A Logical Study of Distributed Timed Automata

THESIS
Submitted in partial fulfilment of the requirements
for the degree of
DOCTOR OF PHILOSOPHY

By

DEEPAK D'SOUZA
Chennai Mathematical Institute, Chennai.

Under the Supervision of
Prof. P. S. THIAGARAJAN
Chennai Mathematical Institute, Chennai.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN) INDIA

January 2000.

Acknowledgements

The work in this thesis would not have been possible without the guidance and encouragement of my advisor Prof. P. S. Thiagarajan. During the course of my Ph. D. I have had many occasions to draw on his resourcefulness, clarity of thought, and his wealth of experience as a researcher. I have learnt from him the virtues of rigour and professionalism, and I am grateful to him for having kept me focussed at the times it mattered most. In short, I thank him for being a model advisor.

I am very grateful to Dr. Madhavan Mukund, who has been in many ways a role model for me. Throughout my stay at CMI I have relied heavily on his advice in technical (and non-technical!) matters and his readiness to help has always been heartening. It has been a pleasure to discuss several topics in this thesis with him, and much of the material presented here has benefited from his comments.

I am grateful to the others in the CS group at CMI, Dr. Narayan Kumar and Dr. K. V. Subrahmanyam for the several discussions I have had with them. Kumar in particular has been very patient in discussing several philosophical aspects of my thesis with me. A special word of thanks is due to P. Madhusudan who has been my course-mate since the start of my Ph. D. I have benefited a great deal from his inquiring mind and maturity of thought. I have also had many useful discussions with A. Srinivasan and K. Sambaiah on several topics in this thesis.

B. Meenakshi has been an immense help to me in preparing this thesis. She has painstakingly gone through each draft and given me very valuable feedback. Thanks Shoba, for all the help, the companionship, and especially for keeping me on course.

I would like to thank Mogens Nielsen and Thomas Hune for an enjoyable summer at BRICS, Aarhus University, where a part of this work was completed.

A special word of thanks to the CS faculty at Matscience and CMI, for the excellent course work they took us through. The close cooperation between Matscience and CMI indeed made for a very conducive working environment for us students.

I would like to thank the administrative staff at CMI—S. Sripathy, V. Vijaylakshmi, Rajeshwari Nair and G. Samson, for all the help they have rendered over the years.

Finally, I would like to dedicate this thesis to my parents, especially my late mother, who would have been the happiest to see this thesis in print.

Deepak D'Souza
Chennai, January 2000.

Birla Institute of Technology & Science
Pilani Rajasthan

Certificate

This is to certify that the thesis entitled "A Logical Study of Distributed Timed Automata" and submitted by Deepak D'Souza (ID. No. 1994PZYF772) for award of Ph. D. Degree of the Institute, embodies original work done by him under my supervision.

Signature in full
of the Supervisor:

Name: P. S. Thiagarajan

Designation: Professor,
Chennai Mathematical Institute,
Chennai.

Date:

Contents

Chapter 1

Introduction

The work in this thesis falls in the broad area of research in computer science known as *formal verification.* Formal verification involves proving mathematically that the design of a system satisfies its specification. The term *system* is used here to mean any kind of algorithmic activity, like a computer program, a traffic signalling system, or a controller in a chemical plant.

Formal verification must be distinguished from the field of *testing*, where one tests the actual implementation of the system under different systematically generated test conditions. As is to be expected, these test cases cannot cover all possible scenarios. Formal verification on the other hand aims to prove that the design of the system is correct, i.e. it satisfies its specification, in *all* possible scenarios. Both techniques must however be seen as complementary activities. Each is applicable in a different phase of system development—verification in the design phase, and testing in the post-implementation stage—and together they contribute to an increased level of confidence in the correct working of the system.

The importance of formal methods in ensuring the correct working of a system cannot be over emphasised. Errors in safety critical systems like railway signalling systems, avionics software, and controllers in nuclear reactors, can be catastrophic in terms of loss of human life and property. Most errors are subtle, difficult to simulate, and hence beyond the scope of human inspection. Furthermore these errors are, more often than not, errors in the *design* of the system. Thus the use of formal methods in the design phase of system development can be a big step towards building correctly functioning systems.

Formal methods in verification can be broadly classified under the two approaches of *theorem proving* and *model checking.* Theorem proving typically involves an axiomatisation of the constructs of a programming language. An assertion about a program is proved by a series of logical deductions, using precisely defined inference rules. This method is only semi-automatic in the sense that while portions of the proof can be carried out by the theorem proving tool, the user must guide the general direction of the proof by laying down sub-goals.

Model checking on the other hand is a fully automatic method of verification. This is largely because it restricts itself to essentially finite state systems. The term "model checking" is derived from the fact that in this method one checks whether the system is a "model" of the given specification, in much the same manner that one checks whether a truth assignment satisfies a given propositional logic formula.

In model checking, the system to be verified is modelled as a transition system comprising states, which can be thought of as a collection of propositions describing that state, and transitions or actions which cause a change from one state to another. Thus, if the system to be modelled is a program, then a state in the model would comprise a description of the current values of the variables and the location counter in the program. For example, the propositions `location-counter = 1` and `request = false` could describe the current state of a program. Each statement in the program, like `request := true` is an action which typically causes a change in the current state of the program.

Specifications in this framework are given in *Temporal Logic*, a specification language originally proposed by Pnueli in [P77]. The operators in temporal logic like "always" and "eventually" have proved to be well suited for specifying safety and liveness properties of systems. A safety property says that something undesirable never happens. For example, a safety property could require that "two processes are never in their critical sections simultaneously." A liveness property on the other hand asserts that the system makes progress: "if a process asks for a resource, then its request is eventually granted."

Temporal logic specifications are essentially interpreted over *runs* of the system. These are valid sequences of states and actions that the system can execute. Temporal logic specifications are usually classified as being either linear time or branching time. In the branching time framework, the behaviour of the system is viewed as a *run tree* which incorporates the branching due to non-deterministic choices made by the system. In the linear time setting, one views the behaviour of the system as comprising the set of valid runs of the system. In particular, we say a system satisfies a linear time temporal logic (LTL) formula φ if every run of the system satisfies φ.

The automata-theoretic approach to model checking introduced by Vardi and Wolper [VW86] allows the model checking problem to be phrased as a purely automata-theoretic problem. This permits the application of well-studied automata-theoretic algorithms to the problem of model checking. In the linear time setting, the system is modelled as a finite state automaton $\mathcal{S}$, which recognises exactly the runs of the system. A temporal logic specification φ can be converted to an automaton $\mathcal{A}_\varphi$ which recognises exactly the the models of φ. Thus the model checking problem "*Does $\mathcal{S}$ satisfy φ?*" can be rephrased as: is $L(\mathcal{S}) \subseteq L(\mathcal{A}_\varphi)$? This is equivalent to checking if $L(\mathcal{S}) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset$. Here $\mathcal{A}_{\neg\varphi}$ is an automaton accepting the models of the negation of φ, and is hence also the complement of the language accepted by $\mathcal{A}_\varphi$.

A pleasant feature of this framework of model checking is its connection to logic. In the early sixties Büchi had showed that language recognition by automata was the same as logical definability by S1S—a monadic second order logic interpreted over words. LTL was later shown to be expressively equivalent to the first-order fragment of S1S. This, along with the fact that it has a decision procedure which is exponential in the size of the formula as against the non-elementary decision procedure for the first-order fragment of S1S, is often seen as an endorsement of LTL as a good specification logic.

