when there are more of them in the same automaton, there will be a non-deterministic choice of the synchronized transitions. The product ranges over automata A_j which participate as slaves, *i.e.*, for which at least one slave transition is synchronizable with t.

Then we take a tuple y of the product α. This means we make a choice amongst all the possible firing combinations. This tuple is of size $\text{Card}(\text{SYNA}_i^c)$, and each of its components contains a synchronizable slave transition.

For each automaton A_j participating in the synchronization, the transition $y(j) = \ell_1 \rightarrow \ell_2$ has been chosen to synchronize on channel c, and the new current location of automaton A_j is given by $\mathfrak{L}'(j) = \ell_2$. The clocks associated with the reset set R_j of the slave transition $y(j)$ are all reset to zero ($C'(j)(k) = 0$). The values of the other clocks are kept unchanged upon the firing (cf. condition (3) in Semantics 1).

For each automaton A_j *not* participating in the synchronization, the locations and clock values are kept unchanged (cf. condition (4) in Semantics 1).

Finally, the new set of true variables is computed as follows: we remove all the variables which must be set to false. This means the union of the sets $\mathbb{F}_i(t)$ and $\mathbb{F}_j(y(j))$, i being the rank of automaton A_i, the one holding t, and j the rank of A_j, ranging on all the synchronizable automata. Likewise, we add the sets of variables that must be set to true, *i.e.*, the union of $\mathbb{T}_i(t)$ and $\mathbb{T}_j(y(j))$, for all the A_j that participate in the synchronization (cf. condition (5) in Semantics 1).

The other case is when transition t fires alone. There are two alternatives for which t fires alone. The first one is simply because it does not hold any synchronization label ($Syn_i(t) = \varnothing$) condition (6). The other possibility, is that even if t is a master transition, there is no slave transition which is enabled ($\text{SYNA}_i^c = \varnothing$) (cf. condition (7)).

In any case, all the clocks in $R_i(t)$ must be reset, and all the other clocks are left unchanged (cf. condition (8)). Moreover, because only t fires, the values related to the other automaton $A_j \neq A_i$ are also left unchanged (location-wise $\mathfrak{L}(m) = \mathfrak{L}'(m)$ and clock-wise $C(m) = C'(m)$) (cf. condition (9)).

4.2.2 Time Transition

When time elapses, no location, nor any variable modification occurs ($\mathfrak{L} = \mathfrak{L}'$ and $\mathcal{V} = \mathcal{V}'$). The only constraint is that of the invariants of the active locations. This is stated as follows: time can elapse by an amount of θ units of time, as long as the resulting clock valuation C' respects the invariants of the active locations $\mathfrak{L}(i)$ of each automaton A_i.

$$(\mathfrak{L}, C, \mathcal{V}) \overset{\theta \in \mathbb{R}_+^*}{\rightarrow} (\mathfrak{L}', C', \mathcal{V}') \Leftrightarrow$$
$$\left(\begin{array}{ll} \mathfrak{L} = \mathfrak{L}' & \wedge \\ \mathcal{V} = \mathcal{V}' & \wedge \\ (\forall A_i \in \mathbb{A}) & \\ \left(\begin{array}{ll} (\forall x \in X_i)(C'(i)(x) = C(i)(x) + \theta) & \wedge \\ C'(i) \models Inv_i(\mathfrak{L}(i)) & \end{array} \right) & \end{array} \right)$$

5 FROM STATE MACHINES SYSTEMS TO TIMED AUTOMATA SYSTEMS

Let $\langle \mathbb{S}, \mathbb{E} \rangle$ be a System of State Machines, with $|\mathbb{S}| = n$. We aim to translate the SMS $\langle \mathbb{S}, \mathbb{E} \rangle$ into a Timed Automata System $(\mathbb{A}, \mathbb{C}, \mathbb{V})$. Also, let us denote by $m = |\mathbb{A}|$.

Let $\mathbb{S} = \bigcup_{i=1}^{n} \{s_i\}$ and $s_i = (N_i, R_i, H_i, Tr_i, I_i, O_i, C_i, root, D_i, UpB_i, \Sigma_i, \mathcal{L}_i)$.

Let $\mathbb{A} = \bigcup_{i=1}^{m} \{A_i\}$ and $A_i = (L_i, \ell_{0_i}, X_i, T_i, Inv_i, G_i, B_i, Z_i, \mathbb{T}_i, \mathbb{F}_i, Syn_i, \Omega_i)$.

$(\mathbb{A}, \mathbb{C}, \mathbb{V})$ is the timed automata system obtained by applying our transformation $\mathcal{T}$, that will be detailed in the sequel, to $(\mathbb{S}, \mathbb{E})$. But first, let us define some mappings between elements of state machines and of timed automata.

- *NtoL*: $\left(\bigcup_{i=1}^{n} N_i \right) \longrightarrow \left(\bigcup_{i=1}^{m} L_i \right)$ be a mapping which associates each state machine node with a timed automaton location,

