PLC-Automaton Composition

Honours Project: 1999

André Renaud Dr. Padmanabhan Krishnan [1]

[1]Supervisor

Abstract

Based on a discussion of timed automata and a subset of these called PLC automata, described by H. Dierks, the problem of the composition of PLC automata is presented. Two methods for performing this composition are described, synchronous and asynchronous.

The synchronous method requires that if and only if both initial automata accept a timed word, then it will appear in the resultant automaton. However, we prove that PLC automata are not closed under this form of composition. From this we introduce the asynchronous composition, which requires synchronization on the symbol element of the timed word alone. Four possible methods of asynchronous composition are described, and their effects on the 'essential' language, the language containing only words with no ignored symbols, are discussed. A few elementary results are also presented.

Contents

1 Introduction **3**
1.1 Report Structure 4

2 Preliminaries **5**
2.1 Timed Automata 5
2.1.1 Automata Run 5
2.1.2 Accepting Runs 6
2.2 PLC Automata 7
2.2.1 Graphical Notation 8
2.2.2 Acceptance Criteria 9
2.2.3 Conversion to Timed Automata 9

3 Composition: Definition and Examples **11**
3.1 Synchronous Composition 11
3.1.1 Augmentation 11
3.1.2 Synchronous Composition of Timed Automata 12
3.2 Asynchronous Method 13
3.2.1 Structure 13
3.3 Example 14
3.3.1 Initial Automata 14
3.3.2 Composite Automata 14

4 Languages **17**
4.1 Essential Language 17
4.2 Synchronous Composition Closure 17
4.3 Comparison of languages 19
4.3.1 Minimum, $\cap$ 19
4.3.2 Maximum, $\cup$ 21
4.3.3 Minimum, $\cup$ 22
4.3.4 Maximum, $\cap$ 24
4.3.5 Language Conclusions 25

5 Conclusion **26**
5.1 Further Work 26

Chapter 1

Introduction

Real time systems are currently used in a wide variety of situations. Most consumer devices are required to have real time response to input, and many critical systems are also required to have deterministic response times to avoid failure. When designing these systems a simple modelling technique is highly useful. Such a modelling system was described by Rajeev Alur and David Dill [AD94], in which they discuss an extension to standard automata which allows restricting transitions based on the values of multiple timers. However these timed automata proved to be too general, and so a simplified model, PLC automata, was defined by Henning Dierks [Die97]. These PLC automata are a simplification of timed automata, allowing only one timer, and limiting the timer restrictions to a form of delay for symbols.

When dealing with these systems, a large problem is that of complexity. A complex structure may be difficult to build monolithically, and once built it may contain errors due to its innate complexity. One standard method of dealing with this complexity is composition, combining various sub-parts which contain individual desirable properties into the larger whole, which will hopefully exhibit the required properties, without introducing any additional problems.

We will consider two distinct methods for composing PLC automata. In Section 3.1 a method for synchronous composition will be discussed. An asynchronous method, with four different variants will be considered in Section 3.2.

Synchronous Method

For standard automata we are normally interested in the intersection composition. This is where all symbols occur simultaneously within both sub-automata. However with timed automata, there are two separate elements to each transition, a symbol and a time. Thus we can have compositional methods which are synchronized on either one or both of these elements. The synchronous composition method will attempt to synchronize on both of these, thus requiring the resultant timed word to be accepted by both sub-automata.

Asynchronous Method

Conversely to the synchronous method, which attempted to use both elements of a timed word, the asynchronous method will attempt to use only the sym-

bol, not the timing information. Thus the resultant automaton will accepted the intersection of the untimed languages of the original automata. We will consider the properties of the timing constraints associated with the composite automaton. The asynchronous method is a structure based method, in which we maintain a PLC structure, and attempt to create logical values fitting with the original system.

1.1 Report Structure

Chapter 2 will present an introduction to the foundation work in this area, precisely defining both real time automata and PLC automata, along with their behaviour. We will discuss both timed and PLC automata, and demonstrate a method for converting from PLC automata to timed automata. This chapter will be used as the basis for the additional research conducted for this report.

In Chapter 3 methods for composing automata in both the synchronous and asynchronous methods are discussed. We will also present an example of each of the four different asynchronous techniques, and an explanation of how this composite result relates to the original automata.

In Chapter 4 a simplified version of the languages used in the composition, called the 'essential' language is described. Using this language we will discuss the properties satisfied by the various methods of composition techniques.

Finally, Chapter 5 presents out conclusions, and describes some ideas for further development.

Acknowledgements

I would like to thank Dr. Padmanabhan Krishnan for assisting and guiding me with this research, as well as Jane McKenzie for proof reading this paper many times.

Chapter 2

Preliminaries

2.1 Timed Automata

An introduction to timed automata [AD94], describes them as an extension to finite automata. Formally an automaton is a tuple $\mathcal{A} = (\Sigma, S, S_0, C, E)$ where:

- Σ is a finite alphabet,
- S is a finite set of states,
- $S_0 \subseteq S$ is a set of initial states,
- C is a finite set of clocks,
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ gives the set of transitions. An edge $(s, s', \alpha, \lambda, \delta)$ represents a transition from state s to state s' on input symbol α. The set $\lambda \subseteq C$ gives the clocks to be reset with this transition, and δ is a clock constraint over C.

The transitions within timed automata require the satisfaction of clock constraints. We will define a set $\Phi(X)$, to be a set of clock constraints δ. These constraints are defined inductively by:

$$\delta := x \leq c | c \leq x | \neg\delta | \delta_1 \wedge \delta_2$$

where x is a clock in X, and c is a constant in $\mathbb{Q}$.

However to keep the notation simple and consistent, the following notation, which is similar to standard automata notation, will be used; S will be called Q, and S_0, which typically contains only one element, will be represented by π_0/q_0. An example of a timed automaton is displayed in Figure 2.1. This shows a two state automaton with one timer, which accepts strings of alternating a's and b's. Each 'a' must be no more than two time units after the preceding 'b', and each 'b' must be at least one time unit after the preceding 'a'. The clock X is reset on both transitions.

2.1.1 Automata Run

In order to define a run, a few auxiliary concepts are required.

