Submodule construction for systems of timed I/O automata*

J. Drissi[1], G. v. Bochmann[2]

[1] *Dept. d'IRO, Université de Montréal, CP. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada, Phone: (514) 343-6161, Fax: (514) 343-5834, drissi@iro.umontreal.ca*
[2] *School of Information Technology & Engineering, University of Ottawa , Colonnel By Hall (A510), P.O.Box 450 Stn A,Ottawa,Ont.,K1N 6N5, Canada, Phone : (613) 562-5800 ext. 6205, Fax 562-5175 , bochmann@site.uottawa.ca*

Abstract. This paper addresses the problem of designing a submodule of a given system of communicating timed I/O automata. The problem may be formulated mathematically by the equation $(C||X)\ \mathfrak{R}\ A$ under the constraint $I_X=In$, where C represents the specification of the known part of the system, called the context, A represents the specification of the whole system, X represents the specification of the submodule to be constructed, || is a composition operator, $\mathfrak{R}$ is a conformance relation and In is the required set of inputs for X. As conformance relation, we consider the safe realization relation. This relation is implied by all the well known criteria of trace equivalence, complete trace equivalence, quasi equivalence and reduction. We propose an algorithms for solving the problem with respect to the safe realization and we characterize the set of solutions.

1 Introduction

One common problem, encountered in the hierarchical design of complex systems, in the synthesis of controllers and in the reuse of components, is the submodule construction problem, also called factorization problem or equation solving problem. The submodule construction problem (SCP) is to construct the specification of a submodule X when the specification of the whole system and all submodules except X are given. Such a problem may be formulated mathematically by the equation $(C||X)\ \mathfrak{R}\ A$ under the constraint $I_X=In$, where C represents the specification of the known part of the system, A represent the specification of the whole system, || is a composition operator and $\mathfrak{R}$ is a conformance relation and In is the required set of inputs for X. The SCP was first formulated and treated in [Merl 83], where specifications are expressed in terms of execution sequences, and trace equivalence was used as conformance relation. In [Shields 89], the author uses Milner's Calculus of Communicating Systems to model the same problem. Many other works [Haghverdi 96][Qin 1991] have been done using labelled transition systems as a model for the specifications and the strong and/or the observational equivalences as conformance relations. In [Drissi 99], we consider

* This work was supported by an NSERC Research grant.

this problem in the context of partial I/O automata for systems specification for various conformance relations.

Real-time programs such as airplane controllers, real-time operating systems, switching software and process controllers in manufacturing plants are inherently reactive, and their interaction with the environment must occur in real-time. Others systems in which the explicit notion of time plays an important role are communication systems and particularly communication protocols; the performance of such systems vitally depends on the value of timers, which control message retransmission. The correct operation of such systems is more than logical consistency in terms of event sequences, and extends to the satisfaction of real-time constraint. Timed models have been introduced for the specification and verification of real-time systems [Alur 94a]. In [Maler 95], the authors show that the control synthesis problem is solvable when the plant specification is given by a timed automaton. The solution is obtained by solving fixed-point equations involving both discrete transition relations and linear inequalities.

In the above work, it is assumed that the controller can precisely observe the whole configuration of the plant. However, in realistic situations the plant can be observed only up to some equivalence relation on its states, and the controller has to operate under some uncertainty. We generalize our previous work on submodule construction in the case of partial observation [Drissi 99] by dealing with partial timed I/O automata for systems specification and by using the safe realization as conformance relation.

The rest of the paper is structured as follows. In Section 2, we define basic notions. Section 3 presents the submodule construction problem and the architecture in which this problem will be solved. In Section 4, we propose an algorithm which takes as input a timed I/O automaton A, a timed I/O automaton C, and a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, and produces as output a timed I/O automaton $Sol_{\$}$ (if it exists) with $I_{Sol_{\$}}=In$ and $O_{Sol_{\$}}=(I_C \backslash I_A) \cup (O_A \backslash O_C)$, such that the composition of C with $Sol_{\$}$ is a safe realization of A. Then we characterize the set of all such solutions. Finally, in Section 5 we conclude the paper. The functions used in Algorithm 1 are defined in detail in the annex.

2 Timed Input-Output Automata

2.1 Basic notions and definitions

In this paper, a timed I/O automaton (briefly *TIOA*) A, is a tuple $(S_A, I_A, O_A, X_A, M_A, T_A, s_{0A})$ where S_A is a finite set of states with s_{0A} as the initial state, I_A is a non-empty, finite set of inputs, O_A is a non-empty, finite set of outputs with $I_A \cap O_A = \emptyset$, X_A is a finite set of clocks, M_A is a mapping that labels each state s in S_A with some clock constraint - called state invariant and

denoted $Inv(s)$ - in $\Phi(X_A)$ and $T_A \subseteq S_A \times (I_A \cup O_A) \times 2^{X_A} \times \Phi(X_A) \times S_A$ is a transition set. An element $(s, u, \lambda, \varphi, s') \in T_A$ represents a transition from state s to state s' on symbol u, the set $\lambda \subseteq X_A$ gives the clocks to be reset with this transition and φ is a clock constraint over X_A that specifies when the transition is enabled. The set $\Phi(X_A)$ of clock constraints is defined by the grammar :

$$\varphi := x \leq c \mid x \geq c \mid x < c \mid x > c \mid \varphi_1 \wedge \varphi_2$$

where x is a clock in X_A and c is a constant in the set $\mathbb{N}$ of nonnegative integers.

Definition 1 (k-polyhedral sets) [Maler 95]: Let k be a positive integer constant. For any positive integer constant d, we associate with k three subsets of $2^{\mathbb{R}^d}$:

$\mathcal{H}_k$: the set of sets having one of the following forms : $\mathbb{R}^d$, $\emptyset$, $\{v \in \mathbb{R}^d : v_i \# c$ for $i \in [1, d]\}$, $\{v \in \mathbb{R}^d : v_i - v_j \# c$ for $i, j \in [1, d]\}$, for some $\# \in \{<, \leq, >, \geq\}$, $c \in \{0, ..., k\}$.

$\mathcal{H}_k^{\cap}$: the set of convex sets consisting of intersections of elements of $\mathcal{H}_k$.

$\mathcal{H}_k^{*}$: the set of k-polyhedral sets containing all sets obtained from $\mathcal{H}_k$ via union, intersection and complementation.

For every k, $\mathcal{H}_k^{*}$ has a finite number of elements, each can be written as a finite union of convex sets.

A clock interpretation v for a set X_A of clocks assigns a real value to each clock; that is, it is a mapping from X_A to the set $\mathbb{R}$ of nonnegative reals. We say that a clock interpretation v for X_A satisfies a clock constraint φ over X_A if and only if φ evaluates to true according to the values given by v. For $\delta \in \mathbb{R}$, $v+\delta$ denotes the clock interpretation which maps every clock x to the value $v(x)+\delta$. For $Y \subseteq X_A$, $v[Y:=0]$ denotes the clock interpretation for X_A which assigns 0 to each $x \in Y$, and agrees with v over the rest of the clocks.

While transitions are instantaneous, time can elapse in a state. In each state the automaton can let time progress, i.e. it remains in the state and lets the values of the clocks increase uniformly as long as the invariant is satisfied [Labroue 98]. Since the model of I/O automata makes a very clear distinction between those actions that are performed under the control of the automaton -the outputs- and those actions that are performed under the control of its environment -the inputs-, we require that at least one transition labeled by an output action is enabled before the violation of the invariant.