Semantics 2 From a State Machine System to a Timed Automata System

$$
\begin{array}{l}
(\mathbb{A},\mathbb{C},\mathbb{V}) = \mathcal{T}(\ (\mathbb{S},\mathbb{E})\ ) \Leftrightarrow \\
(\forall s_i \in \mathbb{S})(\forall r \in R_i)(\exists A_r^i \in \mathbb{A}) \\
\left(\begin{array}{l}
A_r^i = (L_r^i, \ell_{0_r}^i, X_r^i, T_r^i, G_r^i, B_r^i, Inv_r^i, Z_r^i, \mathbb{T}_r^i, \mathbb{F}_r^i, Syn_r^i, \Omega_r^i) \quad \wedge \qquad (1)\\
L_r^i = \{\ell \mid (\exists x \in N_i)(r \to x \wedge NtoL(x) = \ell)\} \cup \{RtoL(r)\} \quad \wedge \\
X_r^i = \{c \mid (\exists x \in N_i)(r \to x \wedge NtoX(x) = c)\} \quad \wedge \qquad (2)\\
(\exists n \in \{x|r \to x\})(n \in \Delta_i) \Leftrightarrow \ell_{0_r}^i = NtoL(n) \quad \wedge \\
(\forall n \in \{x|r \to x\})(n \notin \Delta_i) \Leftrightarrow \ell_{0_r}^i = RtoL(r) \quad \wedge \qquad (3)\\
(\forall n \in \{x|r \to x\})(\exists (u_b,u_s) \in \mathbb{N} \times \mathbb{B} \cup \{\infty\} \times \top) \\
\left(\begin{array}{l}
UpB_i(n) = (u_b,u_s) \Rightarrow \\
\left(\begin{array}{lll}
u_s = \top & \Rightarrow Inv_r^i(NtoL(n)) = NtoX(n) < u_b & \wedge \\
u_s = \bot & \Rightarrow Inv_r^i(NtoL(n)) = NtoX(n) \leqslant u_b &
\end{array}\right)
\end{array}\right) \quad \wedge \qquad (4)\\
T_r^i = \\
\left\{\begin{array}{l}
\begin{array}{llll}
\tau = toL(x) \xrightarrow{\alpha} toL(y) & | & & \\
tr \in Tr_i & \wedge & tr = (src, in, c, out, tgt) & \wedge \\
s = {}^{\bullet}\mathcal{I}_{src}^{tgt} \cap \{n|r \to n\} & \wedge & t = \mathcal{I}^{\bullet}{}_{src}^{tgt} \cap \{n|r \to n\} & \wedge \\
s = \varnothing \Rightarrow x = r & \wedge & s \neq \varnothing \Rightarrow x \in s & \wedge \\
t = \varnothing \Rightarrow y = r & \wedge & t \neq \varnothing \Rightarrow y \in t & \wedge \\
t \neq \varnothing \Rightarrow Z_r^i(\tau) = \{NtoX(y)\} & \wedge & \neg(x = y = r) & \wedge
\end{array}\\
x \neq src \Rightarrow \\
\left(\begin{array}{ll}
\alpha = \epsilon & \wedge \\
G_r^i(\tau) = C_r^i(\tau) = \mathbb{T}_r^i(\tau) = \mathbb{F}_r^i(\tau) = \varnothing & \wedge \\
Syn_r^i(\tau) = (lbl(tr), ?) &
\end{array}\right) \quad \wedge \qquad (5)\\
x = src \Rightarrow (\exists (l_b, l_s, u_b, u_s) \in \mathbb{N} \times \mathbb{B} \times \{\mathbb{N} \times \mathbb{B} \cup \{\infty\} \times \top\}) \\
\left(\begin{array}{l}
\begin{array}{llll}
\alpha = \mathcal{L}_i(tr) & \wedge & Syn_r^i(\tau) = (lbl(tr), !) & \wedge \\
B_r^i(\tau) = \mathbb{F}_r^i(\tau) = in & \wedge & \mathbb{T}_r^i(\tau) = out & \wedge \qquad (6)
\end{array}\\
c = (l_s, l_b, u_b, u_s) \Rightarrow (\exists \lambda)(\exists \mu) \\
\left(\begin{array}{lllll}
l_s & = \top & \Rightarrow & \lambda = NtoX(n) > l_b & \wedge \\
l_s & = \bot & \Rightarrow & \lambda = NtoX(n) \geqslant l_b & \wedge \\
u_s & = \top & \Rightarrow & \mu = NtoX(n) < u_b & \wedge \\
u_s & = \bot & \Rightarrow & \mu = NtoX(n) \leqslant u_b & \wedge \\
G_r^i(\tau) = \{\lambda, \mu\} & & & &
\end{array}\right) \qquad (7)
\end{array}\right)
\end{array}\right\} \quad \wedge \\
\Omega_r^i = \{a \mid t \in T_r^i \wedge t = \ell \xrightarrow{a} \ell'\} \qquad (8)
\end{array}\right)
\end{array}
$$

- *RtoL*: $\left(\bigcup_{i=1}^{n} R_i\right) \longrightarrow \left(\bigcup_{i=1}^{m} L_i\right)$ be a mapping associating each state machine region with a timed automaton location,
- $toL = NtoL \cup RtoL$ is a bijective mapping between $\left(\bigcup_{i=1}^{n} N_i\right) \cup \left(\bigcup_{i=1}^{n} R_i\right)$ and $\bigcup_{i=1}^{m} L_i$.
- $lbl : \left(\bigcup_{i=1}^{n} Tr_i\right) \longrightarrow \mathbb{C}$ is a bijective mapping assigning a channel to each transition of the state machine.
- *NtoX*: $\left(\bigcup_{i=1}^{n} N_i\right) \longrightarrow \left(\bigcup_{i=1}^{m} X_i\right)$ is a bijective mapping associating each state machine node to a clock of the Timed Automata System.

Moreover, we will use an implicit bijective mapping between regions and automata. This means that a timed automaton A_r^i will give the behavior of the region $r \in R_i$ of the state machine s_i.

Definition 25. *A state machine system* $(\mathbb{S},\mathbb{E})$ *is transformed into the semantically equivalent* $(\mathbb{A},\mathbb{C},\mathbb{V})$*, by using the previous mappings, according to transformation* $\mathcal{T}$*, that is defined by Semantics 2*

The transformation is done one region at a time: each region $r \in R_i$ will be translated to an automaton A_r^i (cf. condition (1) in Semantics 2). As one can guess from the mappings' definitions, the set of locations is given in two parts: each node of a state machine will correspond to a location, but we will also use an extra location for each region, which, when active, will indicate that all the active nodes are outside the region, *i.e.*, no node of the region is actually active.

We can see this mechanism in more detail for a given location set L_r^i. We see that each location set L_r^i is made of the location $RtoL(r)$ which is associated with the region r, as well as the set of locations that are associated with the nodes x which are the direct sons of r ($r \to x$). The same mechanism is applied to give the clock set

X_r^i: each direct sub-node of the region r is mapped to a clock $NtoX(x)$ (cf. condition (2) in Semantics 2).

The next step is to specify which state machine element, the initial location $\ell_{0_r}^i$ is associated with. This is done using a simple pattern $(C \Leftrightarrow A) \wedge (\neg C \Leftrightarrow B)$ (cf. condition (3) in Semantics 2).

On the one hand, if there exists a node n in the current region r such that n belongs to the initial nodal configuration Δ_i, then the initial location $\ell_{0_r}^i$ is associated with this node by the mapping $NtoL$ (and conversely). Because $n \in \Delta_i$, this means that node n is initially activated in the state machine s_i, and thus, the corresponding location $NtoL(n)$ has also to be initially activated in automaton A_r^i. On the other hand, if no node belongs to the initial nodal configuration in the current region r, then the initial location is the one associated with the region r by the mapping $RtoL$ (and conversely). The location $RtoL(r)$ is a special location specifying that no node of the region is active, which is precisely the case initially when no node of the region belongs to the initial nodal configuration Δ_i.

The next clause concerns the translation of the upper bound delay constraint of every node n that belongs to r. An upper bound delay serves as a validity time interval: it defines how long a node can be continuously activated. When the upper bound delay is reached, time cannot elapse any more, unless the node is deactivated. The upper bound $UpB_i(n)$ of node n, is given by a pair (u_b, u_s) in which u_b (possible infinite) specifies the amount of time limit by which the node must be deactivated, and u_s defines whether the value u_b is comprised in the validity time interval. If $u_s = \top$, meaning that u_s is true, then the time interval is *strict* and u_b is *not* comprised in the validity interval (cf. condition (4) in Semantics 2).

We then arrive to the major part: the specification of the set of timed automata transitions (cf. conditions (5) in Semantics 2). The set T_r^i is composed of two parts: the first one will actually create the timed automata transitions, whereas the second part will fill all the mappings whose domain is T_r^i. It may seem awkward to include all that information in the conditions for the creation of the set T_r^i, but this is actually the better place for it to be defined, as in there, we know that the translation link between timed automata and state machine transitions (*i.e.*, τ translates tr).

The transition set T_r^i is then the set of transitions $\tau = toL(x) \xrightarrow{\alpha} toL(y)$. This expression tells us much information: α is an action label, $toL(x)$ and $toL(y)$ are two automaton locations, and x and y may either be nodes or regions in the source state machine s_j. Let us see what are the conditions on x and y for them to be used as generators of transitions τ. First, we pick a transition tr from the state machine transition set Tr_i. A state machine transition is in fact a 5-tuple $tr = (src, in, c, out, tgt)$, where src is the source node, tgt the target node, in is the set of events that must have been — somehow — previously generated, since all events in in are consumed as tr triggers; c is the time interval in which tr can fire, relatively to the last activation of src; and finally out is the set of events generated by the firing of tr.