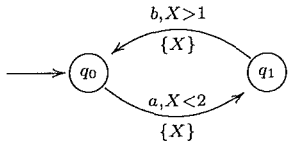


Figure 2.1: Example Timed Automaton

A *clock interpretation* v for a set C of clocks, assigns a real value to each clock; that is, it is a mapping from C to $\mathbb{R}$. We say that a clock interpretation v for C satisfies a clock constraint δ over C if and only if δ evaluates to true using the values given by v. We will define two notational conveniences. $[C \leftarrow \mathbb{R}]$ specifies the clock mapping. $v_i = [\lambda \mapsto r](X)$ will be used to assign all clocks $c \in \lambda$ the value r, after which the equation X will be evaluated, and its result assigned to v_i.

A *time sequence* $\tau = \tau_1\tau_2\tau_3\cdots$ is an infinite sequence of time values $\tau_i \in \mathbb{R}$ with $\tau_i > 0$, satisfying the following constraints:

Monotonicity: τ increases strictly monotonically; that is $\tau_i < \tau_{i+1}$ for all $i \geq 1$.

Progress: For every $t \in \mathbb{R}$, there is some $i \geq 1$ such that $\tau_i > t$.

A *timed word* over an alphabet Σ is a pair (σ, τ), where $\sigma = \sigma_1\sigma_2\ldots$ is an infinite word over Σ and τ is a time sequence. A *timed word* is viewed as an input to an automaton, it presents the symbol σ_i at time τ_i.

We can now define an automata run. A *run* over a timed word, $(\overline{q}, \overline{v})$, of a timed automaton is an infinite sequence of the form

$$r : \langle q_0, v_0\rangle \xrightarrow[\tau_1]{\sigma_1} \langle q_1, v_1\rangle \xrightarrow[\tau_2]{\sigma_2} \langle q_2, v_2\rangle \xrightarrow[\tau_3]{\sigma_3} \cdots$$

with $q_i \in Q$ and $v_i \in [C \leftarrow \mathbb{R}]$, for all $i \geq 0$, satisfying:

- *Initiation:* $q_0 = \pi_0$, and $v_0(x) = 0$ for all $x \in C$
- *Consecution:* for all $i \geq 1$, there is an edge in E of the form $\langle q_{i-1}, q_i, \sigma_i, \lambda_i, \sigma_i\rangle$ such that $(v_{i-1} + \tau_i - \tau_{i-1})$ satisfies σ_i and $v_i = [\lambda_i \mapsto 0](v_{i-1} + \tau_i - \tau_{i-1})$.

2.1.2 Accepting Runs

We will add the notion of acceptance to our timed automata through the use of Büchi, [Tho90], acceptance criteria. Büchi acceptance criteria add an extra element to the timed automaton called F. This is a set of *accepting* states. A run r over a Büchi automaton is accepted if and only if $\inf(r) \cap F \neq 0$, where $\inf(r)$ is defined as all states visited infinitely often during the run r. Graphically we will denote these states as a double circle, as in Figure 2.2

This acceptance criteria implies that a run is accepted if and only if it visits one or more of the accepting states infinitely often. So in the example in Figure 2.2 a run must contain an infinite number of b's to be accepted. It may or may not contain an infinite number of a's.

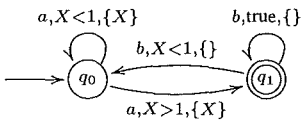


Figure 2.2: Example Acceptance Automaton

2.2 PLC Automata

PLC automata, as described by Henning Dierks [Die97], are a model designed to express the behaviour of polling systems. They are a subclass of timed automata, being restricted to only one clock, and certain types of clock restrictions. Formally they are an automaton defined as a tuple, $\mathcal{A} = (Q, \Sigma, \delta, \pi_0, \varepsilon, S_t, S_e, \Omega, \omega)$, where

- Q is a nonempty, finite set of *states*.
- Σ is a nonempty, finite set of *inputs*.
- δ is a transition function of type $Q \times \Sigma \rightarrow Q$
- $\pi_0 \in Q$ is the initial state.
- $\varepsilon > 0$ is the *upper bound* for a cycle.
- S_t is a function of type $Q \rightarrow \mathbb{R}_{\geq 0}$.
- S_e is a function of type $Q \rightarrow \mathcal{P}(\Sigma)\backslash\{0\}$, assigning to each state a set of *delayed inputs* that cause no change of the state during the first time units the automaton stays in this state.
- Ω is a nonempty, finite set of *outputs*.
- ω is a function of type $Q \rightarrow \Omega$ (*output function*).

The S_t function assigns to each state π a *delay time*. This delay time is used to determine the duration for which a specified set of inputs (defined in S_e) are ignored. Given $S_t(q_i) = 3.5$, any symbol $\in S_e(q_i)$ arriving whilst the automaton is in state q_i, and has been there for < 3.5 time units, will be ignored. This will not reset the clock, so if $\alpha \in S_e(q_i)$ arrived at time 3.4, and was ignored, then a second α one time unit later will be visible.

Within the concepts discussed in this paper we will ignore the ε value, which relates to the time it takes to poll the inputs, as well as both ω and Ω, as we are not interested in the output functionality. This simplification does not remove the ability of our restricted PLC automata to model the polling property of systems.

It should also be noted that S_t is not a partial function, that is, every state has some transition on every input. We will relax this slightly and only require that each state $q_i \in Q$ has a transition on every symbol in $S_e(q_i)$, that is, every symbol that can be ignored at a state for a period must eventually have the possibility of a transition from that state. PLC automata also have a deterministic requirement, and are unable to cope with λ (null) transitions.

2.2.1 Graphical Notation

We will use the following graphical notation to represent a particular state of a PLC automaton:

$$\left[\frac{\pi}{D, \{s_1 \dots s_n\}}\right]$$

where:

- $\pi \in Q$ is a particular state.
- $D = S_t(\pi)$, the delay for the current state.
- $\{s_1 \dots s_n\} = S_e(\pi)$, the set of inputs which can be ignored within the timing constraints.

An example of a PLC automaton using this graphical notation is given in Figure 2.3. Informally, state q_0 is sensitive to a and b after 5 time units, while it is always sensitive to c. However state q_2 is always sensitive to a and b, but is sensitive to c only after 4 units of time.