1.1 Model checking for timed systems

This thesis is concerned with the application of model checking techniques in the formal verification of *real time* systems. These are systems whose behaviour depends crucially on the times at which external events happen, and not just on the ordering of the events. These systems are widespread in everyday life and range from household goods like washing machines and toasters, to safety critical systems like automobile braking systems and plant controllers. Traditional techniques for model checking are inadequate when it comes to reasoning about such systems, since they typically abstract away from quantitative timing information in the system. System models and logics speak only of an event following another, and not of *how much* later it can happen.

These issues can be brought out through the example of a railroad crossing controller, a popularly cited example of a timed system. Fig 1.1 shows the railroad crossing and an area C around the crossing, called the critical region. The gate controller receives signals, via sensors located on the track, telling it when a train approaches and exits the crossing. The sensors are located so that a train takes at least 2 minutes to reach the critical region C after signalling "approach." Further, when it leaves the crossing, it sends an "exit" signal to the controller. The "exit" signal can occur anytime between 2 to 5 minutes after the train has entered the critical region. Upon receiving the

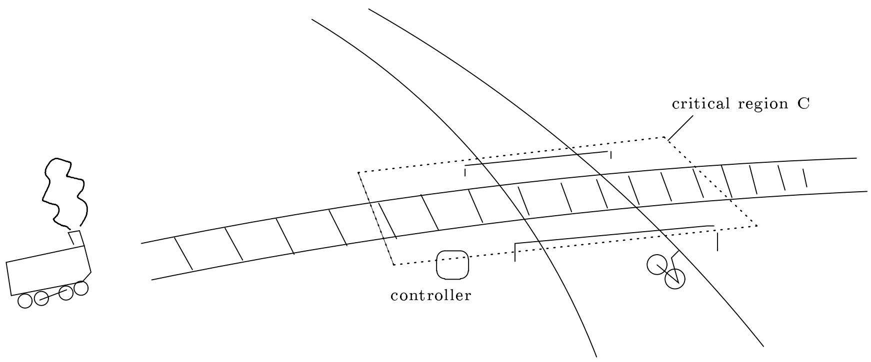

Figure 1.1: Railroad crossing controller.

"approach" signal, the controller must lower the gate. It begins to lower the gate within a minute of receiving the "approach" signal from the train. Once initiated by the controller, the gate takes at most a minute to lower itself completely. A similar sequence of actions takes place when the train leaves the critical area. Upon receiving the "exit" signal from the train, the controller begins to raise the gate. Again there is a delay of at most a minute for the controller to initiate the "raise" action, and for the gate to raise itself completely.

We make the simplifying assumption that there can be at most one train in the system at a given time. Thus a train cannot "approach" the crossing without all trains before it having exited the crossing.

The crucial question we would like to ask of the system is: is it safe? That is, is it the case that whenever a train is in the critical area, the gate is indeed down. Another question we may be interested in is regarding the utility of the system: if there is no train in the region for 10 minutes, is the gate raised to allow traffic to pass during this period?

A natural way to model the system would be to have a separate transition system for each component—one each for the train, the controller, and the gate—since these components function largely on their own and communicate with each other only for specific tasks related to the crossing. The communication between the components can be modelled by having the components synchronise on common actions. For example the train and controller components must synchronise on the action "approach." In our method of synchronisation, for the system to execute a synchronising action, all components involved in that action must be ready to execute the action. Thus the train must wait till the controller is ready, before they can jointly perform the "approach" action. This can be interpreted as saying that when

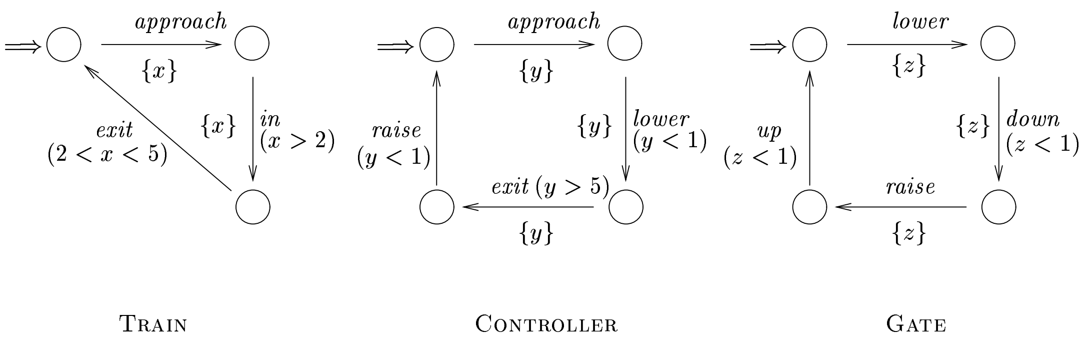

Figure 1.2: Timed automata model of the railroad crossing controller

the train sends the "approach" signal, it must wait for an acknowledgement from the controller before it can continue towards the crossing.

Suppose now that we abstract away from the timing information in the system, and simply model the fact that the components perform a sequence of causally dependent actions. It is not difficult to see that in this case the model we build will turn out to be unsafe. More importantly, the model permits several runs of the system that would otherwise be eliminated due to the timing constraints in the components.

The *timed automata* of Alur and Dill are transition system based models which are specially meant to capture the time-dependent nature of such systems. These are essentially finite state automata which are extended with a finite set of *clock variables*. The transitions of these automata are guarded by conditions on the clock variables, like "$x > 2$," and also specify a subset of clocks which are to be reset if this transition is taken. These automata generate sequences of action and time-stamp pairs called *timed words*. Operationally, we can generate timed words from a timed automaton in much the same way that we generate words from a classical automaton: we begin at an initial state and repeatedly take outgoing transitions from the states we visit, outputting the action label in the process. In the case of a timed automaton, we must additionally keep track of the current values of the clocks. We can let time elapse by staying in a state, with the clocks recording this elapsed time. When we choose to take a transition out of the state, the guard on the transition must be satisfied by the current values of the clock variables. The clock vector is then updated by resetting the set of clocks mentioned on the transition.

Figure 1.2 shows the railroad crossing system modelled as a network of timed automata. Each component is modelled as a timed automaton with a single clock variable. The transitions in each automaton are labelled by arrows of the form $\xrightarrow[X]{a,\,g}$ where a is the action label of the transition, X is the set of clocks to be reset along the transition, and g is the guard on the transition. As described earlier, the component automata must synchronise on common actions.

Let us examine the component for the train in some detail. Firstly, we should point out that we have used a single component to model *all* the trains in the system. This can be done without any loss of generality, since we have assumed that no more than one train can be in the system at a given time. The component has a single clock variable x. The train is initially in its start state (marked with an incoming double arrow in the figure). It can then take the transition labelled "approach" at any time, and reset its clock in the process. It must now let at least 2 minutes elapse before it can do the action "in." Once again x is reset along this transition. Finally, it can take the transition labelled "exit" anytime between 2 to 5 minutes after the "in" action. Of course, when it is working in tandem with the other components in the system, it must wait till the other components are ready to participate before it can perform synchronisation actions like "approach" and "exit."

The safety and utility questions about the system can now be phrased in terms of this model. Every consecutive pair of "in" and "exit" actions must be preceded by a "down" action, and there should be no other action of the gate in between. The utility property can be phrased as follows. At any instant, if an "exit" action occurs, and there is no "approach" action in the next 10 minutes, then the gate must perform a "raise" action within 2 minutes.