We then create two sets s and t, which respectively hold the source impact (${}^\bullet\mathcal{I}_{src}^{tgt}$) and the target impact ($\mathcal{I}^\bullet{}_{src}^{tgt}$) of tr, restricted to the son nodes of the region r (*i.e.*, intersected with the set $\{n | r \rightarrow n\}$).

When s (resp. t) is empty, this means that the source node src (resp. target node tgt) of tr is orthogonal to the elements of s (resp. t). This means that tr is actually entering (resp. leaving) region r. The fact that the source (resp. target) node is outside the region r is specified by setting x (resp. y) to r, as the mapping $toL(r)$ gives a special location which does not correspond to any node in r, and thus can be used to specify that no node of r is deactivated (resp. activated). If both s and t are empty, then tr has no effect on the region r, which means that there can be no transition τ translating tr in the region r, thus the condition $\neg(x = y = r)$.

When s (resp. t) is not empty, it holds the nodes of region r which *could* be impacted by the firing of tr, provided they are active when tr is triggering. In fact, s (resp. t) is inevitably a singleton when considering a region *above* src (resp. tgt), and may only be a bigger set when r is lower in the hierarchy than src (resp. tgt).

This concludes the first part concerning how τ is generated. Then comes the part regarding the mappings using T_r^i as a domain: G_r^i, C_r^i, $\mathbb{T}_r^i$, $\mathbb{F}_r^i$, and Syn_r^i.

In the timed automata obtained using translation $\mathcal{T}$, a transition is either a master or a slave transition (which means that there will be no transition t such that $sync(t) = \varnothing$), according to which synchronization label it holds. In our translation, all the information regarding the source SM transition will be held by master transitions, and master transitions will only be the timed automata transitions whose source location corresponds to the source node of the SM transition it translates, *i.e.*, a timed automaton transition is a master one if $x = src$.

If $x \neq src$, *i.e.*, if tr is translated into a slave timed automaton transition, then obviously, $Syn_r^i(\tau) = (lbl(tr), ?)$: we put the slave synchronization label $(lbl(tr), ?)$, where $lbl(tr)$ gives the channel which will be used to simulate the triggering of tr. The action label α is set to the silent action ϵ, and τ bears no clock guard, no variable guard, no setting to true and no setting to false. In fact, the only effect of τ is to pass the activation token from $toL(x)$ to $toL(y)$.

If $x = src$, i.e, if τ is a master transition, then its action label is copied from that of tr, $\mathcal{L}_i(tr)$. The master synchronization label $(lbl(tr), !)$ is associated with τ, which means that when τ happens, it forces every enabled slave transition using the same channel $lbl(tr)$ to synchronously and atomically trigger with it. As a requirement

to trigger the SM transition tr, it must respect its event guard and clock guard. Let us keep clocks aside for now.

Event guards and manipulations are performed as follows: the event configuration, *i.e.*, the set of present events, must contain every event in in, which are required to be consumed over the triggering of tr. This behavior is translated by setting $C_r^i(\tau)$ and $\mathbb{F}_r^i$ to in. Indeed, $C_r^i(\tau) = in$ means that we require every Boolean variables in in to be true. Moreover, $\mathbb{F}_r^i = in$, means that all the variables in in are set to false (or, more precisely, are removed from the set of true variables, as they are "consumed"). Finally, the set out of events that must be generated when tr triggers, is simply translated as the set of variables to be set to true on the triggering of τ: $\mathbb{T}_r^i(\tau) = out$ (cf. condition (6) in Semantics 2).

Finally, the clock guard c specifies the interval of time instants at which the transition can trigger. An interval is translated into two timed automata clock guards: one specifying the lower bound and the other specifying the upper bound. The clock guard $c = (l_s, l_b, u_b, u_s)$ is decomposable into its lower bound (l_s, l_b) and its upper bound (u_b, u_s), which will respectively be converted into two clock guards λ and μ. The lower (resp. upper) bound clock guard λ (resp. μ) is then equal to the equation $NtoX(n) > l_b$ or to $NtoX(n) \geqslant l_b$ (resp. $NtoX(n) < u_b$ or $NtoX(n) \leqslant u_b$) depending on the strictness given by l_s (resp. u_s). We finally put both clock guards λ and μ inside a common set which will stand for the clock guard of τ, $G_r^i(\tau)$ (cf. condition (7) in Semantics 2). .

The action alphabet Ω_r^i of automaton A_r^i is the set of action labels used by the transitions in T_r^i, *i.e.*, the transitions of the timed automaton A_r^i (cf. condition (8) in Semantics 2). In this form, the definition hides a few characteristics. As we will see, the transitions in T_r^i either hold an *observable* label or the *silent* action ϵ. However, only the master transitions are labelled by an observable action (but they still *can* be labelled using ϵ). This sort of transition is also special because bearing an observable action means that the state machine transition they translate has its source node directly inside the region r. As a last remark, Ω_r^i is a subset of Σ_i.

It is worth noting that an alternative way to translating SM systems into TA might be to first translate the state machine system to Hierarchical Timed Automata (HTA), then to use flattening on HTA [11]. However, this would have required to fit with the HTA semantics defined and to perform flattening afterwards. Here, we rather chose to carry out a complete translation process while proving a strong equivalence relation between the source and target models, as will be proven in the following section.

6 BISIMULATION PRESERVATION PROOF OF THE TRANSFORMATION

There exist many mathematical means which can be used to compare two formalisms, or more basically two graph-based structures. One example is isomorphism, which requires a very strong correspondence between two graphs, as their structure must be exactly the same: there must exist a bijective mapping between states and transitions respectively. A useful intuition behind isomorphism, although not quite exact, is that the structure must be the same except for their naming.

One major drawback of isomorphism is that it is often too strong when considering problems like implementation of a specification. Programmers know that there are (infinitely) many ways to implement a given problem/specification. However, if a programmer would have to compare two implementations only with isomorphism, then he/she would be forced to declare that an implementation is different as soon as it does not exactly possess the same internal representation, or at least as soon as a *bijection* cannot be provided between the internal representations.

Another equivalence was then needed to make comparisons more useful: this is where the notion of *simulation* comes into play. With simulation, we make a partition of states and transitions of the source structure, and if a mapping exists between the elements from the partition and the target structure, then we can say that the target structure *simulates* the source one.

But simulation suffers from the opposite drawback to isomorphism: when considering a mapping, this means that the target structure may comprise unwanted behavior, *i.e.*, states and transitions which have no correspondence with the source structure.

One way to get the best of both worlds is to apply simulation in both directions. We then have a *bisimulation*. A bisimulation stipulates that there exists a bijective mapping between partitions of states and transitions of both compared structures. With bisimilar transition systems, we get the same apparent effect, but in many different ways. The accepted differences between them can be defined as the equivalence classes of the state/transition partitions.