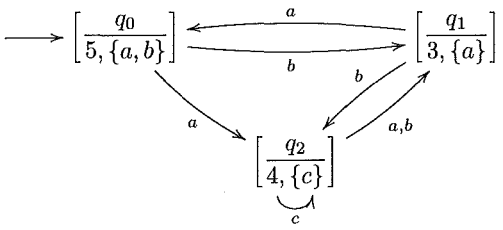


Figure 2.3: Example PLC Automaton

Sample Run

Using the automaton shown in Figure 2.3 we will now describe a sample run, demonstrating how the timing constraints work within PLC automata.

$$(a, 4) \mapsto (a, 6) \mapsto (b, 8) \mapsto (a, 9) \mapsto (a, 12) \dots$$

Figure 2.4: Sample Run of PLC Automaton

In the sample run shown in Figure 2.4, we start in state q_0. The 'a' at time 4 will be ignored, as it falls before the settling time of 5 for the state q_0. The 'a' at time 6 will now be acknowledged, and the transition to state q_2 is made. The relative clock is now 0, as it is reset on every transition, and thus the 'b' at time 8 has a relative time for state q_2 of 2. Since 'b' is not in the ignored set for state q_2, the transition is made to state q_1. Again the relative clock is now 0, so the transition on 'a' at time 9 is inactive, as it has a relative timing of 1, which is less than the delay time of state q_1, 3. The 'a' at time 12 has a relative clock valuation of 4, and is thus active, taking us back to state q_1.

2.2.2 Acceptance Criteria

For demonstrative purposes, we will define a simple acceptance criterion, which will allow us to display acceptable and unacceptable runs of an automaton. We will first define the timed word. Consider tw, a timed word, such that

$$tw : \omega \rightarrow (\Sigma \times \mathbb{R}^+)$$

Given an automata $\mathcal{A}$, a run of $\mathcal{A}$ over tw is

$$\rho : \omega \rightarrow Q \text{ such that } \rho_0 = \pi_0, \rho_i \overset{\substack{tw^1(i) \\ tw^2(i)}}{\longmapsto} \rho_{i+1}$$

These timed steps are defined as

$$q \overset{a}{\underset{t}{\longmapsto}} q'$$

iff

- $S_t(q) \geq t, a \in S_e(q), q = q' \implies a$ ignored *or*
- $S_t(a) < t, \exists q \xrightarrow{a} q' \in \delta \implies$ delay expired *or*
- $a \notin S_e(q), \exists q \xrightarrow{a} q' \implies$ "urgent" action

2.2.3 Conversion to Timed Automata

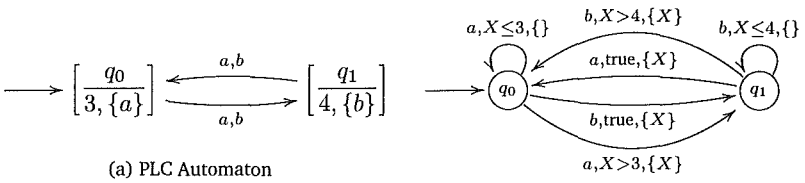
We would like to be able to use many of the properties of timed automata within our PLC automata; as such we will define a method for converting between the two, [DFMV98]. This conversion is fairly straightforward.

Given a PLC-Automaton A_1, such that, $A_1 = (Q_1, \Sigma_1, \delta_1, \pi_{0_1}, S_{t_1}, S_{e_1})$, the equivalent timed $A_2 = (Q_2, \Sigma_2, q_{0_2}, C_2, E_2)$ can be formed as follows:

- $\Sigma_2 = \Sigma_1$
- $Q_2 = Q_1$
- $p_{0_2} = \pi_{0_1}$
- $C_2 = \{X\}$, a new clock
- $\forall q_i \in Q_1, \forall \alpha_i \in S_{e_1}(q_i), (q_i, q_i, \alpha_i, X \leq S_{t_1}(q_i), \emptyset) \in E_2$
- $\forall q_i \in Q_1$,
  $\forall \alpha_i$ s.t $\delta_1(q_i, \alpha_i) \rightarrow q_j, \begin{cases} (q_i, q_j, \alpha_i, X > S_{t_1}(q_i), \{X\}) \in E_2 & \alpha_i \in S_{e_1}(q_i) \\ (q_i, q_j, \alpha_i, \mathit{true}, \{X\}) \in E_2 & \alpha_i \notin S_{e_1}(q_i) \end{cases}$

The final rule above shows the creation of transitions from the delayed and non-delayed symbols. If a state has a delay time of $S_{t_1}(q_i)$, then the transition requires the clock X to be greater than the delay on any symbols in $S_{e_1}(q_i)$. However if the symbol is not in the set, then it can be instantly moved on with no clock restriction.

An example of this conversion is shown in Figure 2.5. Figure 2.5(a) shows the original timed automaton. Figure 2.5(b) shows the subsequent timed automata generated from the method shown above. The self loop on q_0 corresponds to the ignored a, while the self loop on q_1 corresponds to the ignored b.



(a) PLC Automaton

(b) Timed Automaton

Figure 2.5: Example Conversion from a PLC Automaton to a Timed Automaton

Chapter 3

Composition: Definition and Examples

In this chapter we will examine various methods of combining PLC automata. The first, and simplest, is the synchronous product, over perhaps different alphabets.

3.1 Synchronous Composition

We will define the synchronous composition as the intersection of two automata. The method we will use for this composition will require us to convert the PLC-Automata models into standard Timed Automata. We will then synchronously compose these timed automata, [AD94]. Finally we will attempt to take the resultant composite timed automaton, and convert it back into a PLC automaton.

3.1.1 Augmentation

When composing two PLC automata we will require that $\Sigma_1 = \Sigma_2$. If this is not the case, we can augment the alphabets of the individual automaton using the following method.

Augment both automata with the symbols which are unique to the other.

$$\forall \alpha, \alpha \in \Sigma_1, \alpha \notin \Sigma_2 \implies \forall q_i \in Q_2, (q_i \times \alpha \rightarrow q_i) \in \delta_2^{'}$$

Similarly, the reverse:

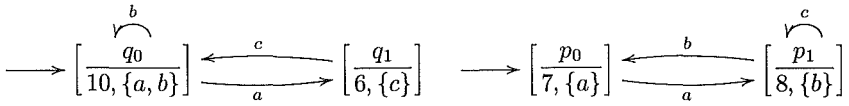
$$\forall \alpha, \alpha \in \Sigma_2, \alpha \notin \Sigma_1 \implies \forall q_i \in Q_1, (q_i \times \alpha \rightarrow q_i) \in \delta_1^{'}$$

The result is that when one automaton makes a move about which the other has no knowledge, i.e.: it does not recognise the symbol, the move does not involve a state change or timer reset. This allows the automaton to see as many 'unknown' symbols as we like, without affecting its interaction with the known alphabet.