If for each $s \in S_A$ and all $u \in I_A$ and any clock interpretation v that satisfies $Inv(s)$ there exists $(s, u, \lambda, \varphi, s') \in T_A$ such that v also satisfies φ, then A is said to be completely specified or input-enabled; otherwise A is partially specified. A *TIOA* A is said to be nondeterministic if there exists $(s, u, \lambda, \varphi, s') \in T_A$, $(s, u, \lambda', \varphi', s'') \in T_A$ with $s' \neq s''$ and a clock interpretation v that satisfies simultaneously $Inv(s)$, φ and φ' for some s and u; otherwise A is deterministic. The timed input-

enabled form of a *TIOA A*, denoted by *Tief*(*A*), is obtained as follow : for each state $s_1 \in S_A$ and an input $x_1 \in I_A$ let $Trans(s_1, x_1)=\{(s, x, \lambda, \varphi, s') \in T_A \mid s=s_1 \text{ and } x=x_1\}$ the set of transitions starting in s_1 and labeled with x_1, if the set of clock interpretations which satisfy $\overline{\varphi}(s_1, x_1)$, i.e. satisfy $Inv(s_1)$ and do not satisfy any constraint φ presents in$Trans(s_1, x_1)$, is not empty then we add (s_1, x_1, X_A, $\overline{\varphi}(s_1, x_1)$, $Fail_A$) to T_A, where *Fail* is an additional special state.

We note that the clocks are fictitious clocks, invented to express the timing properties of a *TIOA*, and all the clocks increase at an uniform rate corresponding to a fixed global time frame. Initially, all the clocks are finitialized to 0. A clock can be reset to zero simultaneously with any transition. At any instant, the reading of a clock equals the time elapsed since the last time it was reset.

Consider the sequence $\sigma=(a_1, t_1)...(a_k, t_k) \in ((I_A \cup O_A) \times \mathbb{R})^*$ with $t_1, ..., t_k$ an increasing suit of values in $\mathbb{R}$ and let $\nu_1(x)=t_1$ for each $x \in X_A$ and $\nu_{i+1}=\nu_i[\lambda_i:=0]+(t_{i+1}-t_i)$ for $k\text{-}1 \geq i \geq 1$, the sequence σ is said to be a timed-trace from the state s_{oA}, if there exist states $s_1, ..., s_{k+1} \in S_A$ such that $s_1=s_{oA}$, $(s_i, a_i, \lambda_i, \varphi_i, s_{i+1}) \in T_A$, and ν_i satisfies $Inv(s_i)$ and φ_i for each $i = 1,...,k$. For a state s in S_A, the sequence σ in $((I_A \cup O_A) \times \mathbb{R})^*$ is a timed-trace from the state s if there exists $\sigma_1=\sigma_2\sigma$ in $((I_A \cup O_A) \times \mathbb{R})^*$ such that σ_1 is a timed-trace from the state s_{oA} and σ_2 leads to s. The set of timed-traces from the state s is denoted $Ttr_A(s)$ and we denote it Ttr_A if $s=s_{oA}$. The property of non-Zenoness requires that for an infinite sequence the suit $t_1, ..., t_n$ ... is unbounded. For a deterministic *TIOA A*, a state s and a sequence $\sigma \in Ttr_A(s)$ uniquely determine the final state of the trace σ which we denote s_σ. A state s' of a *TIOA A* is reachable from a state s if there exists $\sigma \in Ttr_A(s)$ such that $s_\sigma=s'$. If s is the initial state of A then s' is said to be reachable. The set $\{s_1, s_2, ..., s_k, s_{k+1}\}$, denoted by $S(s_1, \sigma)$, represents the set of states reachable from s_1 by a prefix of σ. For each sequence $\sigma \in (\Sigma \times \mathbb{R})^*$ and a subset Σ' of Σ, the Σ'-projection of σ, denoted $Pr_{\Sigma'}(\sigma)$, is obtained by deleting from σ each pair (a, t) such that a is not in Σ'. For simplicity, we denote also by $Pr_A(\sigma)$, the projection of σ over the alphabet $(I_A \cup O_A)$ of the *TIOA A*. For a set of traces Y, we denote by $Pr_{\Sigma'}(Y)$, the set containing the Σ'-projection of the elements in Y. For a set X containing sets of traces, we denote by $Pr_{\Sigma'}(X)$, the set $\{Pr_{\Sigma'}(Y) \mid Y \in X\}$.

The connected component of A containing the initial state is the *TIOA* $CC(A)=(S_C, I_C, O_C, X_C, M_C, T_C, s_{oC})$ such that $S_C=\{s \in S_A \mid s \text{ reachable}\}$, $I_C=I_A$, $O_C=O_A$, $X_C=X_A$, $M_C=M_A|S_C$, $T_C=\{(s, u, \lambda, \varphi, s') \in T_A \mid s \in S_C\}$ and $s_{oC}=s_{oA}$, the notation $M_A|S_C$ represents the restriction of M_A to the set S_C. If $A=CC(A)$ then A is said to be initially connected.

We define a chaos *TIOA*, whose traces contain all the words over the alphabet, by $Ch=(\{ch\}, I, O, \emptyset, True, H, ch)$, where $H=\{(ch, t, \emptyset, true, ch) \mid t \in I \cup O\}$.

2.2 The composition of Timed I/O automata

A system can be considered as a finite collection of *TIOAs* communicating with one another and with the environment.

Definition 2 : Given two *TIOAs* $A_1=(S_{A_1}, I_{A_1}, O_{A_1}, X_{A_1}, M_{A_1}, T_{A_1}, s_{o1})$ and $A_2=(S_{A_2}, I_{A_2}, O_{A_2}, X_{A_2}, M_{A_2}, T_{A_2}, s_{o2})$ such that $O_{A_1}\cap O_{A_2}=\emptyset$ and $X_{A_1}\cap X_{A_2}=\emptyset$. The composition of A_1 and A_2 denoted $A_1||A_2$, is defined as the connected component of the *TIOA* $A=(S_A, I_A, O_A, X_A, M_A, T_A, s_{oA})$ where :

- $S_A=S_{A_1}\times S_{A_2}$,
- $I_A=(I_{A_1}\cup I_{A_2})\setminus(O_{A_1}\cup O_{A_2})$,
- $O_A=O_{A_1}\cup O_{A_2}$,
- $X_A=X_{A_1}\cup X_{A_2}$,
- $M_A(s_1, s_2)=M_{A_1}(s_1)\wedge M_{A_2}(s_2)$
- $((s_1, s_2), u, \lambda_1\cup\lambda_2, \varphi_1\wedge\varphi_2, (s_1', s_2'))\in T_A$ iff $\varphi_1\wedge\varphi_2\wedge Inv((s_1, s_2))\neq$false and for all $i\in\{1, 2\}$, if $u\in(I_{A_i}\cup O_{A_i})$ then $(s_i, u, \lambda_i, \varphi_i, s_i')\in T_{A_i}$, else $s_i=s_i'$, $\lambda_i=\emptyset$ and φ_i=true,
- $s_{oA}=(s_{o1}, s_{o2})$.

The composition of *TIOAs* is commutative and associative. This composition allows a number of composing *TIOAs* to accept the same input simultaneously.

We define a safety property which formalizes the non-occurrence of an unspecified reception in the composition A of a collection of *TIOA* $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, X_{A_i}, M_{A_i}, T_{A_i}, s_{oi}))_{1\leq i\leq n}$. We denote by ε the empty word.