Definition 26. *Let $S_1 = (Q_1, q_1^0, \rightarrow_1, \Sigma_1)$ and $S_2 = (Q_2, q_2^0, \rightarrow_2, \Sigma_2)$ be two timed transition systems and let $\sim \subseteq (Q_1 \times Q_2)$ be an equivalence relation. We say that S_1 and S_2 are strongly time bisimilar iff:*

- $q_1^0 \sim q_2^0$ *and*

- $(\forall a \in \Sigma_1 \cup \Sigma_2 \cup \mathbb{R}^+)$
$\left(\begin{array}{ll} q_1 \xrightarrow{a}_1 q_1' \wedge q_1 \sim q_2 \Rightarrow (\exists q_2')(q_2 \xrightarrow{a}_2 q_2' \wedge q_1' \sim q_2') & \wedge \\ q_2 \xrightarrow{a}_2 q_2' \wedge q_1 \sim q_2 \Rightarrow (\exists q_1')(q_1 \xrightarrow{a}_1 q_1' \wedge q_1' \sim q_2') & \end{array} \right)$

Theorem 1. *Let $(\mathbb{S}, \mathbb{E})$ be a state machine system and $(\mathbb{A}, \mathbb{C}, \mathbb{V})$ a timed automata system. If $\mathcal{T}((\mathbb{S}, \mathbb{E})) = (\mathbb{A}, \mathbb{C}, \mathbb{V})$, then their semantics are strongly time bisimilar.*

Proof

For two semantics (*i.e.*, Timed Transition Systems) to be strongly time bisimilar, there must exist at least one relation R between their respective states, and that relation must be a strong time bisimulation. Our first step will then be to define this relation in a general way, *i.e.*, in a way that can apply on any two systems which are related by a given translation relation $\mathcal{T}$.

For our next step, we will need to prove that the initial states of both semantics are related by R (induction base case). Then (induction step) that for each pair in R, one can fire exactly the same set of transitions, and that the target states also form a pair in R. As always, transitions will be separated according to their discreteness.

First, we need some definitions, mainly that of R. Let $(\mathbb{S}, \mathbb{E})$ be a state machine system and $(\mathbb{A}, \mathbb{C}, \mathbb{V})$ a timed automata system, where:

- $\mathcal{T}(\ (\mathbb{S}, \mathbb{E})\ ) = (\mathbb{A}, \mathbb{C}, \mathbb{V})$, i.e $(\mathbb{S}, \mathbb{E})$ is translated into $(\mathbb{A}, \mathbb{C}, \mathbb{V})$,
- $\mathbb{S} = \bigcup_{i=1}^{n} \{s_i\}$ and $s_i = (N_i, R_i, H_i, Tr_i, I_i, O_i, C_i, root_i, D_i, UpB_i, \Sigma_i, \mathcal{L}_i)$,
- $\mathbb{A} = \bigcup_{i=1}^{m} \{A_r^i\}$ and $A_r^i = (L_r^i, \ell_{0_r}^i, X_r^i, T_r^i, Inv_r^i, G_r^i, B_r^i, Z_r^i, \mathbb{T}_r^i, \mathbb{F}_r^i, Syn_r^i, \Omega_r^i)$, where i serves as an ordering (index) of automaton A^i and r is the region of $(\mathbb{S}, \mathbb{E})$ that A^i implements (we changed the notation a bit: previously, i was the index of the state machine s_i, such that r was a region of R_i).

Let $T^{\mathbb{S}} = (S, s_0, \rightarrow_{sm}, \Sigma_{sm})$ and $T^{\mathbb{A}} = (Q, q_0, \rightarrow_{at}, \Sigma_{at})$ be the respective (TTS) semantics of $(\mathbb{S}, \mathbb{E})$ and $(\mathbb{A}, \mathbb{C}, \mathbb{V})$. Let $R \subseteq S \times Q$ be the relation defined as follows:

$$(s, q) \in R \Leftrightarrow \left(\begin{array}{ll} s = (\mathcal{N}, \gamma, \lambda) & \wedge \\ q = (\mathcal{L}, \gamma', \lambda') & \wedge \\ \mathcal{N} \equiv \mathcal{L} & \wedge \\ \gamma \equiv \gamma' & \wedge \\ \lambda = \lambda' & \end{array} \right)$$

where $\mathcal{N} \equiv \mathcal{L} \Leftrightarrow (\forall s_i \in \mathbb{S})(\forall n \in \mathcal{N}(i))(\forall A_r^k \in \mathbb{A}) \left(\begin{array}{ll} (\forall l \in L_r^k)(NtoL(n) = l \Rightarrow \mathcal{L}(k) = l) & \wedge \\ (\forall l \in L_r^k)(NtoL(n) \neq l) \Rightarrow \mathcal{L}(k) = RtoL(r) & \end{array} \right)$

and $\gamma \equiv \gamma' \Leftrightarrow (\forall s_i \in \mathbb{S})(\forall n_a \in N_i)(\forall A_r^k \in \mathbb{A})(\forall x_b \in X_r^k)(NtoX(n_a) = x_b \Rightarrow \gamma(i)(a) = \gamma'(k)(b))$

The definition of R (obviously) comes from the definition of the transformation $\mathcal{T}$. It states that two states $s \in S$ and $q \in Q$ are linked by R, meaning that they are to be considered "bisimilarly equivalent". Let us review how this is done:

- each active node $n \in \mathcal{N}(i)$ of a state machine system is translated to a unique location $l \in L_r^k$ according to the mapping $NtoL$, and each of these locations are active in their corresponding automata A_r^k: $\mathcal{L}(k) = l$. Notice that $\mathcal{N}(i)$ is a set, while $\mathcal{L}(k)$ is a single location. Indeed, while each state machine can have multiple nodes activated simultaneously, only one location can be activated within a single automaton. From the point of view of the timed automata system, the set of locations which are the translation of a node, bijectively maps the set of active nodes $\mathcal{N}(i)$.
- whenever no location in L_r^k of an automaton A_r^k corresponds to an active node (*i.e.*, a node in $\mathcal{N}(i)$), then the active location $\mathcal{L}(k)$ of A_r^k is given by $RtoL(r)$, where r is the region that the automaton translates.

The part describing the correspondence between γ and γ' is straightforward: all the clocks associated with the nodes are always active, and so are the translated clocks of the timed automata. The mapping between the two is accomplished through $NtoX$: if the clock of node n_a is translated into a clock x_b using mapping $NtoX$ then the node clock node $\gamma(i)(a)$ must be equal to the clock value $\gamma'(k)(b)$.

The event part $\lambda = \lambda'$ is even easier, as we use the same representation, we will then consider the two as equal. Indeed, both λ and λ' hold the set of present events.

Proof Scheme

To prove that the relation R is a bisimulation, we will proceed by induction:

- the base case will prove that the pair of initial states is in R, *i.e.*, $(s_0, q_0) \in R$. This implies that $(q_0, s_0) \in R^{-1}$ and thus $\{(s_0, q_0), (q_0, s_0)\} \subseteq R$;
- the first induction step will prove that any step in a state machine has an equivalent step in a timed automaton, *i.e.*, given $(s, q) \in R$ denoted by sRq and $s \xrightarrow{a \in \Sigma \cup \mathbb{R}_+^*}_{sm} s'$ we have $q \xrightarrow{a}_{at} q'$ and $s'Rq'$;
- the reverse induction step will prove that any timed automaton step has its equivalent step in a state machine, *i.e.*, given sRq and $q \xrightarrow{a \in \Sigma \cup \mathbb{R}_+^*}_{at} q'$ we have $s \xrightarrow{a}_{sm} s'$ and $s'Rq'$.