3.1.2 Synchronous Composition of Timed Automata

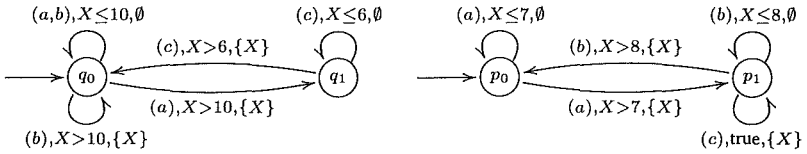
We would like to be able to compose two PLC automata in a synchronous fashion to produce a new automaton with desirable properties, such as word acceptance or language intersection.

Given the two automata described in Figure 3.1, their equivalent timed automata, Figure 3.2, can be determined using the method described above.



(a) Automaton 1 (b) Automaton 2

Figure 3.1: Initial PLC-Automata



(a) Timed Automaton 1 (b) Timed Automaton 2

Figure 3.2: Equivalent Timed Automata

The synchronous composition of these two automata, presented as a timed automata, is shown in Figure 3.3. Converting this back to a PLC automaton now requires each transition to be considered individually. Transitions of the form '$\alpha, X \oplus n \wedge Y \oplus n, S$', and '$\alpha, Z \oplus n, S$', where $\oplus$ ranges over $\{>, \leq\}$, and S ranges over $\{\{X, Y\}, \emptyset\}$, are easily translated back to a PLC automaton. This is because the final PLC automaton can have only one clock, and transitions of these forms are easily converted into transitions of only one clock, suitable for PLC automaton. However transitions which involve the reseting of only one clock, or have restrictions on both clocks which are not identical, cannot be converted into restrictions on only one clock. Thus the composite result cannot be converted back from a timed automaton to a PLC automaton. Because it is now impossible to successfully compose two PLC automata in a synchronous fashion, we will look at some of the alternative methods for asynchronous composition. From this we will accept Lemma 1.

Lemma 1 (Synchronous Composition) *There exist PLC automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that there is no PLC automaton that accepts $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$*

Proof: Examples in Figures 3.1–3.3, and above discussion. ■

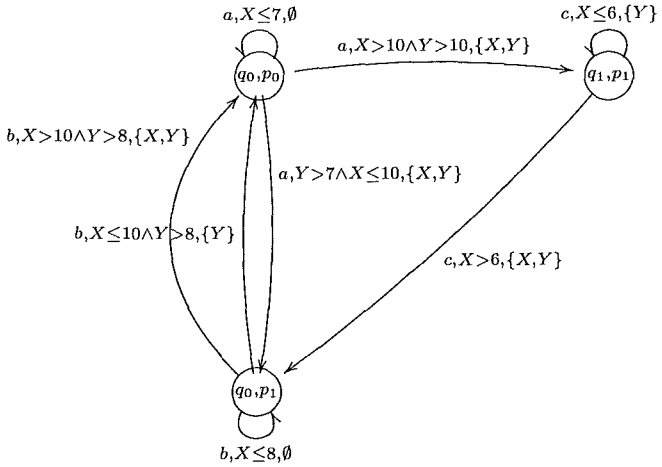


Figure 3.3: Composite Timed Automata

3.2 Asynchronous Method

As the previous examples showed, it does not appear possible to compose PLC automata in a synchronous manner. Thus the construction of the language intersection is not possible. As an alternative we will look at methods which require the retainment of the PLC structure, and observe the effects they have on the combined language.

In this section we will look into methods for structure based asynchronous composition, demonstrating these methods with a brief example.

Given any two automata, $A_1 = (Q_1, \Sigma_1, S_{t_1}, S_{e_1})$, and $A_2 = (Q_2, \Sigma_2, S_{t_2}, S_{e_2})$, whose states are disjoint, i.e.: $Q_1 \cap Q_2 = \emptyset$, the composition, to obtain $A_3 = (Q_3, \Sigma_3, S_{t_3}, S_{e_3})$, is as follows.

3.2.1 Structure

The states of the new automaton, A_3, will be the product of the states of the two initial automata, A_1 and A_2. Thus $Q_3 = Q_1 \times Q_2$.

$$(q_1,q_2) \xrightarrow{a} (q_1',q_2')$$
$$\text{iff } q_1 \xrightarrow{a} q_1' \text{ and } q_2 \xrightarrow{a} q_2'$$

S_t and S_e composition

There are two different ways of composing each of the S_t and S_e arrays.

For S_e, either the *intersection* or *union* is used.

$$\forall q = (q_i, q_j) \in Q_3, S_{t_3}(q) = S_{t_1}(q_i) \oplus S_{t_2}(q_j)$$

where $\oplus$ ranges over $\{\cup, \cap\}$.

For S_t, either the *min* or *max* is used.

$$\forall q = (q_i, q_j) \in Q_3, S_{e_3}(q) = S_{e_1}(q_i) \oplus S_{e_2}(q_j)$$

where $\oplus$ ranges over {min, max}.

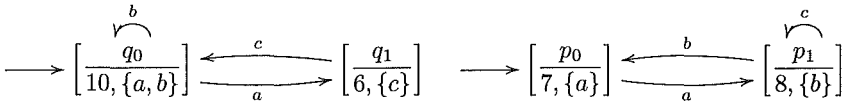
It should be noted that the transition structure does not change between the different methods of composition, only the internal sensors and input sets of each node changes as the 'untimed' languages intersect.

3.3 Example

In this section we will look at a sample composition, showing each of the four proposed methods.

3.3.1 Initial Automata

Consider the two automata shown in Figure 3.4. There is no need to augment either of these, as they share the same alphabet, $\{a, b, c\}$.



(a) Automaton 1 (b) Automaton 2

Figure 3.4: Initial Automata

3.3.2 Composite Automata

Minimum, $\cup$

The automaton displayed in Figure 3.5 is the result of the minimum and $\cup$ composition of the two initial automata. The result here appears to take the 'best' sensor (i.e.: the one with the lowest value), and apply it to all of the potentially unreliable inputs.

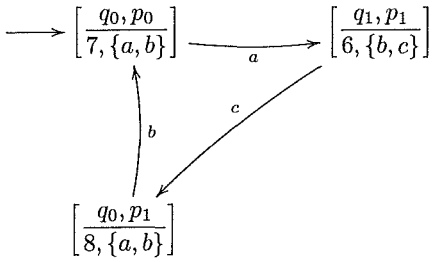


Figure 3.5: Minimum, $\cup$ Composition

Maximum, $\cup$

This composition, shown in Figure 3.6, takes the 'strongest' sensor in every case. It selects the maximum value, thus the slowest sensor, most reliable to all unreliable inputs, and it takes the union of the sets of symbols, thus requiring the largest set of symbols to be restricted. This could be viewed as the most conservative composition; for any given transition it selects the slower/more restrictive option.