1.2 Analysing timed automata

The operational semantics we sketched earlier for a timed automaton $\mathcal{A}$ can be formalised in terms of an *infinite* state transition system $\mathcal{T}_{\mathcal{A}}$. Each state of this transition system is a pair comprising a state of the automaton, and a clock valuation which assigns to each clock a non-negative real value. The initial states of the transition system are the states of form (s_0, v_0) where s_0 is an initial state of the automaton, and v_0 is the zero valuation, in which all clocks have value 0. Suppose we are in a state (s, v) of this transition system at time t. Then we can take a transition labelled (a, t') and go to state (s', v') provided there exists a transition in the automaton of the form $s \xrightarrow[X]{a,\,g} s'$ with $v + (t' - t)$ satisfying the guard g, and v' being the resulting clock vector obtained from $v + (t' - t)$ by setting the clocks in X to zero. The valuation $v + (t' - t)$ is obtained from v by incrementing the value of each clock by $(t' - t)$.

How do we answer questions—like the ones posed about our model of the railroad crossing controller—when the object we need to reason about is *infinite*? Alur and Dill showed that this infinite transition system can be quotiented by defining an appropriate bisimulation on the states of the transition system. The quotiented transition system is finite and has the property that it recognises precisely the "untimed" language of the given timed automaton. By the untimed language we mean the classical word language obtained by projecting away the time stamps of the actions. Thus if $(a_1, t_1), (a_2, t_2), \ldots (a_n, t_n)$ were a timed word generated by the original transition system, then the sequence $a_1, a_2, \ldots, a_n$ is a word accepted by the quotiented automaton. Conversely, if $a_1, a_2, \ldots, a_n$ were a word accepted by the quotiented transition system, then there exist times $t_1, t_2, \ldots, t_n$ such that $(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)$ is a timed word in the original automaton.

It is pertinent to mention here the method of Berthomieu and Menasche which was one of the early methods of analysing timed systems. Their method, introduced in the context of *time Petri nets* [BM83], gives a quotienting of the infinite state semantic transition system by a simple subset construction. It turns out that each "subset" state can be described as a special form of convex polytope, and as a result the quotienting leads to a finite state transition system. This transition system (often referred to as the "zone" automaton), recognises the untimed language of the given timed automaton. Though in the worst case the number of states in the zone automaton is no less than those in the "region" automaton of Alur and Dill, the method performs well in practice and is popularly used in timed automata based verification tools like Kronos and Uppaal.

The quotiented automaton accepting the untimed language can be used to answer questions like "does the timed automaton accept any timed word at all?" since this is true if, and only if, the quotiented automaton itself accepts some word. Thus, properties like the safety property of our controller model can be verified directly using the quotiented automaton. However, to be able to check timing properties like the utility property, one must typically convert the specification itself into a timed automaton S, and then check whether $L(\mathcal{A}) \subseteq L(S)$. Since timed automata are closed under intersection, one can equivalently check if $L(\mathcal{A}) \cap L(S') = \emptyset$, provided that the complement S' of S exists.

1.3 Contributions of this thesis

One of the drawbacks of timed automata is that they are not closed under complementation. In fact, the language inclusion problem for this class is undecidable. As a result, if one wants to carry out model checking in this framework, one must either use deterministic specifications (which can be easily complemented) or one must work with a subclass of timed automata which has the required closure properties. From a theoretical standpoint the second route is more appealing, since it would mean identifying a class which parallels the classical untimed framework which, as we sketched earlier, has several advantages. From a practical point of view too, we would like to retain the feature of non-determinism which can be useful in a specification language.

In this thesis we adopt this route and propose suitable subclasses of timed automata and develop their theory. A common feature of these automata is that they comprise a *network* of timed automata which synchronise on common actions. The main restriction on the component automata is that each has a *single* clock, which must be reset with *every* action of that component. The model shown in Fig. 1.2 is in fact such an automaton. (Strictly speaking, the components don't reset their clocks along every transition—for example the train doesn't reset its clock along the "exit" action—but it is easy to see that this can be enforced, in this case, without affecting the behaviour of the model.)

The controller model is an instance of a *product interval automaton*, the weakest subclass we define. Here the synchronisations are determined completely by whether the participating components are willing to take part in the action. By allowing the synchronisations to be specified in a more "global" manner, we obtain the class of *distributed interval automata*, which are more expressive than product interval automata. Here we are allowed to examine the states of the participating components as well. Finally, we introduce *cellular interval automata* which are based on a generalization of distributed interval automata, where we have a separate component for each *action*.

Looked at in another way, these classes of automata are ways of structuring general multi-clock timed automata so as to give us a subclass with a clean theory. As we show, each of these subclasses is closed under the boolean operations of union, intersection and complementation. These automata can be determinized, and hence complemented, due to the disciplined way in which the clocks are reset. In addition, they admit a logical characterisation in terms of a full monadic second order logic interpreted over timed words. Typically, these logics have a timed modality of the form $\Delta_i(x, I)$ which asserts that at the position x in the timed word, the time since component i last performed an action lies in the interval I.

In the case of product interval automata, we investigate a natural temporal logic suggested by this logical characterisation, called timed product LTL, and denoted TLTL$^{\otimes}$. This logic comprises boolean combinations of assertions about the timed behaviour of each component. We use an action-indexed "next-state" modality $\langle a, I\rangle\alpha$ to specify that the next action of the component in question is labelled a and the time since it last performed an action lies in the interval I. Further, the formula α is true in the resulting state. We give automata-theoretic solutions for the satisfiability and model checking problems for this specification language. This involves constructing a product interval automaton for a given TLTL$^{\otimes}$ formula, which recognizes exactly the models of the given formula.

One the contributions of this thesis is the technique used for reasoning about timed automata in terms of classical untimed systems. We show that our automata can be represented symbolically using the notion of a "proper" symbolic alphabet. As a result we can reason about the timed languages of these automata using powerful techniques available in Mazurkiewicz trace theory and automata theory.

As an application of the theory developed here, we use product interval automata to model asynchronous circuits. These are circuits in which there is no common clock pulse which controls the switching of the gates in the circuit. Traditionally, these circuits are considered difficult to design and analyse. Maler and Pnueli [MP95] used a network of timed automata which communicate through shared memory to analyse the behaviour of these circuits. We show that product interval automata suffice to model this class of timed behaviours. In conjunction with TLTL$^{\otimes}$, we can specify and verify some useful properties of these circuits.

A final feature of these automata is that they are well-suited to the application of the recently proposed partial order reduction methods for timed systems [BJ+98, M99]. The method is explained briefly below.

Partial order based reduction methods [V92, KP92, G96] are an established way of tackling the so-called *state explosion* problem in the verification of distributed systems. The state explosion problem arises due to the fact that a distributed system may have a small description in terms of each of its components, but the actual global state space of the system could be exponentially large in the description of the components. This "explosion" of states can be prohibitively large for verification tools to handle. The partial order based reduction methods aim to tackle this problem by exploiting the structure in the global transition system. They make use of the fact that for certain "trace invariant" properties, it is sufficient to examine just one of the

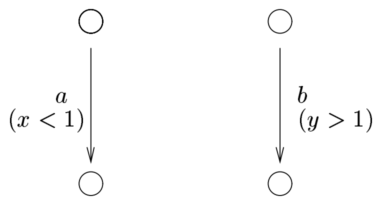

Figure 1.3: Causality introduced by timing constraints.

several interleaved runs corresponding to a single partially ordered computation of a distributed system. This often leads to a substantial reduction in the search space for the verification algorithms.