Definition 3 : Given a collection of *TIOA* $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, X_{A_i}, M_{A_i}, T_{A_i}, s_{oi}))_{1\leq i\leq n}$, the composition $A=A_1||A_2||...||A_n$ is safe, written $\mathfrak{S}(A)$, iff any word σ in $((I_A\cup O_A)\times\mathbb{R})^*$ such that $Pr_{A_i}(\sigma)\in Ttr_{A_i}.((I_{A_i}\times\mathbb{R})\cup\{\varepsilon\})$ for all i, is a timed-trace of A (i.e. $\sigma\in Ttr_A$).

2.3 Removing ε-transitions from a Timed I/O automaton

When a transition is considered as an invisible action we call it an ε-transition. An example is an action which is internal in a composition. It is well known that in the untimed case such transitions do not increase the expressive power of finite automata. In the case of timed automata, the situation is different. The ε-transitions with clock resets strictly increase the power of timed automata [Bérard 98].

In the case of ε-transitions without reset, i.e. ε-transitions which do no reset clocks, we find in [Bérard 96] an algorithm to construct, given a *TIOA*, an equivalent one without such

internal transitions. This algorithm has two steps, it first suppresses the cycles of ε-transitions and then the remaining ones. The purpose of an ε-transition without reset is only to check that the clock constraint carried by the ε-transition is satisfied at some point in time. The main idea to remove this ε-transition is to shift this verification either forward or backward. This is done by using the forward closure or the backward closure of a clock constraint and by adding some new clocks.

Definition 4 : Let ϕ be a clock constraint. The forward closure of ϕ, denoted by $\overrightarrow{\phi}$, is a formula which is satisfied by a clock valuation ν if ϕ is satisfied by the clock valuation ν-δ for some $\delta \geq 0$:

$$\nu \models \overrightarrow{\phi} \text{ if } \exists \delta \geq 0,\ \nu\text{-}\delta \models \phi$$

Definition 5 : Let ϕ be a clock constraint and let λ be a set of clocks. The backward closure of ϕ with respect to λ, denoted by $\overleftarrow{\phi}^{\lambda}$, is a formula which is satisfied by a clock valuation ν if ϕ is satisfied by the clock valuation $\nu[\lambda:=0]+\delta$ for some $\delta \geq 0$:

$$\nu \models \overleftarrow{\phi}^{\lambda} \text{ if } \exists \delta \geq 0,\ \nu[\lambda:=0]+\delta \models \phi$$

Example :

In this example we illustrate how we remove an ε-transition without reset. For the automaton *A* we use the forward closure of the constraint ϕ_2 to remove the ε-transition and we obtain the automaton *B*. Note that a new clock x_o and an associated constraint are necessary. This method cannot be used when the ε-transition is not followed by a visible action. In order to remove the ε-transition in the automaton C we use the backward closure of the constraint ϕ_2 and we obtain the automaton *D*.

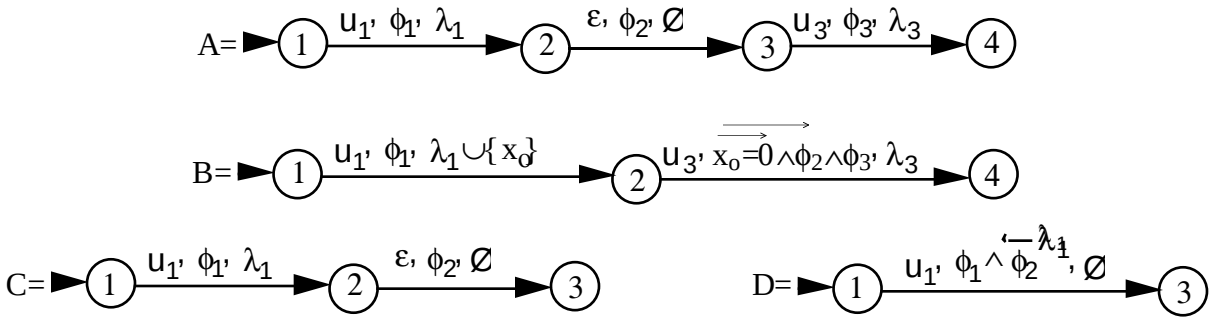

Figure 1: Two automata and their equivalent without ε-transitions

2.4 Minimization of the number of clocks of a Timed I/O automaton

The number of clocks used in a specification mainly grows for two reasons. First,

specifications are often written in high-level description languages and later compiled into timed automata having a number of clocks proportional to the number of time-outs that appear in the description. However, these time-outs are rarely active at the same time, and hence, the number of clocks can be reduced. Second, complex systems are described as the composition of simpler components each having a small number of clocks. It turns out that, due to the synchronization of transitions, many clocks are simultaneously reset and therefore they will be equal for some time since they all proceed at the same speed. Clearly in this case, only one of these clocks is really necessary.

Taking into account these observations, the authors in [Daws 96] propose a method for reducing the number of clocks of a timed automaton by combining two algorithms. The first one consists of detecting active clocks. Intuitively, a clock is active at some state if its value at the state may influence the future evolution of the system. This may happen whenever the clock appears in the invariant condition of the state or is tested in the condition of some of the outgoing edges. The values of inactive clocks at state *s* are not relevant to the evolution of the system at *s*. This means that the number of clocks required is equal to the greatest number of active clocks in all states. The second algorithm consists of detecting pairs of clocks that are always equal. Two clocks are equal in a state if for every incoming edge they are both reset or are equal in the source location and none is reset.

2.5 Tightening the constraints

Given a *TIOA A*, some transitions will never be fired since the constraints associated with such transitions are always false when we reach the corresponding states. Removing these transitions from *A* will not change the set Ttr_A. There have been a number of works dealing with this problem [Courcoubetis 91][Somé 97]. The region graph [Alur 94a] corresponding to the *TIOA A* can be used to remove such transitions but the complexity is polynomial in the number of states and edges of *A* and exponential in the number of clocks and the binary length of the constants that appear in the constraints. In [Somé 97] the author propose an algorithm based on the propagation of clock relations which allows the detection of such transitions. Intuitively, we will find for each state the set of clock interpretations which are satisfied in the state and update the constraints associated with each outgoing transition. Then each transition for which the updated constraint is false will be removed.

Example :

The example of Figure 2 illustrates the propagation of clock relations. For example in state 3, vsup(x, 1) means that the value of the clock x is superior to 1 at this state. The transition (3, c,

x<1, Ø, 4) will never be fired. Hence this transition can be removed from the automaton.

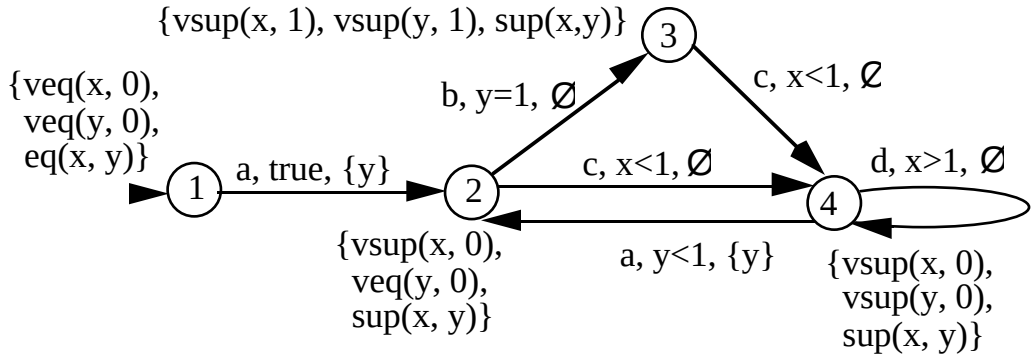

Figure 2 : Propagation of clock relations approach

2.6 The safe realization relation

Definition 6 : For a *TIOA A* and a composite *TIOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, we say that *B* realizes *A* with safety, written $B \leq_{\mathbb{S}} A$, iff

i - for every *TIOA E*, with $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A)$ implies $\mathbb{S}(E||B_1||B_2||...||B_n)$,

Definition 6 means that for any environment *E* over the alphabet of *A*, if the composition of *A* and *E* is safe then the composition of B_1, B_2, ..., B_n and *E* must also be safe, i.e. in any reachable state of the composition $E||B_1||B_2||...||B_n$, there is no unspecified reception.