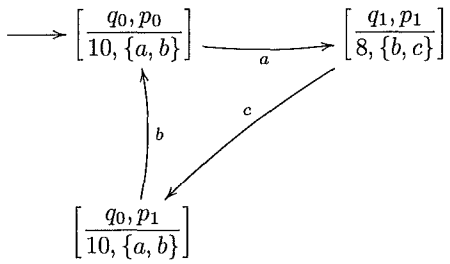


Figure 3.6: Maximum, $\cup$ Composition

Minimum, $\cap$

This composition, shown in Figure 3.7, is, not surprisingly, the opposite of the maximum, union composition. For each composite node, the fastest sensor is chosen, and it is applied to the smallest set of symbols possible (i.e.: the minimum). The result of this is an automaton which delays for as little as possible. If the maximum, union composition was conservative, then this would be the most permissive, containing the fewest restraints. For example in the (q_1, p_1) state, the fastest sensor for b is used; that from q_1, where b is assumed to be instantly stable (i.e.: time delay of 0). Similarly for c, the sensor in p_1 is used, which gives c a delay of 0. The result of this is the empty set, as both b and c have delays of 0, and as such do not appear in the delay set.

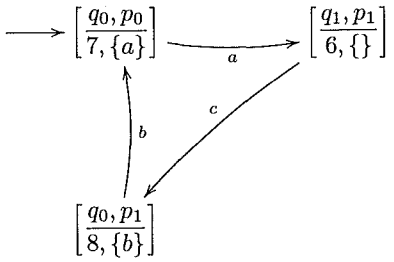


Figure 3.7: Minimum, $\cap$ Composition

Maximum, $\cap$

Shown in Figure 3.8, this composition is similar to the first, minimum, union. It selects the slowest sensor, but only attaches it to the smallest number of elements. The result is an automaton which is slow on symbols that were common to previous timing constraints, but ignores symbols that were from a single automaton only.

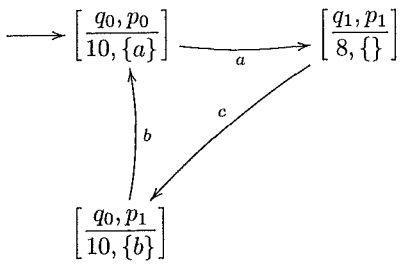


Figure 3.8: Maximum, $\cap$ Composition

Thus concludes the definition of the structure based composition of PLC automata. While they do not exactly correspond to the standard product construction, there are a number of similarities. In the next chapter we study the language behaviour of the composite automata.

Chapter 4

Languages

4.1 Essential Language

For any PLC-Automata, $\mathcal{A}$, we define the language of words it can accept as $\mathcal{L}(\mathcal{A})$. Within this language $\mathcal{L}(\mathcal{A})$, if every word is collapsed, by removing all ignored transitions and merging identical collapsed words, we get a language $\mathcal{L}(\mathcal{A})^0$, which contains all the minimal runs of the automata. We will call this the essential language. This essential language allows for easier manipulation and comparison, and will be used for the following language comparisons.

Using the sample run previously shown in Figure 2.4, we can demonstrate the effect of employing only the essential language. Initially we had a transition on 'a' at time 4. This was ignored and therefore it will not appear in the essential version of this run. Removing the ignored symbols, the sample run now becomes that shown in Figure 4.1.

$$(a, 6) \mapsto (b, 8) \mapsto (a, 12) \ldots$$

Figure 4.1: Sample Run for Essential Language

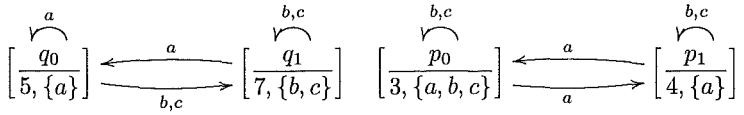
It should be noted that for every word in the standard language of an automaton, there is a word in the essential language which is equivalent, except that all ignored symbols have been removed. Thus there is an $N : 1$ relationship between the two sets.

4.2 Synchronous Composition Closure

As shown in Section 3.1.2, we are unable to build a new PLC automaton which accepts the intersection of the languages of the two initial automata. However if we consider the essential language, rather than the standard language, a slightly modified construction can now be attempted.

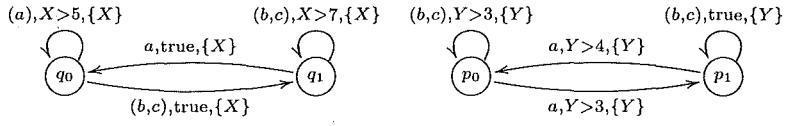
In the modified construction transitions will no longer be allowed to occur during periods of ignored time for the PLC states. This will not have any influence on the essential language, as these transitions would have been invalid

anyway. We now only produce transitions of symbols which are non-ignored. This will result in transitions only being bounded by $>$ as a timing constraint, since $\leq$ was used when symbols occurred before their delay time, which is now impossible.



(a) Automaton 1 (b) Automaton 2

Figure 4.2: Initial PLC Automata



(a) Automaton 1 (b) Automaton 2

Figure 4.3: Resultant Timed Automata

In the resultant timed automata, shown in Figure 4.3, we can see that the conversion has added no additional transitions, unlike the case for the original conversion. These extra transitions were all due to symbols arriving before their delay time had elapsed, which no longer occurs.