Base Induction Case

Let $s_0 = (C_0, \gamma_0, \varnothing)$ and $q_0 = (\mathcal{L}_0, h_0, \varnothing)$.

• We have to prove that the two initial states form a bisimilar pair, *i.e.*, $(s_0, q_0) \in R$, or, more shortly, $s_0 R q_0$.

The initial nodal configuration C_0, is such that $(\forall s_i \in \mathbb{S})(C_0(i) = \Delta_i)$. From the translation, we have:

$(\forall s_i \in \mathbb{S})(\forall r \in R_i)(\exists A_r^j \in \mathbb{A})(\exists n \in \{x | r \to x\})((n \in \Delta_i) \Leftrightarrow \ell_{0r}^{\ j} = NtoL(n))$,

which can rewritten as:

$(\forall s_i \in \mathbb{S})(\forall n \in \Delta_i)(\forall r \in R_i)(\exists A_r^j \in \mathbb{A})(r \to n \Leftrightarrow \ell_{0r}^{\ j} = NtoL(n))$.

This means that for any initial node n, there exists one and only one corresponding *initial* location in automaton A_r^i. The reverse is not true, because we have some regions that contain no initial nodes, for that case the translation states the following:

$(\forall s_i \in \mathbb{S})(\forall r \in R_i)(\exists A_r^j \in \mathbb{A})((\forall n \in \{x | r \to x\})(n \notin \Delta_i) \Leftrightarrow \ell_{0r}^{\ j} = RtoL(r))$.

This means that whenever a region has no initial node, then the translation of this region is the initial location of the corresponding automaton.

Both cases state that for each automaton, the initial location either comes from an initially active node, or from an initially inactive region (*i.e.*, all the nodes of the region are initially inactive); which is exactly the definition of bisimilar equivalence of nodes and locations, then $C_0 \equiv \mathcal{L}_0$.

Every clock is initially set to zero in both semantics, therefore:

$(\forall s_i \in \mathbb{S})(\forall n_a \in N_i)(\forall A_r^j \in \mathbb{A})(\forall x_b \in X_r^j)(NtoX(n) = x_b \Rightarrow \gamma_0(i)(a) = h_0(j)(b) = 0)$. And then $\gamma_0 \equiv h_0$.

In addition, $\lambda_0 = \lambda_0' = \varnothing$ and finally, $s_0 R q_0$.

6.1 Induction Step

Hypothesis: we suppose that up to a certain point, we have sRq and that $s \xrightarrow{a}_{sm} s'$. We need to prove that $q \xrightarrow{a}_{at} q'$ and $s'Rq'$.

• First, let us suppose that $a \in \Sigma$, *i.e.*, that $s \xrightarrow{a} s'$ is a discrete transition. Moreover, let us suppose that $s = (\mathcal{N}, \gamma, \lambda)$, $s' = (\mathcal{N}', \gamma', \lambda')$, $q = (\mathcal{L}, h, \mathcal{V})$ and $q' = (\mathcal{L}', h', \mathcal{V}')$.

From the semantics of state machines, we know that there exists a transition $e = (n, in, c, out, n')$ in a state machine $s_i \in \mathbb{S}$, such that (let us just concentrate on the node point of view for the moment):

- the source node of e belongs to the current local nodal configuration of s_i, *i.e.*, $n \in \mathcal{N}(i)$
- the target node of e belongs to the next local nodal configuration of s_i: $n' \in \mathcal{N}'(i)$.

The nodal configuration $\mathcal{N}$ will be changed into $\mathcal{N}'$ according to the source and target impact of e. We know that e is translated into at least one master transition t (only master transitions bear an action label, here a). Our objectives are to prove that:

- triggering t is possible w.r.t the nodal part
- triggering t does not break bisimilarity w.r.t the nodal part.

6.1.1 Triggering t is possible w.r.t. the nodal part.

We know that t must exist, because the source state n of e has to be in the source impact of e. We also know that each region corresponds to an automaton which yields its behavior. Let us suppose that n belongs to r, *i.e.*, $r \to n$. This means that automaton $A_r^j \in \mathbb{A}$, the one that gives the behavior of region r, necessarily has a location ℓ which is the translation of n, *i.e.*, $NtoL(n) = \ell$. From the translation, we also know that there exists a transition $\tau = toL(x) \xrightarrow{a} toL(y)$ for which $x = n$, and this transition is a master one ($x = n \Rightarrow Syn_r^i(\tau) = (lbl(e), !)$). τ is then the transition t we are looking for. As sRq and because $n \in \mathcal{N}(i)$ then $\mathcal{L}(j) = NtoL(n) = \ell$. When only considering the nodal configuration, t is then allowed to trigger; or stated differently, if t is not able to be taken, it is because of another type of constraint.

6.1.2 Triggering t does not break bisimilarity w.r.t. the nodal part.

Put more formally, we have to prove $s'Rq'$. We already know how s' is built: $s' = (s \backslash {}^\bullet\mathcal{I}_n^{n'}) \cup \mathcal{I}^{\bullet n'}_n$. The step from s to s' is done by a single transition e on the state machine side, while it is accomplished by one transition in each of the impacted automata, *i.e.*, the automata translating the regions having nodes in ${}^\bullet\mathcal{I}_n^{n'}$ or $\mathcal{I}^{\bullet n'}_n$. We have to examine each transition individually, which means that we have to examine the source and target impact for each region. In the following cases, we suppose that region r is translated into automaton A_r^j and we denote by $\rho = \{n | r \rightarrow n\}$ the set of nodes directly in r.

If ${}^\bullet\mathcal{I}_n^{n'} \cap \mathcal{N}(i) \cap \rho = \{a\}$ and $\mathcal{I}^{\bullet n'}_n \cap \rho = \{b\}$, we have: $a \in \mathcal{N}(i)$ and $b \in \mathcal{N}'(i)$ and $\mathcal{L}(j) = NtoL(a)$ and $\mathcal{L}'(j) = NtoL(b)$, which locally respects bisimilarity.

If ${}^\bullet\mathcal{I}_n^{n'} \cap \mathcal{N}(i) \cap \rho = \{a\}$ and $\mathcal{I}^{\bullet n'}_n = \varnothing$, we have $a \in \mathcal{N}(i)$ and $(\forall x \in N_i)(r \rightarrow x \Rightarrow x \notin \mathcal{N}'(i))$ and $\mathcal{L}(j) = NtoL(a)$ and $\mathcal{L}'(j) = RtoL(r)$, which locally respects bisimilarity.

If ${}^\bullet\mathcal{I}_n^{n'} \cap \mathcal{N}(i) \cap \rho = \varnothing$ and $\mathcal{I}^{\bullet n'}_n \cap \rho = \{b\}$, we have $(\forall x \in N_i)(r \rightarrow x \Rightarrow x \notin \mathcal{N}(i))$ and $b \in \mathcal{N}'(i)$ and $\mathcal{L}(j) = RtoL(r)$ and $\mathcal{L}'(j) = NtoL(b)$, which locally respects bisimilarity.

Each of the previous cases represents a transition in automaton A_r^j, and we saw that in each case the firing of that transition locally preserves bisimilarity.