The reflection $\tilde{A}$ of a *TIOA A* represents the most liberal timed-environment in with *A* is safe, i.e. $\tilde{A}$ must accept all the outputs produced by *A* and nothing more, and produces all the inputs accepted by A. Since *A* may be non-deterministic, each state s of $\tilde{A}$ must have the following properties :

- The sets of outputs of *s* will be equal to the intersection of the sets of inputs of all the states of A which are reachable by a timed trace leading to s.
- The set of inputs of s will be equal to the union of the sets of outputs of all the states of A which are reachable by a timed trace leading to s.

For the construction of $\tilde{A}$, we use two functions *RestrictOutputs* and *AugmentInputs*. The first function takes a *TIOA A* and produces a *TIOA RestrictOutputs*(*A*) for which all the states which are reachable by a common timed trace produce the same set of outputs (the intersection of the sets of outputs of the corresponding states in *A*). The second function takes a *TIOA A* and produces a *TIOA AugmentInputs*(*A*) for which all the states which are reachable by a common timed trace accept the same set of inputs (the union of the sets of inputs of the corresponding states in *A*). In the case of a deterministic *TIOA* A, we have *RestrictOutputs*(*A*)=*AugmentInputs*(*A*)=A.

Definition 7 : The reflection of a *TIOA* $A=(S_A, I_A, O_A, X_A, M_A, T_A, s_{0A})$ is the *TIOA* $\tilde{A}=AugmentInputs(RestrictOutputs((S_A, I_{\tilde{A}}, O_{\tilde{A}}, X_A, M_A, T_A, s_{0A})))$ where $I_{\tilde{A}}=O_A$, $O_{\tilde{A}}=I_A$.

Lemma 1: For a *TIOA A* and a composite *TIOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, the following propositions are equivalent :

i - $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$,

ii - for every *TIOA E*, with $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A) \Rightarrow \mathbb{S}(E||B_1||B_2||...||B_n)$.

2.7 The timed-safe realization relation

Definition 8 : For a *TIOA A* and a composite *TIOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, we say that *B* realizes *A* with *T*-safety, written $B \leq_{\mathbb{T}\mathbb{S}} A$, iff

i - $B \leq_{\mathbb{S}} A$,

ii - for every state $(s, s_1, s_2, ..., s_n)$ in $\tilde{A}||B_1||B_2||...||B_n$, $Inv((s_1, s_2, ..., s_n))$ implies $Inv(s)$.

Definition 8 means that $B_1||B_2||...||B_n$, realizes *A* with safety, moreover, for each state $(s_1, s_2, ..., s_n)$ in $B_1||B_2||...||B_n$ such that $(s, s_1, s_2, ..., s_n)$ is in $\tilde{A}||B_1||B_2||...||B_n$ the invariant $Inv((s_1, s_2, ..., s_n))$ implies $Inv(s)$.

3 The design of a submodule

3.1 The architecture

We consider the class of systems which can be represented by two *TIOA* that communicate with one another and with an environment (Figure 3). One *TIOA*, called the *context C*, models the known part of the system, the behavior of which is given, while the other deterministic *TIOA*, called the new component *Comp*, represents the behavior of a certain component of the system (the submodule to be designed). The set of inputs accepted by the system from the environment can be divided into three disjoint sets. The first is the set of inputs of the context *C* non observable by the component, the second is the set of inputs of the context observable by the component, and the third represents the set of inputs observable by the component, but not visible by the context. Similarly, the set of outputs delivered by the system to the environment can be divided in three disjoint sets. The first is the set of outputs delivered by the context *C* to the environment which are not observable by the new component, the second is the set of outputs delivered by the context *C* to the environment which are observable by the component, and the third represents the set of outputs of the component accepted by the environment and not delivered to the context. In addition, the context and the component can communicate with internal actions not observable by the environment.

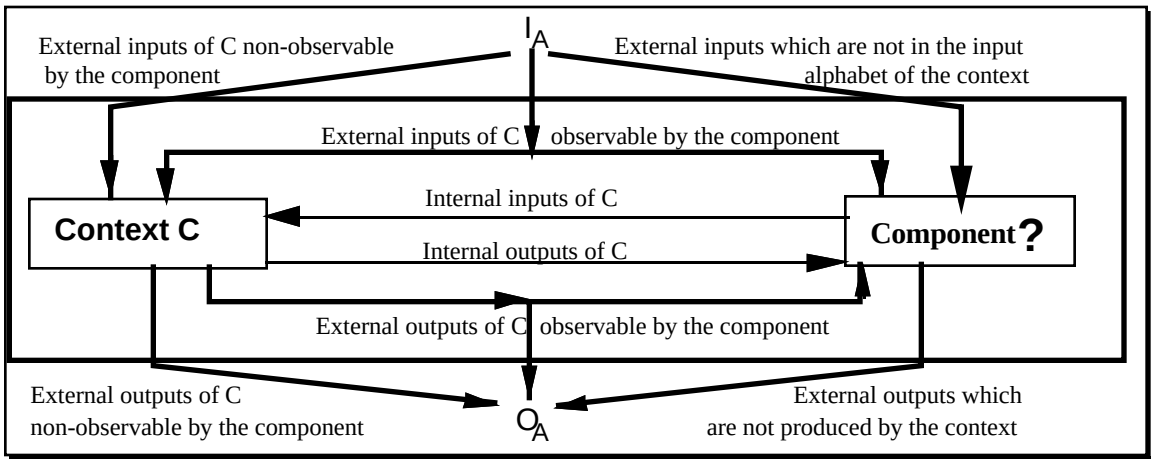

Figure 3: The composition of two communicating components *C* and *Comp*

3.2 The problem

The problem is known as the problem of submodule construction, redesign or equation solving, where an appropriate conformity criterion should hold between a designed system and its given specification *A* and where the system consists of a given component *C* and a new component *X* to be designed. In this paper, we consider this problem as a problem of equation solution in the realm of *TIOA* for the equation $(C||X)\leq_{\S} A$ under the constraint $I_X=In$ with *X* being a free variable and *In* a given set of inputs, which satisfies the constraint $(I_A\backslash I_C)\cup(O_C\backslash O_A)\subseteq In\subseteq I_A\cup O_C$.