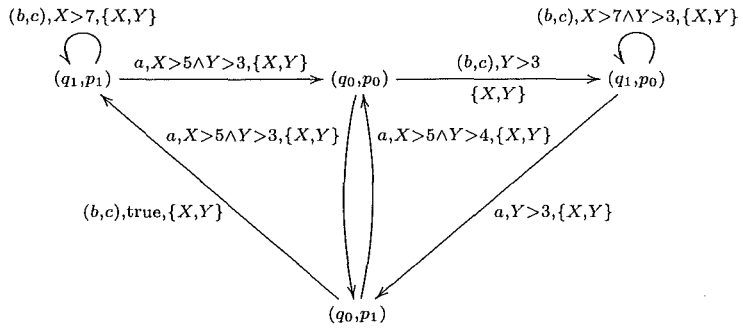


Figure 4.4: Synchronous Composite Automaton

The final composite automaton, shown in Figure 4.4, with each state having only one transition on any given symbol, is now much simpler than the composite result obtained by the general method. We would like to be able to convert this back to a PLC automaton. However determining the timing constraints and delayed symbols is non-trivial. From most states there is a choice

between either of the delayed times of the two original combining states. This would lead us to choose either the minimum or maximum of these two values, but the composition would then tend to the asynchronous method. This would also mean one transition was either over, or under-compensated. For example in Figure 4.4, state (q_0, p_0) has two transitions, $(b, c), Y > 3, \{X, Y\}$, and $a, X > 5 \wedge Y > 3, \{X, Y\}$. If we choose the maximum of the two timing constraints, 5, the (b, c) transition is too heavily restricted. However, if we choose 3 the a transition will be too permissive. Determining the delayed symbols is simpler however. If a transition from a state, q_i, has a timing restriction of anything other than true, then the symbol(s) of that transition must appear in the delayed set, $S_e(q_i)$. If the transition has a restriction of true, then that symbol will not appear in the set. This set construction turns out to be union, because if a symbol was delayed in its initial automaton, it must be delayed in the composite result. Thus every delayed symbol from the initial states must be delayed in the composite state.

From this we can conclude that the synchronous essential language composition is of little use. It is restricted to the essential language, and can thus not be generalized, so its ability to model standard PLC automata is diminished. It is also highly similar to two of the asynchronous methods ($\{\min, \max\}, \cup$), and as these are more general methods, which still work on the entire language, they are more useful compositions.

4.3 Comparison of languages

For the purposes of this report, we are interested in comparing the resultant languages from the asynchronous composition methods discussed earlier, with the intersection of the two languages from the original automata used in the composition.

4.3.1 Minimum, $\cap$

We are interested in the status of the following class of conjectures.

Conjecture 1 (Minimum, $\cap$ Comparison)

$$L^0_{min,\cap} \square L^0_{\cap} \textit{ where } \square \textit{ could range over } \{=, \subset, \subseteq, \supset, \supseteq\}$$

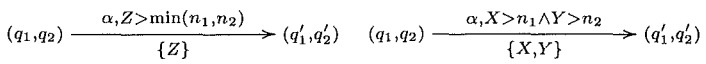
$$\left[\frac{q_i}{n_i, S_i}\right] \xrightarrow{\alpha} \left[\frac{q_i'}{n_i', S_i'}\right], i = 1, 2$$

Figure 4.5: General PLC Automaton

In comparing the language of the intersection, which as shown in Section 3.1 cannot be represented as a PLC automaton, with the language of the asynchronous composition, discussed in Section 3.2, consider the general PLC transition, as shown in Figure 4.5. Both will be converted into standard timed automata and the languages compared. When performing this conversion, the three scenarios that must be investigated are:

- $\alpha \in S_1 \wedge \alpha \in S_2$
- $\alpha \in S_1 \wedge \alpha \notin S_2$
- $\alpha \notin S_1 \wedge \alpha \notin S_2$

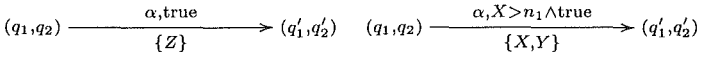
We will thus construct the two composite transitions for the three above scenarios. These are displayed in Figures 4.6, 4.7 and 4.8 respectively.



(a) Asynchronous (b) Synchronous

Figure 4.6: $\alpha \in S_1 \wedge \alpha \in S_2$

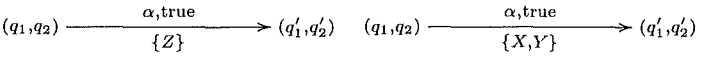
From the first of these, $\alpha \in S_1 \wedge \alpha \in S_2$, shown in Figure 4.6, we can see that there are no transitions accepted by the synchronous result, that are not accepted by the asynchronous one. For example, given $n_1 = 5 \wedge n_2 = 6$, the composite conditions become $Z > 5$ and $X > 5 \wedge Y > 6$ respectively. A time step of $(\alpha, 5.1)$ would now be permitted by the first transition, but not by the second.



(a) Asynchronous (b) Synchronous

Figure 4.7: $\alpha \in S_1 \wedge \alpha \notin S_2$

Similarly to the first minimum, $\cap$, scenario, we can see that in the behaviour shown in Figure 4.7, the asynchronous transition will be enabled on all clock valuations on α, as its clock constraint is true. However, there is still a constraint on the synchronous composition, $X > n_1$. Thus the synchronous method must be delayed by a minimum of n_1, whereas the asynchronous transition is always enabled.



(a) Asynchronous (b) Synchronous

Figure 4.8: $\alpha \notin S_1 \wedge \alpha \notin S_2$

In the final of the three minimum, $\cap$, scenarios, shown in Figure 4.8 we find both automata accepting all timing conditions of α. When the semantics of this final composition are considered this is not surprising, as this is the case when α is not in either of the restricted sets, and is thus completely unrestricted.

From the three scenarios for minimum, $\cap$, we can see that overall, the asynchronous composition would always accept at least all of the words accepted by the synchronous composition. Further to this we can see that in the first two cases, Figures 4.6 and 4.7, there are additional words accepted by the asynchronous method, as the timing constraints on the synchronous transitions are more strict. From this Conjecture 1 can be completed.

Lemma 2 (Minimum, $\cap$, Language Comparison)

$$L^0_{\min,\cap} \supset L^0_{\cap}$$

Proof: Let $(\alpha_0, t_0)(\alpha_1, t_1)\ldots(\alpha_n, t_n)\ldots \in L^0_{\cap}$ be the given word.
$\therefore$ There is a run of $\mathcal{A}_1 \parallel \mathcal{A}_2$, the synchronous composition, over it. By observation, and the constructions in Figures 4.6 – 4.8, this run can be converted to a run of $\mathcal{A}_1 \cap_{\min,\cap} \mathcal{A}_2$ over the given word.
It should be noted that this conversion only works because in PLC automata, every non-ignored transition resets the clock. Thus when contemplating the subsequent transition we know that the clock is zero, and it can be examined independently of previous results.