The regions/nodes not impacted by the triggering of e remain unchanged, and the transitions using the same synchronization label $lbl(e)$ only come from the previous case, then $\mathcal{N}' \equiv \mathcal{L}'$.

On the automaton side, only master transitions hold the clock/event constraints. We will now concentrate on them and forget about slave transitions.

6.1.3 Events

The transition $e = (n, in, c, out, n')$ holds two event information:

- the set in tests whether the events in it were previously generated (and not yet consumed); if it is the case and that e does trigger, then the in events are removed from the event configuration λ.
- the set out gives the events to be generated (*i.e.*, to be added to the event configuration) upon the triggering of e.

This can be shortly and formally written as: $in \subseteq \lambda \wedge \lambda' = (\lambda \backslash in) \cup out$.

We know that sRq, then $\lambda = \mathcal{V}$.

We know from the translation that $C_r^j(t) = in$. From the timed automata semantics, t respects its constraints w.r.t. in events, when $C_r^j(t) \subseteq \mathcal{V}$. The transition e can fire, meaning that $in \subseteq \lambda$, and then $C_r^j(t) \subseteq \mathcal{V}$ is also true (and t is not prevented to fire by the event configuration).

We also know, from the timed automata semantics, that $\mathcal{V}' = (\mathcal{V} \backslash \mathbb{F}_r^j(t)) \cup \mathbb{T}_r^j(t)$ when $t \in T_r^j$ is a master transition of A_r^j. And finally we know, from the translation, that $\mathbb{F}_r^j(t) = in$ and $\mathbb{T}_r^j(t) = out$. This means that $\mathcal{V}' = \lambda'$.

6.1.4 Clocks

Firing $e = (n_k, in, c, out, n'_l)$ means that it respects its clock constraint c. This clock constraint c is a 4-tuple: $c = (l_s, l_b, u_b, u_s)$, where l_b and u_b are respectively the lower and upper bounds, and l_s and u_s respectively state whether these bounds are excluded ($\top$ if excluded, $\bot$ otherwise). To make the subsequent proof more concise, we will use the two following notations: $\underline{\bot} = \text{“} > \text{”}$, $\underline{\top} = \text{“} \geqslant \text{”}$, $\overline{\bot} = \text{“} < \text{”}$ and $\overline{\top} = \text{“} \leqslant \text{”}$, that is we convert the strictness flag into its corresponding comparison operator.

Let us suppose that the clock associated with n is the k^{th} component of the clock vector of system s_i. In order to fulfil constraint c, clock configuration γ must respect: $\gamma(i)(k)\ \underline{l_s}\ l_b$ and $\gamma(i)(k)\ \overline{u_s}\ u_b$.

Because sRq, we have $NtoX(n_k) = x_m \Rightarrow \gamma(i)(k) = h(j)(m)$. This means that $h(j)(m)\ \underline{l_s}\ l_b$ and $h(j)(m)\ \overline{u_s}\ u_b$.

From the translation, constraint c is translated to set $G_r^j(t) = \{NtoX(n_k)\ \underline{l_s}\ l_b, NtoX(n_k)\ \overline{u_s}\ u_b\}$, which is exactly what is respected by the current clock value $h(j)(m)$.

Consequently, the firing of t is not prevented by its guard $G_r^j(t)$.

The only manipulation done to state machine clocks is to reset those whose node is in the target impact: $(\forall n_a \in \mathcal{I}^{\bullet n'_l}_{n_k})(\gamma'(i)(a) = 0)$.

The translation specifies that any transition leading to a location translating a node n has to reset the associated clock $NtoX(n)$. Formally, $t = \mathcal{I}^{\bullet tgt}_{src} \cap \{n | r \rightarrow n\}$, and $t \neq \varnothing \Rightarrow Z_r^i(\tau) = \{NtoX(y)\}$, where $y \in t$. And every node $n_a \in \mathcal{I}^{\bullet n'_l}_{n_k}$, where $r \rightarrow n_a$, produces a transition in automaton A_r^j. Indeed, in that case $\mathcal{I}^{\bullet n'_l}_{n_k} \cap \{n | r \rightarrow n\} \neq \varnothing$, which is sufficient to have a transition in A_r^j, as it respects $y \neq r$ (c.f. “$\neg(x = y = r)$” in the translation).

This means $(\forall n_a \in \mathcal{I}^{\bullet n'_l}_{n_k})(\forall A_r^j \in \mathbb{A})(r \rightarrow n_a \wedge NtoX(n_a) = x_b \Rightarrow \gamma'(i)(a) = 0 = h'(j)(b))$. This means that this set of node/clock pairs respects bisimilarity in the new configuration. The other nodes / clocks which do not

respect $r \rightarrow n_a \wedge NtoX(n_a) = x_b$ are kept unchanged, and because they were respecting bisimilarity: sRq, they still do. We can then conclude: $\gamma' \equiv h'$.

6.1.5 Discrete step equivalence

Let $s = (\mathcal{N}, \gamma, \lambda)$, $s' = (\mathcal{N}', \gamma', \lambda')$, $q = (\mathcal{L}, h, \mathcal{V})$ and $q' = (\mathcal{L}', h', \mathcal{V}')$. We proved that, if $sRq \wedge s \xrightarrow{a} q$ then $q \xrightarrow{a} q'$ and $\mathcal{N}' \equiv \mathcal{L}'$ and $\gamma' = h'$ and $\lambda' = \mathcal{V}'$. This means that $s'Rq'$, which concludes this part of the proof.

6.1.6 Continuous Transitions: Satisfaction of the upper bound constraint

We suppose here that $s \xrightarrow{\theta \in \mathbb{R}_+^*} s' \wedge sRq$. We need to prove that $q \xrightarrow{\theta \in \mathbb{R}_+^*} q' \wedge s'Rq'$. Again, let $s = (\mathcal{N}, \gamma, \lambda)$, $s' = (\mathcal{N}', \gamma', \lambda')$, $q = (\mathcal{L}, h, \mathcal{V})$ and $q' = (\mathcal{L}', h', \mathcal{V}')$.

The only way to change the nodal or event (resp. location or variable) configuration is to trigger a transition. But no transition triggering can result in a time lapse at the TTS level. This means that no transition triggered and consequently $\mathcal{N} = \mathcal{N}'$ and $\lambda = \lambda'$ (resp. $\mathcal{L} = \mathcal{L}'$ and $\mathcal{V} = \mathcal{V}'$).

The only mechanism that can constrain the advance of time are the upper bound delays which are assigned to nodes on the state machine side, and the location invariants on timed automata side.

Being allowed to let the time elapse for an amount of θ units of time, means we have: $(\forall s_i \in \mathbb{S})(\forall n_k \in \mathcal{N}(i))(UpB_i(n_k) = (u_s, u_b) \Rightarrow \gamma(i)(k) + \theta \ \overline{u_s} \ ub)$.

Because sRq, we have $(\forall s_i \in \mathbb{S})(\forall n_k \in \mathcal{N}_i)(\forall A_r^j \in \mathbb{A})(\forall x_l \in X_r^j)(NtoX(n_k) = x_l \Rightarrow \gamma(i)(k) + \theta = h(j)(l) + \theta$, which means that $s'Rq'$.