4 The solution for the safe realization relation

4.1 The proposed method

We use a chaos *TIOA Chaos* which represents all the traces over the input alphabet *In* and the output alphabet $(I_C\backslash I_A)\cup(O_A\backslash O_C)$; any solution of $(C||X)\leq_{\S} A$ is trace included in *Chaos*. The main idea of our approach is to remove from *Chaos* all the timed-traces which combined with timed-traces of the context *C* in the environment $\tilde{A}$ may cause a non-safe behavior. This will allow us to capture the set (if not empty) of the permissible timed-traces of the component to be designed in the form of a *TIOA* $Sol_{\S}$, called the generic safe solution. A permissible timed-trace is a timed-trace of a solution of $(C||X)\leq_{\S} A$. For this purpose, we construct the timed input-enabled forms of *A'* and *C'*, where *A'* is obtained from *A* by thigtening the constraints and then removing all the invariants and *C'* is obtained from *C* by thigtening the constraints. We construct the composition $R=Tief(C')||Chaos\ ||Tief(\tilde{A}')$ which is equal to $Tief(C')||Tief(\tilde{A}')$. Since we consider the safe realization relation, we replace all the invariants in *R* by *true*. In the case of partial observation, i.e. $In\subset I_A\cup O_C$, we must hide all the transitions labelled with actions in $(I_A\cup O_C)\backslash In$. We require that

all those transitions do not reset clocks. The obtained automaton will be in general a non deterministic one. Since the class of deterministic *TIOA* is strictly included in the class of non deterministic *TIOA* [Alur 94b], we must transform our automaton before removing a transition leading to the state *Fail*. Two states reachable with a common timed-trace in the transformed automaton must accept the same set of inputs and produce the same set of outputs. We give in the next paragraph an algorithm which constructs the generic safe solution. As input, the algorithm requires two *TIOAs C* and *A* and a set *In* such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$. The set *In* define the input set for the generic solution.

4.2 Description of Algorithm 1

We present in this subsection, the principles of the five steps of our algorithm. The annex includes more details about the algorithm.

Input : The specification of the context *C*, the specification of the system *A* and a set of inputs *In*.
Output : A *TIOA* $Sol_{\S}$ with $I_{Sol_{\S}} = In$ such that $(C \| Sol_{\S}) \leq_{\S} A$ if there is such a solution.

Step 1 : Completing the automata and tightening their constraints
We construct the timed input-enabled forms of *A'* and *C'*, where *A'* is obtained from *A* by thigtening the constraints and then removing all the invariants and *C'* is obtained from *C* by thigtening the constraints.

Step 2 : Composition
We construct the composed automaton $R := Tief(C') \| Tief(\widetilde{A'})$. Each time we reach a state of R which contains $Fail_{C'}$ or $Fail_{\widetilde{IA'}}$ we replace it by $Fail_R$.

Step 3 : Clock minimization
We replace all the invariants in *R* by *true* then we minimize the number of clocks in *R*.

Step 4 : Eliminating ε-transitions
We replace in *R* all the actions which are not in $In \cup (I_C \backslash I_A) \cup (O_A \backslash O_C)$ by the invisible action ε. Then we transform *R* into an equivalent *TIOA* without ε-transitions. For this purpose, we first remove all the ε-transitions which lead to a silent state, and for each state *p* where there exists an outgoing transition labelled by an ε-transition we add a new clock x_p which is reset by any transition entering the state *p*. Then,

• for each path $p_1 \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_1} p_2 \ \ldots\ p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} p_{k+1} \xrightarrow{u,\ \lambda,\ \varphi_{k+1}} p_{k+2}$, where $(p_i)_{1 \leq i \leq k+1}$ are distinct states, we add the transition $(p_1, u, \lambda, x_{p_1}=0 \wedge \varphi_1 \wedge \ldots \wedge \varphi_k \wedge \varphi_{k+1}, p_{k+2})$.

• for each path $p_1 \xrightarrow{u,\ \lambda,\ \varphi_1} p_2 \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_2} \dots p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} Fail_{R_1}$, where $(p_i)_{1\le i\le k}$ are distinct states, we add the transition $(p_1, u, \emptyset, \varphi_1 \wedge \overleftarrow{x_{p_2}=0} \wedge \varphi_2 \wedge \dots \wedge \varphi_{k-1} \wedge \varphi_k\ , Fail_{R_1})$.

• for each path $s_{oR} \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_1} p_2 \dots p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} Fail_R$, we add the transition $(s_{oR}, \mathcal{E}, \emptyset, \overrightarrow{\varphi_1} \wedge \dots \wedge \varphi_{k-1} \wedge \varphi_k, Fail_{R_1})$, where $\mathcal{E}$ is new label.

Finally we remove all the ε-transitions. The obtained automaton is denoted R_1.

Step 5 : Eliminating transitions leading to *Fail*

In this step we will remove from R_1 all the traces which lead to $Fail_{R_1}$. Unlike the untimed case, a transition leading to $Fail_{R_1}$ can be avoided if it is possible to put in the state an invariant which disables this transition by leaving the state before the time of occurrence of the transition.

Part 1 : In this part, we will use the function *Remove_$\mathcal{E}$_transitions* which takes as input a *TIOA* and returns, if possible, a *TIOA* without *$\mathcal{E}$_transitions*; else it returns "NO SOLUTION". An *$\mathcal{E}$_transition* is due to a sequence of interaction between the environment and the context leading to the *Fail* state without intervention of the component to be designed. The function *Remove_$\mathcal{E}$_transitions* tries to find an invariant to put in the initial state of the component to be designed such that its earliest interaction with the environment or the context will prohibit the completion of the sequence of interaction leading to the *Fail* state.

Part 2 : Since the component to be designed will be obtained in the form of a non deterministic *TIOA*, the environment and the context cannot precisely observe the state of the component. All states of the component which are reachable by a given timed trace must accept the same set of inputs. For this purpose we use the previously defined function *AugmentInputs* which will take as input the *TIOA* obtained at the end of Part 1 and return a *TIOA* which has the above property.

Part 3 : Now we are ready to deal with the transitions leading to the *Fail* state. We use the function *Remove_Fail* which takes as input the *TIOA* obtained at the end of Part 2 and removes from it, if possible, all the timed traces leading to the *Fail* state. For each transition $(s, u, \emptyset, \varphi_2, Fail_{R_1})$ we first propagate this transition, i.e. we transform our automata to obtain one where all the states reachable by a timed trace leading to s, have a transition which leads with action *u* and under constraint φ_2 to $Fail_{R_1}$. To remove this transition there are many cases. For example (Figure 4), if x_1 is an input action and $Inv(s)=true$, we consider the constraint $\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overleftarrow{\varphi_2}^{\emptyset} \wedge \neg \varphi_2)$ where *CONSTRANS* is the set of all the constraints associated with transitions starting in *s* and labelled by outputs. If $\Phi \neq$ false, we put $Inv(s) = \overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset}$ to disable the transition $(s, x_1, \emptyset, \varphi_2,$

$Fail_{R_1}$) and each ingoing transition (s', u', λ', φ', s) is duplicated into two transitions (s', u', λ', $\varphi' \wedge Inv(s)^{\lambda'}$, s) and (s', u', $\emptyset$, $\varphi' \wedge \neg Inv(s)^{\lambda'}$, $Fail_{R_1}$) where $Inv(s)^{\lambda'}$ is the set of constraints derived from $Inv(s)$ by removing the constraints on clock in λ. Otherwise, we remove all the outgoing transitions from s and replace s by $Fail_{R_1}$.

If we obtain an automaton at the end of this step we call it $Sol_{\S}$.