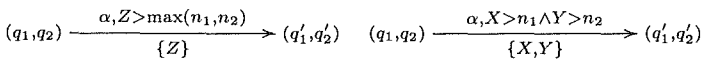
4.3.2 Maximum, $\cup$

Using the same technique as above, the state of the following conjecture will be investigated:

Conjecture 2 (Maximum, $\cup$ Comparison)

$$L^0_{\max,\cap} \sqsubset\!\sqsupset L^0_{\cap} \textit{ where } \sqsubset\!\sqsupset \textit{ could range over } \{=, \subset, \subseteq, \supset, \supseteq\}$$

We will again consider the three case scenarios described in Section 4.3.1.

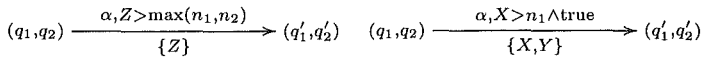


(a) Asynchronous (b) Synchronous

Figure 4.9: $\alpha \in S_1 \wedge \alpha \in S_2$

For the first case, where $\alpha \in S_1 \wedge \alpha \in S_2$, the composite transitions are as described in Figure 4.9. This differs from that described in the minimum, $\cap$, composition, in that the restrictions on both transitions are now equivalent. The asynchronous restriction, $Z > \max(n_1, n_2)$, and the synchronous, $X > n_1 \wedge Y > n_2$, both require that the larger of the two timing constraints, $\{n_1, n_2\}$, is satisfied before the transition is active.

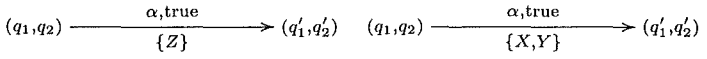
The second scenario, $\alpha \in S_1 \wedge S \notin S_2$, shown in Figure 4.10, is a slightly different case to those seen before. In this case, we have the most restrictive conditions on the asynchronous transition, $Z > \max(n_1, n_2)$, whilst the restriction on the synchronous transition, $X > n_1$, would appear to be weaker. However there are now two possibilities:



(a) Asynchronous (b) Synchronous

Figure 4.10: $\alpha \in S_1 \wedge \alpha \notin S_2$

- $n_1 < n_2$ – The synchronous composition can now accept additional time steps, any of whose clock valuations are between $(n_1, n_2]$.
- $n_1 \geq n_2$ – The transitions are now equivalent, as $\max(n_1, n_2) = n_2$. There are no additional time steps accepted by either transition.



(a) Asynchronous (b) Synchronous

Figure 4.11: $\alpha \notin S_1 \wedge \alpha \notin S_2$

In the final of the maximum, $\cup$, composition scenarios, shown in Figure 4.11, we have the trivial case, where α is not an ignored symbol. Since both of the restrictions are true, there is no difference in the activation properties of the transition; both accept the same timing constraints.

From the three cases of the maximum, $\cup$, we can see that in the first and third case, the restrictions were equivalent between asynchronous and synchronous. However, in the second case, $\alpha \in S_1 \wedge \alpha \notin S_2$, there was a possibility for additional transitions to be active, as long as $n_1 < n_2$. Thus, in general, Conjecture 2 can be adapted into the Lemma below.

Lemma 3 (Maximum, $\cup$, Language Comparison)

$$L^0_{\max,\cup} \subset L^0_{\cap}$$

Proof: As shown in the above constructions (Figures 4.9 – 4.11) ■

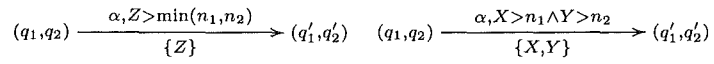
4.3.3 Minimum, $\cup$

Similarly to above we will now examine the essential language for the minimum, $\cup$, composition.

Conjecture 3 (Minimum, $\cup$ Comparison)

$$L^0_{\min,\cup} \square L^0_{\cap} \textit{ where } \square \textit{ could range over } \{=, \subset, \subseteq, \supset, \supseteq\}$$

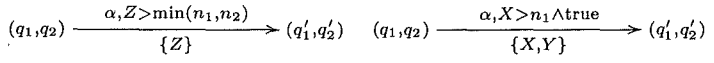
The first of the three scenarios for minimum, $\cup$, is shown in Figure 4.12. It can be seen that it is identical to the first scenario for the minimum, $\cap$, as the determining factor here is the minimum function. The set combining element



(a) Asynchronous (b) Synchronous

Figure 4.12: $\alpha \in S_1 \wedge \alpha \in S_2$

$(\cup, \cap)$ only affects the status of the second scenario. We can see here that, as in the minimum, $\cap$, example, there are additional time steps accepted by the asynchronous composition; any steps whose clock valuation occurs between $\min(n_1, n_2)$ and $\max(n_1, n_2)$.

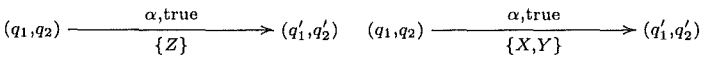


(a) Asynchronous (b) Synchronous

Figure 4.13: $\alpha \in S_1 \wedge \alpha \notin S_2$

In the second scenario, $\alpha \in S_1 \wedge \alpha \notin S_2$, shown in Figure 4.13, the situation is similar to the second scenario for maximum, $\cup$. However, we now have the minimum, not maximum, and thus the timed step behaviour is reversed. There are now two possible cases:

- $n_1 \leq n_2$ – The synchronous transition has an identical constraint to the asynchronous, and the timed step acceptance is equivalent.
- $n_1 > n_2$ – The synchronous transition is more restrictive than the asynchronous, and, as such, timed steps whose clock valuation falls between (n_1, n_2) will now be active on the asynchronous, but not on the synchronous.



(a) Asynchronous (b) Synchronous

Figure 4.14: $\alpha \notin S_1 \wedge \alpha \notin S_2$

In the third scenario, displayed in Figure 4.14, we again have the trivial case, in which α is a non-ignored symbol, and, as such, the timed step acceptance is equivalent among the two transitions.

For the minimum, $\cup$, composition, we now have two scenarios in which it is possible for the asynchronous composition to allow additional timed steps. In one of these two, the case in the second scenario in which n_1 was the smallest clock delay, there is the possibility for the timed step acceptance to be identical.

In the final scenario, we have the trivial case in which timed step acceptance is again identical. Thus we can complete Conjecture 3 to that displayed in Lemma 4.

Lemma 4 (Minimum, $\cup$, Language Comparison)

$$L^0_{\min,\cup} \supset L^0_{\cap}$$

Proof: As shown in the above constructions (Figures 4.12 – 4.14) ■

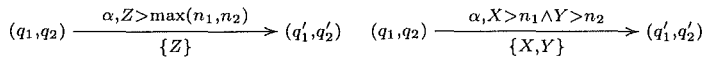
4.3.4 Maximum, $\cap$

For the final compositional method, maximum, $\cap$, we consider the situation where the slowest sensor is chosen, but only on those inputs that are common to both automata. This could be the case where maximum 'safety' was required, but non-ignored symbols were assumed to be perfectly read by the sensor.