We proved that, if we can let time advance by an amount of θ time units from s and q, then the resulting states are bisimilar. There remains to prove that we can indeed let θ time units elapse from q, *i.e.*, $q \xrightarrow{\theta \in \mathbb{R}_+^*} q'$.

From the translation, we have: $UpB_i(n_k) = (u_b, u_s) \Rightarrow Inv_r^j(NtoL(n_k)) = NtoX(n_k) \ \overline{u_s} \ u_b$, which semantically means $UpB_i(n_k) = (u_b, u_s) \wedge x_l = NtoX(n_k) \Rightarrow h(j)(l) + \theta \ \overline{u_s} \ u_b$. And we already saw that this is true. Then $q \xrightarrow{\theta \in \mathbb{R}_+^*} q'$. And the proof is then complete.

6.2 Reverse simulation

In this section, we will prove the opposite simulation: each transition of $T^{\mathbb{A}}$ has a corresponding (bi-)similar transition in $T^{\mathbb{S}}$. Formally, $(\forall q, q' \in Q)(q \xrightarrow{a} q' \wedge s \sim q \Rightarrow (\exists s' \in S)(s \xrightarrow{a} s' \wedge s' \sim q'))$. We use the same notations as previously.

From the semantics, we know that there must exist a transition t labelled with a which generates the transition $q \xrightarrow{a} q'$ in $T^{\mathbb{A}}$. We need to prove that it is possible to fire a state machine transition e, such that the configuration s' after the firing of e is bisimilar to the configuration q' obtained after the firing of t.

6.2.1 Triggering e is possible w.r.t. the nodal part.

The main point is to observe that each translated transition is either a slave or a master transition. This implies that whenever a transition takes place in $T^{\mathbb{A}}$, the step is triggered by a *master* transition (let us call it t, and let $t = \ell \xrightarrow{a} \ell'$), possibly with many slave transitions. This implies that there exists a transition e in a state machine, with $e = (n_k, in, c, out, n_l)$ and $NtoL(n) = \ell$. Because sRq, the node n is then necessarily active and there exists a state machine s_i so that $n \in \mathcal{N}(i)$. Consequently, e is firable if we only consider the nodal part of the semantics.

6.2.2 Firing & Bisimulation preservation.

Preserving that the resulting configurations of t and e are bisimilar actually uses the same argumentation as for the opposite simulation. We just proved that t corresponds to (or translates) e and there remains to prove that *both* transitions lead to bisimilar configurations. There is no precedence in that latter part, and having t that simulates e instead of the contrary is not relevant. Nodes, clocks and events are all handled by transitions. Then, as soon as we know which timed automaton master transition translates which state machine transition, we can reuse/apply the same arguments. We must add that it is true because timed automaton master transitions and state machine transitions can be bijectively related (to one state machine transition correspond one and only one timed automaton master transition).

6.2.3 Continuous Transitions: Invariants Satisfaction.

As in the previous discrete part, we will be able to reuse the same argument as that of the opposite simulation. Indeed, there exists a clear bijective mapping between nodes and locations ($NtoL$), and nodes and clocks ($NtoX$). This means that to each location ℓ, there corresponds a node n, so that $NtoL(n) = \ell$ (or a region r, so that $RtoL(r) = \ell$, but in that case, the location holds no invariant). As a result, the simulation side has no influence on the rest of the previous argumentation.

7 Conclusion

The main objective of this work was to provide a translation from a subset of UML state machines augmented with some features to timed automata, that was guaranteed to preserve the semantics of the source language. To achieve a sufficient level of confidence about the semantics preservation, it was necessary to give a completely formal abstract syntax and semantics for both formalisms, as well as the translation itself.

The main contribution of the technique discussed in the present paper, consisted in providing a formal semantics for the considered subset of SM, and proving the soundness of the translation w.r.t. a widely recognized equivalence relation: strong timed bisimulation. The proof has been conducted by induction while considering discrete and continuous transitions separately. This formal proof constitutes, indeed, a main feature that differentiates this work from a number of similar existing approaches. From a theoretical point a view, let us recall that carrying out such a proven translation between state machines and timed automata allowed us to highlight the semantic similarity between these models, at least unidirectionally: SM $\longrightarrow$ TA. We should also point out the usefulness of the abstract syntax that we have defined for state machines. This syntax is indeed convenient to outline hierarchy relations in these models. Furthermore, we believe that the genericity offered by the defined syntax makes it possible to deal with other state/transition formalisms. We can also mention that such a genericity is ensured at the cost of a certain abstraction degree. In practice, the work discussed here is fully in line with the increasing willingness of bringing UML diagrams, and more generally model-driven techniques, into play for the design and development of safety critical systems [17]. We believe that, although we considered a restricted subset of UML state machines, the developed contribution sets a sound framework to consider further aspects regarding the transformation of SMs.

We should recall that the primary goal of the present work is to formally prove a strong preservation of the dynamics along the translation we make on the considred subset of UML SMs. This work should be seen as a first step towards a wider formalization of UML diagrams. As perspectives for this work, we want to extend our source model (SM) to take into account additional features that are not considered so far in our translation, such as final states, *History* pseudo-states, entry/exit/do behaviors and forks/joins. We also wish to minimize the number of timed automata clocks generated through our translation [12]. A possible way is to create clocks dynamically as in time Petri Nets, or Stateful Timed CSP [39]. We should also mention that in the current study, we mainly focused on the soundness of the translation and the timed bisimulation preservation. This explains some choices such as, for instance, regarding the syntax statement, to help us derive our proof. Nevetheless, some abstraction should be possible regarding the way the utilized syntaxes are stated. Moreover, to be able to conduct our translation and prove the equivalence, we had to fix several choices in terms of syntax and semantics in both the source and target models. Yet, regarding the possibility to adopt different choices and/or extend the subset of the UML SM specifications to consider, we believe that the present contribution can serve as a relevant basis.

Moreover, after having manually demonstrated the existence of a bisimulation relation between SMs and TA, we are paving the way for validating our translation process by theorem provers. Indeed, by bringing a completely (manual) formal translation, that is entirely based on set theory, we have shown that the translation is perfectly encodable using theorem prover tools. What may prove to be harder, nevertheless, is the handling of bisimulation and its proof. We believe that, if a support to our translation is set by means of a theorem prover, for instance, this would greatly simplify coping with different choices and/or extensions regarding the considered subset of UML SMs.

References

[1] R. Alur and D.L. Dill, "Automata, Languages and Programming", Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90), Pages 322-335, Springer-Verlag New York, Inc. New York, NY, USA, 1990.
[2] R. Alur and D.L. Dill, "A theory of Timed Automata", Theoretical Computer Science Journal, vol. 126, no.2, pp. 183-235, 1994.
[3] Tobias Amnell, Alexandre David, Elena Fersman, M. Oliver M¨oller, Paul Petterson and Wang Yi, Tools for Real-Time UML: Formal Verification and Code Synthesis, SIVOES'2001, pp. 18–22, 2001.
[4] C. André, F. Mallet, R. de Simone, "Modeling time(s)", In Engels, G., Op dyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS, vol. 4735 of Lecture Notes in Computer Science, Springer, pp. 559-573, 2007.
[5] I. Berrada, R. Castanet and P. Félix, "Testing communicating systems: a model, a methodology and a tool", Proceedings of the 17th IFIP International Conference on Testing of Communicating Systems (TESTCOM'05), Lecture Notes in Computer Science, Elsevier, Montreal, Canada, May 2005.
[6] B. Berthomieu, F. Peres and F. Vernadat, "Bridging the Gap Between Timed Automata and Bounded Time Petri Nets", Proceedings of Formal Modeling and Analysis of Timed Systems, 4th International Conference (FORMATS 2006), pp. 82-97, 2006.
[7] Egon Borger, Alessandra Cavarra and Elvinia Riccobene, On formalizing UML state machines using ASMs, Information and Software Technology, vol. 46, issue 5, pp. 287-292, 2004.
[8] F. Cassez and O.H. Roux, "Structural translation from Time Petri Nets to Timed Automata", the Journal of Systems and Software, Elsevier, vol. 79, no. 10, pp. 1456-1468, 2006.
[9] M.L. Crane and J. Dingel. "UML vs. Classical vs. Rhapsody Statecharts: Not all Models are Created Equal", Journal for Software and Systems Modelling (SoSym), vol. 6, no. 4, pp. 412-435, 2007.