In the presence of timing constraints, however, this independence is destroyed since actions can be locationally independent but still forced into a causal order due to the timing constraints. In the Fig. 1.3 assume that the two components have just synchronised and reset their clocks, so that $x = y = 0$. Now even if a and b are independent, a is forced to occur before b due to the timing constraints.

Recently Bengtsson et al. [BJ+98] proposed a *local-time* semantics for networks of timed automata, where the local clocks of each automaton are allowed to progress independently of clocks in other components. Before each synchronising action, the participating components must ensure that the time elapsed locally for each of them are the same. This new semantics preserves the behaviour of the timed system provided we consider only such "synchronised" runs.

In this local time semantics, locationally independent actions are now also independent in the semantic transition system. However we are more interested in their independence in the quotiented transition system (the region or zone automaton) as it is this transition system which is searched during verification. In general, the locational independence may not imply independence in the quotiented transition system, and one has to examine the current region or zone to make sure the transitions don't disable each other. In [M99] Minea shows that for a certain class of systems called "sync-reset" systems in which each component has a single clock which is reset with every action that the component takes part in, locational independence does mean independence in the zone automaton as well.

Product interval automata are indeed instances of such sync-reset systems, and hence systems which are modelled using these automata are amenable to the efficient application of these methods.

However, these methods are still in a nascent stage and the actual gains have not been quantified. The hope is however that although more behaviour is added by the desynchronisation of clocks, the number of zones will turn out to be fewer. This, coupled with the reductions obtained by partial order based methods, is expected to lead to more efficient verification of timed systems.

1.4 Related Work

Subclasses of timed automata

As mentioned earlier, the property of our automata which make them determinizable is that they refer to implicit clocks only, which are reset in a uniform way. In fact it is not difficult to see that we can determinize a general timed automaton using just the classical techniques for determinizing and complementing classical automata, *provided* the resetting of clocks along transitions is determined solely by the input word.

Some important subclasses of timed automata have been proposed earlier based on this idea. In [AFH94] Alur, Fix and Henzinger proposed a subclass of timed automata called *event recording automata*. For each action a there is an implicit clock x_a which is reset with each occurrence of that action. Thus the clock x_a records the time elapsed since the last occurrence of the action a. The authors go on to show that a larger class of automata can be obtained by having another implicit clock for each action, called an event predicting clock, which keeps track of the time yet to elapse before the next occurrence of the action. This class, obtained by using both recording and predicting clocks, is called *event clock automata*.

These automata were further generalised in [HRS98] to allow the use of recording and predicting clocks for an arbitrary *automaton*. Roughly speaking, if an automaton $\mathcal{A}$ is thought of as running on a given timed word σ *starting* at time t, then the clock $x_{\mathcal{A}}$ records the last time t' at which $\mathcal{A}$ accepted σ starting at t'. This class of automata is called *recursive event clock automata*.

All three classes above were shown to be closed under boolean operations. In terms of modelling power, event recording automata—expressively the weakest of the three—are believed to be sufficiently powerful.

The classes product interval automata and distributed interval automata which we introduce are less expressive than event recording automata. However, our cellular interval automata have the same expressive power as event recording automata.

Another related piece of work is by Jonsson and Yi [YJ94] in the setting of timed CSP. They show that language inclusion problem is decidable for a class of automata similar to product interval automata, but where the delays are constants and not intervals. We show a variety of results about product interval automata which subsume this result.

Logical characterisations

The logical characterisations we give in this thesis are similar in flavour to Wilke's characterisation of timed automata via the logic $\mathcal{L}\overset{\leftrightarrow}{d}$ [W94]. $\mathcal{L}\overset{\leftrightarrow}{d}$, called the monadic second order logic of relative distance, is interpreted over timed words in much the same way as Büchi's MSO for classical words, and is shown to characterise the class of timed languages accepted by timed automata.

However, the fact that timed automata are not closed under complement is reflected in the fact that $\mathcal{L}\overset{\leftrightarrow}{d}$ is not a full monadic second order logic. It places syntactic restrictions on the formulas, particularly on the use of existential quantification over set variables.

In [HRS98] the authors give a logical characterisation of recursive event clock automata. Once again, their logic, called MinMaxML is not a full monadic second order logic, with the second order quantification being restricted. This is due to the fact that recursive event clock automata are not closed under geometric projection.

In contrast to these characterisations, our automata have full monadic second order logic characterisations. Further, we show that we can extend our methods to obtain a logical characterisation for event recording automata. A logical characterisation along the same lines for event clock automata is worked out in [DH99].

Timed temporal logics

Several temporal logics proposed for use as specification languages for timed systems turn out to be undecidable when interpreted over a dense time domain [AH93]. We discuss here two logics which are known to have decidable satisfiability problems—the Metric Interval Temporal Logic (MITL) of Alur, Feder and Henzinger, and the Event Clock Temporal Logic of Raskin and Schobbens. In fact it was shown recently in [HRS98] that both logics have the same expressive power, albeit differing complexities of decision procedures. In addition, MITL and EventClock TL are shown to be expressively equivalent to the first order fragment of MinMaxML, the monadic second order logic characterising recursive event clock automata.

MITL was introduced in [AFH96]. The logic uses an interval-indexed version of the "until" modality U_I which constrains formulas to be satisfied during the specified period. For example, the formula $\Box(p \Rightarrow \Diamond_{[1,2]}q)$ says that at all points t in the timed word, whenever p is true at t, there is a later point t' in the word where q is true, and $t' - t \in [1, 2]$. MITL restricts the interval I to be *non-singular*, i.e. I should not of the form $[c, c]$. In the absence of this restriction, the logic is shown to be undecidable.

Satisfiability of MITL formulas are shown to be EXPSPACE-complete, while the fragment of MITL which uses intervals which either begin at 0 or end at ∞—called $\mathrm{MITL}_{0,\infty}$—is equally expressive [HRS98] and is PSPACE-complete.

In [RS97] Raskin and Schobbens define a timed temporal logic based on event clock automata called State Clock Logic and later called Event Clock Temporal Logic in [HRS98]. Along with the usual "until" operator, they have "history" and "prophesy" operators. The history operator is of the form $\triangleleft_I \alpha$ and means that the formula α was last true t time units ago, for some $t \in I$. The prophesy operator $\triangleright_I$ has a similar interpretation.

The problem of deciding satisfiability of an Event Clock Logic formula is shown to be PSPACE-complete [HRS98]. In both MITL and Event Clock Logic, the satisfiability problem is solved along the lines of classical LTL, by defining a timed automaton (in the case of Event Clock Logic, an event clock automaton) which recognizes exactly the models of the given formula. In both cases the number of clocks used is proportional to the length of the given formula.

The temporal logic we study, $\mathrm{TLTL}^{\otimes}$, is less expressive than MITL. Our main motivation for examining $\mathrm{TLTL}^{\otimes}$ is that it is the natural temporal logic suggested by $\mathrm{TMSO}^{\otimes}$, our monadic second order logic characterising product interval automata. As we show, $\mathrm{TLTL}^{\otimes}$ is expressively equivalent to the first order fragment of $\mathrm{TMSO}^{\otimes}$.