Similarly to the other language comparisons we will be considering the status of the following class of conjecture:

Conjecture 4 (Maximum, $\cap$, Comparison)

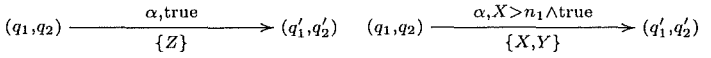
$$L^0_{\max,\cap} \square\, L^0_{\cap} \textit{ where } \square \textit{ could range over } \{=, \subset, \subseteq, \supset, \supseteq\}$$



(a) Asynchronous (b) Synchronous

Figure 4.15: $\alpha \in S_1 \wedge \alpha \in S_2$

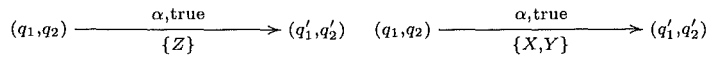
In the first scenario, $\alpha \in S_1 \wedge \alpha \in S_2$, shown in Figure 4.15 the case is identical to that of maximum, $\cup$. Both transitions are active on the same timing constraints, requiring timed steps to have a clock valuation of greater than the maximum of n_1 and n_2.



(a) Asynchronous (b) Synchronous

Figure 4.16: $\alpha \in S_1 \wedge \alpha \notin S_2$

In $\alpha \in S_1 \wedge \alpha \notin S_2$, the second scenario, shown in Figure 4.16, we can see that since the asynchronous transition has no restriction, whilst the synchronous composition has a restriction, $X > n_1$, there are thus timed steps which are active under the asynchronous composition, but not under the synchronous. All timed steps whose clock valuations are less than n_1 will behave differently under the two methods.



(a) Asynchronous (b) Synchronous

Figure 4.17: $\alpha \notin S_1 \wedge \alpha \notin S_2$

As with the other methods, the third scenario, where $\alpha \notin S_1 \wedge \alpha \notin S_2$, is trivial. We can thus see that the timed words active on both transitions would be identical.

For the maximum, $\cap$, compositional language, there is only one scenario in which there is a difference in the timed word activation, similarly with minimum, $\cap$. In the second scenario, the asynchronous transition has no timing constraint, and, as such, the overall language acceptance was more lenient for the asynchronous automaton. We can complete the earlier conjecture to that shown in Lemma 5.

Lemma 5 (Maximum, $\cap$, Language Comparison)

$$L^0_{\max,\cap} \supset L^0_{\cap}$$

Proof: As shown in above constructions (Figures 4.15 – 4.17) ■

4.3.5 Language Conclusions

It is surprising to note that there is only one situation where the asynchronous language is a subset of the intersection language, maximum, $\cup$. This is the most restrictive situation, where the most symbols are ignored for the longest length of time. A method for synchronous composition was considered but was not highly applicable, and should not be used. This was due to both its restriction to the essential language, and its inability to accurately distinguish the timing values, requiring the choice of either the minimum or maximum clock value on each composite state.

The significance of these results is described in the next chapter and further work is proposed to complete the language theory relative associated with polling systems.

Chapter 5

Conclusion

Synchronous composition was initially examined as a model for strict language intersection. However, due to the limitations on the clock constraints within PLC automata, it was shown that PLC automata are not closed under this operation. This led to the development of a structure based asynchronous method, which we examined from a purely intuitive level in Section 3.3. Their similarity in structure, and differences in timing constraints were demonstrated.

For simplicity we defined the essential language, to allow for easier manipulation of the languages involved. Although this did remove some of the generality of the compositional methods, the essential language for PLC automata is very similar to the general languages for standard timed automata. With standard timed automata, symbols which occur outside of their timing restrictions cause the automata to crash. However PLC automata allow these symbols, but simply ignore them. The essential language restricts the PLC automata to accepting only words not containing these ignored symbols. In this model the PLC automata now react in a similar manner to the standard timed automata.

The impact of the essential language on the synchronous composition is that although it is now possible to build a synchronous result, the composition is reduced to a highly restricted version of one of the asynchronous methods.

We showed how the various asynchronous languages compared with the essential version of the intersection language of the initial automata. It was interesting to note that three of the methods resulted in a language which was a superset of the intersection language. However the maximum, $\cup$, method accepted a subset of the intersection language. This can be attributed to its use of the most restrictive conditions in both timing constraints and symbol restrictions. Although this method was identical to the synchronous method in all but one scenario, this was one of the more common scenarios. In this, one state ignores a symbol, but the delay for the second state is larger. Thus in the composite result the symbol is ignored for longer than in either of the initial states.

5.1 Further Work

A very preliminary analysis of a method for complementing PLC automata was made. However it presents several issues, and thus remains incomplete. The

method used was a conversion from PLC Automata to Event-Recording Timed Automata [AFH94], which have been proven to be closed under complementation. We would then complement the resultant event-recording automaton, and attempt to convert it back to PLC automaton. The conversion from PLC–ERA, however was non trivial. This appears to result in an infinite state expansion, because every action, ignored or not, must reset clocks in an ERA. The conversion thus has a problem with ignored symbols, and attempts to count the number of occurrences of an ignored symbol until it is finally past the clock restriction. However, since there can be an arbitrary number of these ignored occurrences, it is impossible to build a finite automaton to model the result. It should be noted that these problems do not occur when using the essential language, as keeping track of ignored symbols is no longer necessary.

Bibliography

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH94] R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *Proc. Sixth International Conference on Computer Aided Verification*, LNCS818, pages 1–13. Springer Verlag, 1994.

[DFMV98] H. Dierks, A. Fehnker, A. Mader, and F. W. Vaandrager. Operational and Logical Semantics for Polling Real-Time Systems. In A. P. Ravn and H. Rischel, editors, *Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume LNCS 1486, pages 29–40, Lyngby, Denmark, 1998. Springer Verlag.

[Die97] H. Dierks. PLC-Automata: A new class of Implementable Real-Time Automata. In M. Bertran and T. Rus, editors, *Transformation Based Reactive Systems Development: ARTS'97*, volume 1231 of LNCS, pages 126–137. Springer-Verlag, 1997.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, pages 133–192. Elsevier, 1990.