4.3 Example for Step 5 :

In the following example we illustrate how we remove a transition leading to $Fail_{R_1}$. We consider a state s with his ingoing and outgoing transitions. We have :

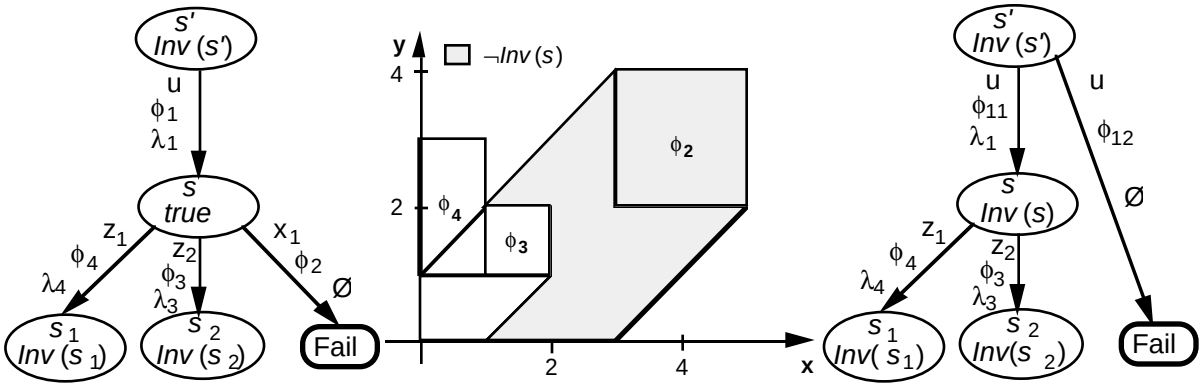

Figure 4 : Illustration of removing a transition leading to $Fail_{R_1}$.

the constraint φ_1 is : $(x \leq 5) \wedge (y \leq 4) \wedge (y \geq 2)$ and $\lambda_1 = \{y\}$, the constraint φ_2 is : $(x \leq 5) \wedge (x \geq 3) \wedge (y \leq 4) \wedge (y \geq 2)$, the constraint φ_3 is : $(x \leq 2) \wedge (x \geq 1) \wedge (y \leq 2) \wedge (y \geq 1)$, the constraint φ_4 is : $(x \leq 1) \wedge (y \leq 3) \wedge (y \geq 1)$, the obtained constraint Φ is : $(x \leq 2) \wedge (y \leq 2) \wedge (y - x \leq 1)$, since $\Phi \neq$false the obtained constraint $\overline{\Phi}^{\emptyset}$ is : $(x \leq 2) \wedge (y \leq 2) \wedge (y - x \leq 1) \wedge (x - y \leq 1)$, and the obtained constraint $\overline{\varphi_2}^{\emptyset}$ is : $(y - x \leq 1) \wedge (y \leq 4) \wedge (x \leq 5) \wedge (x - y \leq 3)$,
then $Inv(s)$ will be : $((x \leq 2) \wedge (y \leq 2) \wedge (y - x \leq 1) \wedge (x - y \leq 1)) \vee (y - x > 1) \vee (y > 4) \vee (x > 5) \vee (x - y > 3)$,
the constraint $Inv(s)^{\{y\}}$ is : $(x \leq 1) \vee (x > 3)$,
then φ_{11} will be : $((x \leq 1) \vee (x > 3)) \wedge (y \leq 4) \wedge (y \geq 2)$ and φ_{12} will be : $(x > 1) \wedge (x \leq 3) \wedge (y \leq 4) \wedge (y \geq 2)$.

4.4 The set of solutions

Theorem 1 : Given two *TIOAs* A and C, and given a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, if Algorithm 1 produces a *TIOA* $Sol_{\S}$ then it is a solution to the submodule construction problem, i.e. $(C \parallel Sol_{\S}) \leq_{\S} A$ for the case $I_X = In$. Otherwise there is no solution to this problem.

The solution obtained by Algorithm 1 is generic, which means that we can derive from it all

the solutions for the equation $(C \| X) \leq_{\S} A$ under the constraint $I_X=In$.

Theorem 2 : Given two *TIOAs A* and *C*, and given a set *In* such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, if Algorithm 1 produces an *TIOA* $Sol_{\S}$ then for any *TIOA B*, with $I_B=In$ and $O_B=(I_C \backslash I_A) \cup (O_A \backslash O_C)$, the following propositions are equivalent :

i - $B \leq_{\mathbb{T}\S} Sol_{\S}$,

ii - *B* is a solution of the equation $(C \| X) \leq_{\S} A$.

5 Conclusion

We have presented in this paper an approach to solve the problem of submodule construction in the realm of timed I/O automata. This problem may be formulated mathematically by the equation $(C||X)$ ℜ A under the constraint $I_X=In$, where *C* represents the specification of the known part of the system, *A* represents the specification of the whole system, *X* represents the specification of the submodule to be constructed, || is a composition operator, ℜ is a conformance relation and *In* is the required set of inputs for *X*. The conformance relation considered is the safe realization relation. We consider the case where the non-observable interactions between the context and the environment does not reset clocks. The set of solutions to the equation (if they exist) can be represented as the set of timed-safe realizations of a timed I/O automaton $Sol_{\S}$. An algorithm for finding $Sol_{\S}$ is given.

The approach considered in this paper may be extended to treat the problem of submodule construction for other conformance relations, like for example the conforming implantation relation [Drissi 99] which imposes constraints on the outputs produced in each state of the whole system. Another interesting problem is the case where the non-observable interactions between the context and the environment may reset clocks. Further work will be done in these directions.

References

[Alur 94a] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science, 126:183-235, 1994.

[Alur 94b] R. Alur, L. Fix and T. A. Henzinger, *A determinizable class of timed automata*, In Proceedings of CAV'94, Lecture Notes in Computer Science, vol. 818, pp. 1-13, 1994.

[Bérard 96] B. Bérard, P. Gastin and A. Petit, *On the power of non observable actions in timed automata*, In Proceedings of STACS'96, number 1046 in Lecture Notes in Computer Science, pp. 257-268, Springer Verlag, 1996.

[Bérard 98] B. Bérard, V. Diekert, P. Gastin and A. Petit, *Characterization of the expressive power of silent transitions in timed automata*, In Fundamenta Informaticae, 36(2):145-182, 1998.

[Courcoubetis 91] C. Courcoubetis and M. Yannakakis, *Minimum and maximum delay problems in real-time systems*, In proceedings of CAV'91, number 575 in Lecture Notes in Computer Science, pages 399-409, Spinger Verlag, 1991.

[Daws 96] C. Daws and S. Yovine, *Reducing the number of clock variables of timed automata*, In Proceedings of RTSS'96, Whasington DC, USA, Dec. 4-6, 1996.

[Drissi 99] J. Drissi and G. v. Bochmann, *Submodule construction for systems of I/O automata*, Technical Report no. 1133, DIRO, University of Montreal, 1999.

[Haghverdi 96] E. Haghverdi and H. Ural, *An Algorithm for Submodule Construction*, Technical report of the Department of computer Science, University of Ottawa, 1996.

[Labroue 98] A. Labroue, *Conditions de vivacité dans les automates temporisés*, Research Report LSV-98-7, Sep. 1998, Laboratoire Spécification et Vérification, Ecole normale supérieure de Cachan, France. http://www.lsv.ens-cachan.fr.

[Lynch 88] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, MIT/LCS/TM-373, Laboratory for computer science, Massachusetts Institute of Technology, Nov. 1998.

[Lynch 92] N. A. Lynch and H. Attiya, *Using mappings to prove timing properties*, Distributed Computing, 6:121-139, 1992.

[Maler 95] O. Maler, A. Pnueli and J. Sifakis, *On the Synthesis of Discrete Controllers for Timed Systems*, In Proceedings of 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 1995, Lecture Notes in Computer Science, vol. 900, pp. 229-242.

[Merl 83] P. Merlin and G. v. Bochmann, *On the Construction of Submodule Specifications and Communication Protocols*, ACM Trans. on Programming Languages and Systems, Vol. 5, No. 1 (Jan. 1983), pp. 1-25.

[Merritt 91] M. Merritt, F. Modugno and M. R. Tuttle, *Time-Constrained Automata*, In Proceedings of CONCUR'91, Lecture Notes in Computer Science, vol. 527, pp. 408-423, Amsterdam, August 1991.

[Qin 1991] H. Qin and P. Lewis, *Factorisation of finite State Machines under Strong and Observational Equivalences*, Journal of Formal Aspects of Computing, Vol. 3, pp 284-307, July-Sept. 1991.