The satisfiability of $\mathrm{TLTL}^{\otimes}$ is also PSPACE-complete. $\mathrm{TLTL}^{\otimes}$ has one advantage, however, in that the number of clocks in its formula automata is equal to the number of components in the system, whereas for both MITL and Event Clock Logic it is proportional to the size of the given formula.

Results from trace theory

This thesis draws on several well-known results from the theory of distributed systems. We summarise these results here.

Regular trace languages were studied by Mazurkiewicz as a way of describing the behaviour of distributed systems. Zielonka gave an automata theoretic characterisation of regular trace languages using *asynchronous au-*

tomata, and *asynchronous cellular automata* [Z87]. This characterisation was later extended to ω-regular trace languages, using asynchronous Büchi automata by Gastin and Petit [GP92].

Diekert and Muscholl [DM94] showed that asynchronous Büchi automata are equivalent to deterministic asynchronous Muller automata, and hence are closed under complementation. Muscholl [M94] later gave a direct construction for the complementation of asynchronous Büchi automata. However this did not involve determinizing these automata. Independently, Klarlund, Mukund, and Sohoni [KMS95] gave a direct determinization construction for Büchi asynchronous automata.

In terms of logical definability, Thomas [Tho89] gave a logical characterisation of regular trace languages in terms of a monadic second order logic interpreted over traces represented as labelled partial orders. This result was later extended to ω-regular trace languages by Ebinger and Muscholl [EM96].

The closure properties of regular product languages, a subclass of regular trace languages, are studied by Thiagarajan in [Thi95]. The logic product-LTL is proposed in this paper as a way of specifying trace invariant properties of distributed systems. The satisfiability and model checking problems for this logic are also solved.

1.5 Organisation of the thesis

The material in this thesis is organised as follows. We begin with *interval automata*, which are the component automata for product interval automata. After introducing the preliminary notions about timed words and timed automata, we define interval automata, and go on to show that they are closed under boolean operations. Finally we exhibit a simple logical characterisation of these timed languages.

The study of interval automata helps us while dealing with product interval automata in Chapter 3. We define these automata and the timed languages accepted by them and show that they are more expressive than interval automata. We show how these automata can be simulated by timed automata. This gives us a way of checking whether the language accepted by a given product interval automaton is empty or not. Next we show that these languages are closed under boolean operations. Finally, we present TMSO$^{\otimes}$, a monadic second order logic which characterises product interval languages.

In Chapter 4 we introduce the timed temporal logic TLTL$^{\otimes}$ and study its satisfiability and model checking problems. The chapter ends with a proof of the fact that TLTL$^{\otimes}$ is expressively equivalent to the first order fragment of TMSO$^{\otimes}$, the logic characterising product interval automata.

Chapter 5 is devoted to the modelling of asynchronous circuits using product interval automata. We describe the behaviour of a circuit as a timed language over an appropriate alphabet, and show that we can define a product interval automaton which accepts this language. We also give some properties which we can verify using this model in conjunction with the logic $\mathrm{TLTL}^{\otimes}$.

We introduce distributed interval automata in Chapter 6. We show that the class of languages accepted by these automata are closed under boolean operations. This chapter contains two detailed sections on a logical characterisation of these languages. We first show that distributed interval languages can be represented as *timed trace* languages. We then give a logical characterisation of these timed trace languages.

In Chapter 7 we study cellular interval automata, the cellular version of distributed interval automata. To reason about the languages accepted by these automata, we introduce what we call *quasi* cellular asynchronous automata and show that they have the same expressive power as asynchronous automata. This helps us to show that cellular interval languages are closed under boolean operations, and also that they admit a logical characterisation. Finally we show how these languages are related to event recording automata, and how the techniques introduced this far help us to give a logical characterisation for event recording automata.

We conclude with Chapter 8, where we summarise our results and detail some avenues for further research.

Chapter 2

Interval Automata

In this chapter we introduce a restricted class of timed automata called interval automata. These automata have a single clock which is reset along every transition. Effectively, these automata can only measure the time elapsed since the last action performed. As a result, these automata are fairly weak in terms of expressiveness.

However, the class of languages accepted by these automata enjoys some nice closure properties. It is closed under boolean operations and, in addition, admits a simple logical characterisation in terms of a monadic second order logic interpreted over timed words. The techniques introduced here are representative of many of the techniques used subsequently in the thesis. The results too will play an important role, particularly in the next chapter where we deal with product interval automata which are essentially networks of interval automata.

In this chapter we first introduce some preliminary notions about timed words and timed automata. We sketch the construction of the region automata for these automata. We then introduce interval automata and answer some natural questions about them. In particular, we show that the class of languages accepted by these automata are closed under boolean operations. Finally, in Section 2.5 we provide a logical characterisation of these languages.

2.1 Preliminary notions

As usual, for an alphabet A we will use A^* and A^ω to denote the set of finite and infinite words over A respectively. We will use A^∞ to denote the set $A^* \cup A^\omega$.

It will be necessary for us to deal with both finite and infinite words, and in this regard it is convenient to use prefixes to play the role of positions in the word. For a word σ in A^∞, $\mathit{prf}(\sigma)$ will be used to denote the set of finite prefixes of σ. The strict and non-strict prefix relations on finite words will be denoted by $\prec$ and $\preceq$ respectively. We will use $|\sigma|$ to denote the length of a word σ. The empty word will be denoted by ϵ.

Büchi automata

It will be helpful to recall the definition of Büchi automata. The reader is referred to [Tho90] for a comprehensive treatment of this subject.

Definition 2.1 *Let A be a finite alphabet. A (mixed) Büchi automaton over the alphabet A is a structure $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, G)$ where*

- *Q is a finite set of states,*
- *$\longrightarrow \subseteq Q \times A \times Q$ is the transition relation,*
- *$Q_{in} \subseteq Q$ is a set of initial states, and*
- *$F, G \subseteq Q$ are, respectively, the finitary and infinitary acceptance state sets.*

Let $\sigma \in A^\infty$. A run of $\mathcal{A}$ over σ is a map $\rho : \mathit{prf}(\sigma) \to Q$ which satisfies:

- $\rho(\epsilon) \in Q_{in}$.
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for every $\tau a \in \mathit{prf}(\sigma)$. ($q \xrightarrow{a} q'$ is alternate notation for $(q, a, q') \in \longrightarrow$).

We say ρ is an *accepting* run on σ iff either

- σ is finite and $\rho(\sigma) \in F$, or,
- σ is infinite and $\rho(\tau) \in G$ for infinitely many $\tau \in \mathit{prf}(\sigma)$.

The set of words accepted by $\mathcal{A}$, denoted $L_{sym}(\mathcal{A})$, is defined to be the set of words in A^∞ on which $\mathcal{A}$ has an accepting run. Following established convention, we term a subset L of A^∞ *regular* if $L = L_{sym}(\mathcal{A})$ for some Büchi automaton $\mathcal{A}$ over A.

It will be convenient for us to sometimes use a *generalised* Büchi automaton. This is a Büchi automaton in which the infinitary acceptance set is replaced by a family of accepting sets $\mathcal{G} = \{G_1, \ldots, G_m\}$ with $m \geq 0$. Let $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, \mathcal{G})$ be a generalised Büchi automaton, with $\mathcal{G} = \{G_1, \ldots, G_m\}$ ($m \geq 0$). Then a run ρ on a word $\sigma \in A^\infty$ is defined as done for a Büchi automaton. We say that ρ is accepting iff either

- σ is finite and $\rho(\sigma) \in F$, or,
- σ is infinite and for each $i \in \{1, \ldots, m\}$, $\rho(\tau) \in G_i$ for infinitely many $\tau \in \mathit{prf}(\sigma)$.