[10] Werner Damm, Bernhard Josko, Amir Pnueli, Anjelika Votintseva, A discrete-time UML semantics for concurrency and communication in safety-critical applications. Sci. Comput. Program. vol. 55, no. 1-3, pp. 81-115, 2005.
[11] Alexandre David and M. Oliver Möller, From HUPPAAL to UPPAAL - A Translation from Hierarchical Timed Automata to Flat Timed Automata, BRICS report RS01-11, Dept. of Computer science, University of Aarhus, pp. 1-11, 2001.
[12] Conrado Daws, Sergio Yovine, "Reducing the number of clock variables of timed automata", 17th Real-Time Systems Symposium (RTSS), pp. 73-81, 1996.
[13] S. Distefano, M. Scarpa, and A. Puliafito, "From UML to Petri Nets: The PCM-Based Methodology", IEEE Trans. Software Eng., vol. 37, no. 1, pp. 65-79, 2011.
[14] B. P. Douglass, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison Wesley, p. 749, 1999.
[15] R. Eshuis and R. Wieringa, "Requirements Level Semantics for UML Statecharts", Proceedings of 4th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2000), pp. 121-140, 2000.
[16] H. Fecher, J. Schönborn, M. Kyas and W.P. de Roever, "29 New unclarities in the semantics of UML 2.0 state machines", Proceedings of the 7th International Conference on Formal Engineering Methods (ICFEM'05), pp. 52-65, 2005.
[17] Mohamed Ghazel, Formalizing a subset of ERTMS/ETCS specifications for verification purposes, Transportation Research Part C: Emerging Technologies, Elsevier, vol. 42, pp. 60-75, 2014.
[18] Mohamed Ghazel, Armand Toguyéni and Michel Bigand, "An UML approach for the metamodelling of automated production systems for monitoring purpose", Computers in Industry, Elsevier, vol. 55, pp. 283-299, 2004.
[19] David Harel and Amnon Naamad, The STATEMATE semantics of statecharts, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 5, no 4, pp. 293-333, 1996
[20] T.A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine, "Symbolic Model Checking for Real-time Systems", Information and Computation Journal, vol. 111, pp. 193-244, 1994.
[21] D.K. Kaynar, N. Lynch, R. Segala and F. Vaandrager, "The Theory of Timed I/O Automata", Morgan & Claypool, 2nd edition, ISBN 9781608450039, 2010.
[22] Alexander Knapp, Stephan Merz, and Christopher Rauh, "Model Checking Timed UML State Machines and Collaborations", 7th Intl. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'2002), pp. 395–414, 2002.
[23] P. Krcal and W. Yi, "Communicating timed automata: the more synchronous, the more difficult to verify", Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06), Seattle, WA, Springer-Verlag, pp. 249-262, 2006.
[24] H. Koziolek and R. Reussner, "A Model Transformation from the Palladio Component Model to Layered Queueing Networks", Proceedings of the SPEC International Workshop on Performance Evaluation: Metrics, Models and Benchmarks (SIPEW '08), Darmstadt, Germany, pp. 58-78, 2005.
[25] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, "Timed Automata with Data Structures for Distributed Systems Design and Analysis", Proceedings of the 3nd International Conference on Software Engineering and Formal Methods (SEFM'05), pp. 44-53, IEEE Computer Society Press, 2005.
[26] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, "Reachability results for timed automata with unbounded data structures", Journal Acta Informatica, vol. 47, no. 5-6, pp. 279-311, September 2010.
[27] K.G. Larsen, P. Pettersson and W. Yi, "UPPAAL in a Nutshell", Int. Journal on Software Tools for Technology Transfer, vol. 1, no. 1/2, pp. 134-152, 1997.
[28] Shuang Liu, Yang Liu, Étienne André, Christine Choppy, Jun Sun, Bimlesh Wadhwa, Jin Song Dong, "A Formal Semantics for Complete UML State Machines with Communications", IFM'2013, pp. 331-346.
[29] Chuan Ma and W. M. Wonham, Nonblocking Supervisory Control of State Tree Structures, IEEE Transactions on Automatic Control, Vol. 51, No. 5, May 2006.
[30] I. Malavolta, H. Muccini, P. Pelliccione and D.A. Tamburri, "Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies", IEEE Trans. Software Eng., vol. 36, no. 1, pp. 119-140, 2010.
[31] F. Mallet, "CCSL: Specifying Clock Constraints with UML/MARTE", Innovations in Systems and Software Engineering (ISSE), Special Issue on UML & Formal Methods, Vol. 4(3), Springer, pp. 309-314, 2008.
[32] Ahmed Mekki, Mohamed Ghazel, Armand Toguyéni, Validating time constrained systems using UML Statecharts Patterns and Timed Automata Observers, International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2009), Rabat, Morroco, 2009.
[33] P.M. Merlin, "A study of the recoverability of computing systems", Ph.D. thesis, University of California, Irvine, CA, 1974.
[34] Jean P. Mermet et al., "UML-B Specification for Proven Embedded Systems design", ISBN 978-1-4419-5256-1, Kluwer Academic Publishers, Boston, 2004.
[35] Ariadi Nugroho, "Level of detail in UML models and its impact on model comprehension: A controlled experiment", Information and Software Technology, vol. 51, issue 12, pp. 1670-1685, 2009.
[36] OMG, "Unified Modeling Language Specification 2.5.1: Superstructure", OMG, December 2017.
[37] F. Peres, B. Berthomieu and F. Vernadat, "On the composition of time Petri nets", Discrete Event Dynamic Systems Journal, vol. 21, no. 3, pp. 395-424, 2011.
[38] Anna Queralt, Ernest Teniente, "Verification and Validation of UML Conceptual Schemas with OCL Constraints", ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 21, no. 2, pp. 1-41, 2012.
[39] Jun Sun, Yang Liu, Jin Song Dong, Yan Liu, Ling Shi, Étienne André, "Modeling and verifying hierarchical real-time systems using stateful timed CSP" ACM Trans. Softw. Eng. Methodol, vol. 22, no. 1: 3, 2013.
[40] H. Tan, K. Beng, Y. Yang and L. Bian, "Systematic Transformation of Functional Analysis Model into OO Design and Implementation", IEEE Trans. Softw. Eng., vol. 32, no. 2, pp. 111-135, 2006.