[Shields 89] M. W. Shields, *Implicit System Specification and the Interface Equation*, Computer Journal, Vol. 32, 5, pp. 399-412, Oct. 1989.

[Somé 97] S. s. Somé, *Dérivation de specification à partir de Scénarios d'interaction*, Thèse de doctorat, DIRO, Université de Montréal, 1997.

Annex

Notations :

v_0 is the clock interpretation defined by $v_0(x)=0$ for each clock x in X_A;

φ^λ= the constraint derived from φ by replacing the clocks in λ by 0;

CONSTR(λ)= the set of constraints $x=0$ where x is a clock in λ;

RestrictOutputs(TIOA A) : TIOA

FOR each state s in A **DO**

TEMP=A;

Remove from *TEMP* all the states from which there exists no path leading to s;

$I_{TEMP}=I_A\cup O_A$;

$O_{TEMP}=\emptyset$;

Rename all the clocks in *TEMP*;

TEMP=*TEMP* ||A;

Remove from *TEMP* all the states from which there exists no path leading to (s,s') with $s'\neq s$;

Add to A the states (s_1, s_2) of *TEMP* with $s_1\neq s_2$;

FOR each state (s_1, s_2) of *TEMP* **DO**

IF $s_1\neq s_2$ **THEN**

FOR each transition ((s_1, s_2), u, λ, φ, (s_3, s_4)) in *TEMP* **DO**

IF $s_3\neq s_4$ **THEN**

Add the transition ((s_1, s_2), u, λ, φ, (s_3, s_4)) to A;

ELSE

Add the transition ((s_1, s_2), u, λ, φ, s_3) to A;

Add the transition ((s_1, s_2), u, λ', φ', s_4) to A to complete the corresponding transition (s_2, u, λ', φ'', s_4) in A with $\varphi'=\varphi''\wedge\neg\varphi$;

FOR each transition (s_2, u', λ', φ', s') in A **DO**

IF u' is not already in a transition starting from (s_1, s_2) **THEN**

Add the transition ((s_1, s_2), u', λ', φ', s') in A;

ELSE

FOR each transition ((s_1, s_2), u, λ, φ, (s_3, s_4)) in *TEMP* **DO**

IF $s_3\neq s_4$ **THEN**

Add the transition (s_1, u, λ, φ, (s_3, s_4)) in A;

Replace the transition (s_1, u, λ', φ', s_4) in A by (s_1, u, λ', $\varphi'\wedge\neg\varphi$, s_4);

FOR each state (s, s_2) of A **DO**

FOR each transition (s, u, λ, φ, s_3) in A with $u\in O_A$ **DO**

Replace each transition ((s, s_2), u, λ', φ', s') in A by ((s, s_2), u, λ, $\varphi\wedge\varphi'$, s') in A;

AugmentInputs*(*TIOA* R_1) : *TIOA

FOR each state $s \neq Fail_{R_1}$ in R_1 **DO**

$TEMP = R_1$;

Remove from *TEMP* all the states from which there exists no path leading to s;

$I_{TEMP} = I_{R_1} \cup O_{R_1}$;

$O_{TEMP} = \emptyset$;

Rename all the clocks in *TEMP*;

$TEMP = TEMP \,||\, R_1$;

Remove from *TEMP* the states $(s', Fail_{R_1})$ and all the transitions leading to them;

Remove from *TEMP* all the states from which there exists no path leading to (s, s') with $s' \neq s$ and $s' \neq Fail_{R_1}$;

Add to R_1 the states (s_1, s_2) of *TEMP* with $s_1 \neq s_2$;

FOR each state (s_1, s_2) of *TEMP* **DO**

IF $s_1 \neq s_2$ **THEN**

FOR each transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ in *TEMP* **DO**

IF $s_3 \neq s_4$ **THEN**

Add the transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ to R_1;

ELSE

Add the transition $((s_1, s_2), u, \lambda, \varphi, s_3)$ to R_1;

Add the transition $((s_1, s_2), u, \lambda', \varphi', s_4)$ to R_1 to complete the corresponding transition $(s_2, u, \lambda', \varphi'', s_4)$ in R_1 with $\varphi' = \varphi'' \wedge \neg\varphi$;

FOR each transition $(s_2, u', \lambda', \varphi', s')$ in R_1 **DO**

IF u' is not already in a transition starting from (s_1, s_2) **THEN**

Add the transition $((s_1, s_2), u', \lambda', \varphi', s')$ in R_1;

ELSE

FOR each transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ in *TEMP* **DO**

IF $s_3 \neq s_4$ **THEN**

Add the transition $(s_1, u, \lambda, \varphi, (s_3, s_4))$ in R_1;

Replace the transition $(s_1, u, \lambda', \varphi', s_4)$ in R_1 by $(s_1, u, \lambda', \varphi' \wedge \neg\varphi, s_4)$;

FOR each state (s, s_2) of R_1 **DO**

FOR each transition $(s, u, \lambda, \varphi, s_3)$ in R_1 **DO**

IF $u \in I_{R_1}$ **THEN**

$PresConst := \{\varphi' \mid ((s, s_2), u, \lambda', \varphi', s')$ in $R_1\}$;

$\varphi'' := \varphi \wedge \neg(\bigvee_{\varphi' \in PresConst} \varphi')$;

Add the transition $((s, s_2), u, \lambda, \varphi'', s_3)$ in R_1;

Remove_$\mathcal{E}$_transitions(TIOA R_1) : TIOA

WHILE there exists a transition $(s_{oR}, \mathcal{E}, \emptyset, \varphi_1, Fail_{R_1})$ **DO**

CONSTRANS=$\{\varphi \mid (s_{oR}, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ and $u \in (I_C \setminus I_A) \cup (O_A \setminus O_C)\}$;

$\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overline{\varphi_1}^{\emptyset} \wedge \neg \varphi_1)$;

IF v_o satisfy $\overline{\varphi_1}^{\emptyset}$ **THEN**

IF $\Phi \wedge \overrightarrow{CONSTR(X_{R_1})} \neq \emptyset$ **THEN**

Delete the transition $(s_{oR}, \mathcal{E}, \emptyset, \varphi_1, Fail_{R_1})$;

$Inv(s_{oR}) := Inv(s_{oR}) \wedge (\overline{\Phi}^{\emptyset} \vee \neg \overline{\varphi_1}^{\emptyset})$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s, u, \lambda, \varphi, s_{oR}) \in T_{R_1}$ **DO**

IF $\varphi^{\lambda} \wedge \neg Inv(s_{oR}) \neq \emptyset$ **THEN**

Replace $(s, u, \lambda, \varphi, s_{oR})$ by $(s, u, \lambda, \varphi \wedge Inv(s_{oR})^{\lambda}, s_{oR})$;

Add in R_1 the transition $(s, u, \emptyset, \varphi \wedge \neg Inv(s_{oR})^{\lambda}, Fail_{R_1})$;

ELSE return " NO SOLUTION"; **STOP**;

ELSE

Delete the transition $(s_{oR}, \mathcal{E}, \emptyset, \varphi_1, Fail_{R_1})$;

IF $\Phi \neq \emptyset$ **THEN**

$Inv(s_{oR}) := Inv(s_{oR}) \wedge (\overline{\Phi}^{\emptyset} \vee \neg \overline{\varphi_1}^{\emptyset})$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s, u, \lambda, \varphi, s_{oR}) \in T_{R_1}$ **DO**