A generalised Büchi automaton can be converted to a standard Büchi automaton as follows. Let $\mathcal{A}$ be the generalised Büchi automaton above. Define $\mathcal{A}' = (Q', \longrightarrow', Q'_{in}, F', G')$ where

- $Q' = Q \times \{0, 1, \ldots, m\}$
- $\longrightarrow'$ is given by $(q, l) \xrightarrow{a}' (q', l')$ iff
  - $q \xrightarrow{a} q'$
  - $l' = (l + 1) \bmod (m + 1)$ if $q' \in G_l$ or $l = 0$. Otherwise $l' = l$.
- $Q'_{in} = Q_{in} \times \{0\}$.
- $F' = F \times \{0, \ldots, m\}$
- $G' = Q \times \{0\}$.

It is not difficult to verify that $L_{sym}(\mathcal{A}) = L_{sym}(\mathcal{A}')$. We observe that the simulation involves a blow-up of $O(m)$ in the number of states of $\mathcal{A}$.

The problem of checking whether the language accepted by a Büchi automaton is empty can be efficiently solved. This can be done by checking that the language of finite words is empty and that the language of infinite words is empty. The language of finite words is empty iff there is no finitary final state reachable from a start state. The language of infinite words is empty iff there is no cycle (in the underlying graph of the automaton) containing an infinitary final state and reachable from a start state. Both these checks can be carried out by well-known linear time algorithms.

Timed automata

The notion of a *timed word* is central to this thesis. In what follows we will use $\mathbb{R}^{>0}$ and $\mathbb{R}^{\geq 0}$ to denote the set of positive and non-negative reals respectively. The non-negative rationals will be denoted by $\mathbb{Q}^{\geq 0}$.

Definition 2.2 *Let Σ be a finite alphabet of actions. A* timed word *over Σ is a member σ of $(\Sigma \times \mathbb{R}^{>0})^{\infty}$ such that*

1. *for all prefixes $\tau(a, t)(b, t')$ of σ we have $t < t'$ (strict monotonicity).*

2. *if σ is infinite, then for each $t \in \mathbb{R}^{>0}$ there exists a prefix $\tau(a, t')$ of σ such that $t' > t$ (progressiveness).*

We use $T\Sigma^*$ and $T\Sigma^\omega$ to denote the set of finite and infinite timed words over Σ, respectively, and set $T\Sigma^\infty = T\Sigma^* \cup T\Sigma^\omega$.

For a finite timed word τ we will use $time(\tau)$ to denote the time of occurrence of the last action in τ. Formally, $time(\epsilon) = 0$, and $time(\tau(a, t)) = t$. Analogously, for a non-empty finite timed word τ we will use $action(\tau)$ to denote the last action in τ.

Wherever convenient, we will also use the representation of a timed word σ as a pair (α, η) where $\alpha \in \Sigma^\infty$ and $\eta : prf(\alpha) \to \mathbb{R}^{>0}$. For an infinite timed word σ we will also sometimes use the representation of σ as (α, η) where $\alpha \in \Sigma^\omega$ and $\eta : \mathbb{N} - \{0\} \to \mathbb{R}^{>0}$. Here $\mathbb{N}$ is the set of natural numbers $\{0, 1, \ldots\}$.

In what follows, we will use intervals with rational bounds to specify timing constraints (and use ∞ as the upper bound to capture unbounded intervals). These intervals will be of the form (l, r), $[l, r)$, $(l, r]$, or $[l, r]$, where $l, r \in \mathbb{Q}^{\geq 0} \cup \{\infty\}$ with $l \leq r$. For an interval of the form $(l, r]$ or $[l, r]$ we require $r \neq \infty$. Further, to avoid empty intervals, unless an interval is of the form $[l, r]$, we require $l < r$. An interval will denote a non-empty, convex subset of reals in the obvious way. For example the interval $[1, \infty)$ denotes the set $\{t \in \mathbb{R} \mid 1 \leq t\}$. The set of all intervals will be denoted by $\mathcal{I}\mathbb{R}$.

Definition 2.3 *A (mixed)* timed Büchi automaton*, TBA for short, over an alphabet Σ is a structure $\mathcal{B} = (Q, \longrightarrow, C, Q_{in}, F, G)$ where:*

- *Q is a finite set of states,*
- *$Q_{in} \subseteq Q$ is a set of initial states,*
- *$F, G \subseteq Q$ are sets of finitary and infinitary accepting states,*
- *C is a finite set of clocks, and*
- *$\longrightarrow$, the transitions of $\mathcal{B}$, is a finite subset of $Q \times \Sigma \times 2^C \times G_C \times Q$ where G_C is the set of clock constraints (guards) which are conjunctions of atomic guards of the form $(x \in I)$, where x ranges over C and I ranges over $\mathcal{I}\mathbb{R}$.*

Actually this is a mild reformulation of the one given by Alur and Dill in [AD94] and it involves no loss of generality. In what follows a transition (q, a, X, g, q') will be written as $q \xrightarrow[X]{a, g} q'$. The manner in which the timed automaton accepts a timed word is defined in terms of clock valuations.

A C-valuation is a map $v : C \to \mathbb{R}^{\geq 0}$. Where C is clear from the context we will say valuation instead of C-valuation. We let val_C stand for the set of C-valuations. Let v be a valuation and $t \in \mathbb{R}^{\geq 0}$. Then $v + t$ is the valuation given by:

$$(v + t)(x) = v(x) + t \text{ for every } x \in C.$$

Suppose v is a valuation, $t \in \mathbb{R}^{\geq 0}$ and $X \subseteq C$. Then the valuation $v[t/X]$ is given by:

$$v[t/X](y) = \begin{cases} t & \text{if } y \in X \\ v(y) & \text{otherwise.} \end{cases}$$

Finally, $\mathbf{0}$ is the null-valuation given by:

$$\mathbf{0}(x) = 0 \text{ for every } x \in C.$$

Next, the notion of a valuation v satisfying a clock constraint g is denoted by $v \models g$ and is defined via:

- $v \models x \in I$ iff $v(x) \in I$
- $v \models \varphi \wedge \varphi'$ iff $v \models \varphi$ and $v \models \varphi'$.

Let $\sigma \in T\Sigma^{\infty}$. Then a run of $\mathcal{B}$ over σ is a pair of maps (ρ, ν) where $\rho : \mathit{prf}(\sigma) \to Q$ and $\nu : \mathit{prf}(\sigma) \to val_C$ are such that the following conditions are satisfied:

- $\rho(\epsilon) \in Q_{in}$ and $\nu(\epsilon) = \mathbf{0}$
- For every $\tau(a, t) \in \mathit{prf}(\sigma)$, there exists a transition $\rho(\tau) \xrightarrow[X]{a,g} \rho(\tau(a, t))$ such that $\nu(\tau) + t \models g$ and $\nu(\tau(a, t)) = (\nu(\tau) + t)[0/X]$.

The run (ρ, ν) is an accepting run iff either

1. σ is finite and $\rho(\sigma) \in F$, or,
2. σ is infinite and $\rho(\tau) \in G$ for infinitely many prefixes τ of σ.

$L(\mathcal{B}) \subseteq T\Sigma^{\infty}$ the language of timed words accepted by $\mathcal{B}$ is then given by:

$$L(\mathcal{B}) = \{\sigma \mid \exists \text{ an accepting run of } \mathcal{B} \text{ over } \sigma\}.$$

For a timed word $\sigma \in T\Sigma^{\infty}$ let $\mathit{untime}(\sigma)$ be the word $\widehat{\sigma} \in \Sigma^{\infty}$ obtained by projecting away the time-stamps from σ. For a timed language $L \subseteq T\Sigma^{\infty}$, $\mathit{untime}(L) \subseteq \Sigma^{\infty}$ will denote the set $\{\mathit{untime}(\sigma) \mid \sigma \in L\}$.

Alur and Dill showed the following result:

Theorem 2.4 ([AD94]) *Given a timed automaton $\mathcal{B}$ over an alphabet Σ one can effectively construct a Büchi automaton $\mathcal{B}'$ over Σ such that $L(\mathcal{B}') = untime(L(\mathcal{B}))$.*

The first step is to "normalise" the guards in $\mathcal{B}$ so that all intervals are integer bounded. This can be done by multiplying all the rational bounds by the least common multiple of all the denominators. This operation does not affect the untimed language of $\mathcal{B}$.

The central idea in this result is the construction of the "region" automaton $R(\mathcal{B})$. Let $\mathcal{B} = (Q, \longrightarrow, C, Q_{in}, F, G)$. For each $x \in C$ let c_x denote the largest interval bound appearing in a guard of the form $(x \in I)$ in $\mathcal{B}$. Then a clock region (w.r.t. $\mathcal{B}$) is a convex subset of val_C given by inequalities of the form

1. For each clock $x \in C$ a constraint of the form $c < x < c+1$, $x = c$, or $x > c_x$ where c is an integer such that $0 \leq c < c_x$.

2. For each pair of clocks $x, y \in C$, a constraint of the form $x - y < c$ or $y - x < c$ or $x - y = c$.

The number of clock regions is finite and can be seen to be bounded by

$$2^{O(|C|)} \cdot |C|! \cdot \prod_{x \in C} (c_x + 1)$$

The automaton $R(\mathcal{B})$ has as its states pairs of the form (q, r) where $q \in Q$ and r is a clock region. The initial states are of the form (q_0, r_0) where $q_0 \in Q_{in}$ and r_0 is the unique region containing $\mathbf{0}$. The transition relation of $R(\mathcal{B})$ is given by: $(q, r) \overset{a}{\longrightarrow}' (q', r')$ iff there exists an edge $q \xrightarrow[X]{b,g} q'$ in $\mathcal{B}$ such that $b = a$, there exists $v \in r$, $v' \in r'$ and $t \in \mathbb{R}^{>0}$ such that $v + t \models g$, and $v' = (v + t)[0/X]$. A state of the form (q, r) is a finitary (respectively infinitary) final state of $R(\mathcal{B})$ iff $q \in F$ (respectively $q \in G$) and $r \in val_C$.

Regions have the property that for a given region, the truth of any guard of $\mathcal{B}$ is invariant across the valuations in it. Further, if r' is a time successor of r—by this we mean there exists $v \in r$ and a $t \in \mathbb{R}^{>0}$ such that $v + t \in r'$—then for every $u \in r$ there exists $t \in \mathbb{R}^{>0}$ such that $u + t \in r'$.

Using these properties of regions it is possible to show that $R(\mathcal{B})$ accepts the untimed language of $\mathcal{B}$ provided $\mathcal{B}$ has the property that for every run ρ of $\mathcal{B}$ on a non-progressive timed word, ρ is also a run on a progressive timed word.

However, this may not be true of $\mathcal{B}$ in general, and one must add extra information in the states of $R(\mathcal{B})$ and define an appropriate Büchi condition to make sure that the run is progressive. The resulting automaton is the automaton $\mathcal{B}'$ promised in the statement of the theorem.

The size of $\mathcal{B}'$ can be seen to be

$$O(|Q| + |E|) \cdot 2^{O(|C|)} \cdot |C|! \cdot \prod_{x \in C} (c_x + 1)$$

where E is the set of transitions of $\mathcal{B}$. Further, this is also an upper bound on the time needed to generate $\mathcal{B}'$.

We can now check if timed language accepted by $\mathcal{B}$ is empty by generating $\mathcal{B}'$ and checking whether the language it accepts is empty. Since $\mathcal{B}'$ is a classical Büchi automaton, and its emptiness can be checked in time linear in its size. Thus, we have:

Corollary 2.5 *The emptiness of the timed language accepted by a TBA $\mathcal{B}$ can be checked in time*

$$O(|Q| + |E|) \cdot 2^{O(|C|)} \cdot |C|! \cdot \prod_{x \in C} (c_x + 1),$$

where Q and E are the state and edge sets of $\mathcal{B}$ respectively, C is the set of clocks used in $\mathcal{B}$, and for each $x \in C$, c_x is the largest normalised interval bound appearing in $\mathcal{B}$ in a guard of the form $(x \in I)$.

Using standard convention we let $|\mathcal{B}|$ denote the size of the representation of $\mathcal{B}$, using binary encoding for the numeric constants. Then the above expression can be seen to bounded by $2^{(|\mathcal{B}|^2)}$.

2.2 Interval automata and their languages

The notion of an *interval alphabet* will be useful in representing timed languages symbolically. Let Σ be a finite alphabet. Then an *interval alphabet* based on Σ is a finite subset of $\Sigma \times \mathcal{IR}$.

Given an interval alphabet Γ over Σ and a word $\widehat{\sigma} \in \Gamma^\infty$, $\widehat{\sigma}$ naturally induces a set of timed words over Σ which we will denote $tw(\widehat{\sigma})$. It is defined as follows. Let $\sigma \in T\Sigma^\infty$. Then $\sigma \in tw(\widehat{\sigma})$ iff

1. $|\sigma| = |\widehat{\sigma}|$.

2. For each prefix $\tau(a, t)$ of σ, and for each prefix $\widehat{\tau}(b, I)$ of $\widehat{\sigma}$ such that $|\tau| = |\widehat{\tau}|$, we have $a = b$ and $t - time(\tau) \in I$.

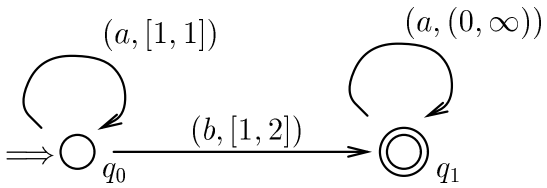

Figure 2.1: An interval automaton over $\{a, b\}$.

For $L \subseteq \Gamma^{\infty}$ we set

$$tw(L) = \bigcup_{\widehat{\sigma} \in L} tw(\widehat{\sigma}).$$

Definition 2.6 *An* interval automaton *over* Σ *is simply a Büchi automaton over an interval alphabet based on* Σ.