IF $\varphi^{\lambda} \wedge \neg Inv(s_{oR}) \neq \emptyset$ **THEN**

Replace $(s, u, \lambda, \varphi, s_{oR})$ by $(s, u, \lambda, \varphi \wedge Inv(s_{oR})^{\lambda}, s_{oR})$;

Add in R_1 the transition $(s, u, \emptyset, \varphi \wedge \neg Inv(s_{oR})^{\lambda}, Fail_{R_1})$;

ELSE

FOR each ingoing transition $(s, u, \lambda, \varphi, s_{oR}) \in T_{R_1}$ **DO**

IF $\varphi^{\lambda} \wedge \overline{\varphi_1}^{\emptyset} \neq \emptyset$ **THEN**

Replace $(s, u, \lambda, \varphi, s_{oR})$ by $(s, u, \lambda, \varphi \wedge (\neg \overline{\varphi_1}^{\emptyset})^{\lambda}, s_{oR})$;

Add in R_1 the transition $(s, u, \emptyset, \varphi \wedge (\overline{\varphi_1}^{\emptyset})^{\lambda}, Fail_{R_1})$;

Remove_Fail*(*TIOA* R_1) : *TIOA

WHILE there exists a transition (s, u, λ, φ_2, $Fail_{R_1}$) **DO**

$TEMP$:=R_1;

Remove from *TEMP* all the states from which there exists no path leading to s;

Add to *TEMP* the state $Fail_{TEMP}$ and the transition (s, u, λ, φ_2, $Fail_{TEMP}$);

I_{TEMP}=$I_{R_1}\cup O_{R_1}$; O_{TEMP}=$\emptyset$;

Rename all the clocks in *TEMP*;

$TEMP$=$TEMP$ ||R_1;

Remove from *TEMP* the states (s', $Fail_{R_1}$) and all the transitions leading to them;

Remove from *TEMP* all the states from which there exists no path leading to ($Fail_{TEMP}$, s') with $s'\neq Fail_{R_1}$;

Add to R_1 the states (s_1, s_2) of *TEMP* with $s_1\neq s_2$;

FOR each state (s_1, s_2) of *TEMP* with $s_1\neq Fail_{TEMP}$ **DO**

IF $s_1\neq s_2$ **THEN**

FOR each transition ((s_1, s_2), u_1, λ_1, φ_1, (s_3, s_4)) in *TEMP* **DO**

IF $s_3\neq s_4$ **THEN** Add the transition ((s_1, s_2), u_1, λ_1, φ_1, (s_3, s_4)) to R_1;

ELSE Add the transition ((s_1, s_2), u_1, λ_1, φ_1, s_3) to R_1;

Add the transition ((s_1, s_2), u_1, λ', φ', s_4) to R_1 to complete the corresponding transition (s_2, u_1, λ', φ'', s_4) in R_1 with $\varphi'=\varphi''\wedge\neg\varphi_1$;

FOR each transition (s_2, u', λ', φ', s') in R_1 **DO**

IF u' is not already in a transition starting from (s_1, s_2) **THEN**

Add the transition ((s_1, s_2), u', λ', φ', s') in R_1;

ELSE

FOR each transition ((s_1, s_2), u_1, λ_1, φ_1, (s_3, s_4)) in *TEMP* **DO**

IF $s_3\neq s_4$ **THEN**

Add the transition (s_1, u_1, λ, φ, (s_3, s_4)) in R_1;

Replace the transition (s_1, u, λ', φ', s_4) in R_1 by (s_1, u, λ', $\varphi'\wedge\neg\varphi$, s_4);

Replace each state ($Fail_{TEMP}$, s') by $Fail_{R_1}$;

IF $s\neq s_{0R_1}$ **THEN**

IF $u\in(I_C\backslash I_A)\cup(O_A\backslash O_C)$ **THEN**

IF $\neg Inv(s)\wedge\overrightarrow{\varphi_2}=\emptyset$ **THEN**

Remove the transition (s, u, λ, φ_2, $Fail_{R_1}$);

Replace the transitions (s, u, λ', φ', s') by (s, u, λ', $\neg\varphi_2\wedge\varphi'$, s');

ELSE

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s')\in T_{R_1}, s'\neq Fail_{R_1}$ and $u\in(I_C\backslash I_A)\cup(O_A\backslash O_C)\}$;

IF $Inv(s) \wedge \neg(\overleftarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi}) \neg Inv(s))^{\emptyset} = \emptyset$ **THEN**

Remove all the outgoing transitions from s;

Replace s by $Fail_{R_1}$;

$R_1 := CC(R_1)$;

ELSE

Remove the transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

Replace the transitions $(s, u, \lambda', \varphi', s')$ by $(s, u, \lambda', \neg\varphi_2 \wedge \varphi', s')$;

$Inv(s) = Inv(s) \wedge \neg(\overleftarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi}) \neg Inv(s))^{\emptyset}$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **DO**

IF $CONSTR(\lambda') \wedge Inv(s) \neq \emptyset$ **THEN**

IF $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **THEN**

Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Add in R_1 the transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

ELSE Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \emptyset, \varphi', Fail_{R_1})$;

ELSE

CONSTRANS$=\{\varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ and $u \in (I_C \setminus I_A) \cup (O_A \setminus O_C)\}$;

$\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overleftarrow{\varphi_2}^{\emptyset} \wedge \neg \varphi_2)$;

IF $Inv(s) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset}) = \emptyset$ **THEN**

Remove all the outgoing transitions from s;

Replace s by $Fail_{R_1}$; $R_1 := CC(R_1)$;

ELSE

Delete the transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

$Inv(s) := Inv(s) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset})$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **DO**

IF $CONSTR(\lambda') \wedge Inv(s) \neq \emptyset$ **THEN**

IF $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **THEN**

Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Add in R_1 the transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

ELSE Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \emptyset, \varphi', Fail_{R_1})$;

ELSE

IF $u \in (I_C \backslash I_A) \cup (O_A \backslash O_C)$ **THEN**

IF $\neg Inv(s) \wedge \overrightarrow{\varphi_2} = \emptyset$ **THEN**

Remove the transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

Replace the transitions $(s, u, \lambda', \varphi', s')$ by $(s, u, \lambda', \neg\varphi_2 \wedge \varphi', s')$;

ELSE

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ and $u \in (I_C \backslash I_A) \cup (O_A \backslash O_C)\}$;

IF v_0 satisfy $Inv(s) \wedge \neg(\overleftarrow{\overrightarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi})}^{\emptyset} \neg Inv(s))$ **THEN**

Remove the transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

Replace the transitions $(s, u, \lambda', \varphi', s')$ by $(s, u, \lambda', \neg\varphi_2 \wedge \varphi', s')$;

$Inv(s) = Inv(s) \wedge \neg(\overleftarrow{\overrightarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi})}^{\emptyset} \neg Inv(s))$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **DO**

IF $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **THEN**

Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Add in R_1 the transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

ELSE return " NO SOLUTION"; **STOP**;

ELSE

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ and $u \in (I_C \backslash I_A) \cup (O_A \backslash O_C)\}$;

$\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overleftarrow{\overrightarrow{\varphi_2} \wedge \neg \varphi_2}^{\emptyset})$;

IF v_0 satisfy $Inv(s) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\overrightarrow{\varphi_2}}^{\emptyset})$ **THEN**

Delete the transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

$Inv(s) := Inv(s) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\overrightarrow{\varphi_2}}^{\emptyset})$;

thigten all the constraints in R_1;

FOR each ingoing transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **DO**

IF $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **THEN**

Replace $(s', u', \lambda', \varphi', s)$ by $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Add in R_1 the transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

ELSE return " NO SOLUTION"; **STOP**;