Thus an interval automaton $\mathcal{A}$ over Σ has edges of the form $q \xrightarrow{(a,I)} q'$, where a is a Σ-action and I is an interval. Viewed as a Büchi automaton over an interval alphabet Γ, $\mathcal{A}$ accepts the symbolic language $L_{sym}(\mathcal{A}) \subseteq \Gamma^{\infty}$. What is more interesting to us however, is the timed language accepted by $\mathcal{A}$, denoted $L(\mathcal{A})$, which we define as

$$L(\mathcal{A}) = tw(L_{sym}(\mathcal{A})).$$

We will say a timed language $L \subseteq T\Sigma^{\infty}$ is a *regular interval language* if $L = L(\mathcal{A})$ for some interval automaton $\mathcal{A}$ over Σ.

Example 2.7 Figure 2.1 shows an interval automaton $\mathcal{A}$ over the alphabet $\Sigma = \{a, b\}$. Here we have $\mathcal{A} = (\{q_0, q_1\}, \longrightarrow, \{q_0\}, \{q_1\}, \{q_1\})$. In the figure we use the convention that the initial states are indicated with incoming double arrows, while final states are indicated by two concentric circles. The automaton accepts all timed words $\sigma \in T\Sigma^{\infty}$ which begin with a (possibly empty) sequence of a actions, unit times apart, followed by a b action at some time between 1 and 2 units from the last action, and finally a sequence of a's again, without any time restrictions. $\square$

We can also define the notion of a run of an interval automaton directly over a timed word. Let $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, G)$ be an interval automaton over Σ, and let $\sigma \in T\Sigma^{\infty}$. Then a run of $\mathcal{A}$ on σ is map $\rho : prf(\sigma) \rightarrow Q$ such that

1. $\rho(\epsilon) \in Q_{in}$

2. For each $\tau(a,t) \in \mathit{prf}(\sigma)$, there exists I, such that $\rho(\tau) \xrightarrow{(a,I)} \rho(\tau(a,t))$ and $(t - \mathit{time}(\tau)) \in I$.

As usual, the run ρ will be termed accepting if either σ is finite and $\rho(\sigma) \in F$, or, σ is infinite and $\rho(\tau) \in G$ for infinitely many $\tau \in \mathit{prf}(\sigma)$.

It is not difficult to see that this is an equivalent way of defining the timed language of $\mathcal{A}$—in the sense that $\sigma \in \mathit{tw}(L_{sym}(\mathcal{A}))$ iff there exists an accepting run of $\mathcal{A}$ on σ, in the sense defined above.

2.3 Checking emptiness

The problem of checking whether the timed language accepted by an interval automaton $\mathcal{A}$ is empty or not, can be easily solved. We can do this by simply checking whether $L_{sym}(\mathcal{A})$ is empty or not. This follows from the following proposition:

Proposition 2.8 *Let Γ be an interval alphabet based on Σ such that Γ does not mention the interval $[0,0]$. Let $\widehat{\sigma} \in \Gamma^\infty$. Then $\mathit{tw}(\widehat{\sigma}) \neq \emptyset$.*

Proof We use the fact that each $I \in \mathcal{IR}$ is a non-empty subset of $\mathbb{R}^{>0}$ provided $I \neq [0,0]$. Let $\mathit{voc}(\Gamma)$ denote the set of intervals mentioned in Γ. Then, with each $I \in \mathit{voc}(\Gamma)$ we can associate a $t_I \in \mathbb{R}^{>0}$ such that $t_I \in I$. Now we can define a timed word $\sigma \in T\Sigma^\infty$ such that $\sigma \in \mathit{tw}(\widehat{\sigma})$. We define σ by defining its set of prefixes as follows. With every prefix of $\widehat{\sigma}$ we inductively associate a prefix of σ. With the empty prefix we associate the empty word itself. Let $\widehat{\tau}(a,I)$ be a prefix of $\widehat{\sigma}$. Then with $\widehat{\tau}(a,I)$ we associate the word $\tau(a,t)$ where τ is the prefix we have associated with $\widehat{\tau}$, and $t = \mathit{time}(\tau) + t_I$. We need to check that σ is progressive. So suppose σ is infinite and let $t \in \mathbb{R}^{>0}$. Let t_0 be the minimum of the set $\{t_I \mid I \in \mathit{voc}(\Gamma)\}$. (Note that the set is finite and hence we can speak of the minimum.) Let τ be the prefix of σ such that $|\tau| > t/t_0$. (Such a τ exists by the Archimedian property of $\mathbb{R}$.) Then it is clear that $\mathit{time}(\tau) > t$. Thus $\sigma \in T\Sigma^\infty$ and, by construction, $\sigma \in \mathit{tw}(\widehat{\sigma})$. $\square$

Now, a symbolic word $\widehat{\sigma}$ which contains a letter of the form $(a,[0,0])$ cannot give rise to a timed word and we have $\mathit{tw}(\widehat{\sigma}) = \emptyset$. Thus, if $\mathcal{A}$ uses the interval $[0,0]$, we can simply drop all the edges in $\mathcal{A}$ labelled by this interval, without affecting the timed language accepted by $\mathcal{A}$. Thus, we can assume without loss of generality that $\mathcal{A}$ is over an interval alphabet Γ which does not make use of the interval $[0,0]$. Checking emptiness of $L(\mathcal{A})$ is then reduced to checking emptiness of $L_{sym}(\mathcal{A})$. This can be done in linear time in the size of $\mathcal{A}$.

Before we close this section we show that interval automata are a subclass of Alur-Dill timed automata [AD94]. In fact they correspond to the class of timed automata which have a single clock which is reset along *every* edge in the automaton.

Proposition 2.9 *Let $L \subseteq T\Sigma^\infty$. Then the following statements are equivalent.*

1. *There exists an interval automaton $\mathcal{A}$ over Σ such that $L = L(\mathcal{A})$.*
2. *There exists a TBA $\mathcal{B}$ over Σ with a single clock which is reset along every edge, such that $L = L(\mathcal{B})$.*

Proof $1 \Rightarrow 2$: Let $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, G)$ be an interval automaton over the interval alphabet Γ based on Σ. Consider the TBA $\mathcal{B} = (Q, \longrightarrow', \{x\}, Q_{in}, F, G)$ where $\longrightarrow'$ is given by:

$$q \underset{\{x\}}{\overset{a,(x\in I)}{\longrightarrow'}} q' \text{ iff } q \overset{(a,I)}{\longrightarrow} q'.$$

It is routine to verify that $L(\mathcal{B}) = L(\mathcal{A})$.

$2 \Rightarrow 1$: Let $\mathcal{B} = (Q, \longrightarrow', C, Q_{in}, F, G)$ be a TBA with $C = \{x\}$ and with $\longrightarrow'$ of the form $q \underset{\{x\}}{\overset{a,g}{\longrightarrow'}} q'$ for some $a \in \Sigma$ and $g \in G_C$. Since a conjunction of atomic guards over a single clock can always be written as a single atomic guard, we can further assume that each clock constraint g is of the form $x \in I$. Let $\mathcal{A}$ be the interval automaton defined as $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, G)$ where $\longrightarrow$ is given by

$$q \overset{(a,I)}{\longrightarrow} q' \text{ iff } q \underset{\{x\}}{\overset{a,(x\in I)}{\longrightarrow'}} q'.$$

It is again easy to check that $L(\mathcal{A}) = L(\mathcal{B})$. □

We give here an example of a language which cannot be accepted by an interval automaton.

Lemma 2.10 Let Σ be the alphabet $\{a, b\}$. Then the timed language L over Σ given by

$$L = \{(b,t)(a,t')(b,t'') \in T\Sigma^\infty \mid t'' - t > 1\}$$

is not a regular interval language over Σ.

Proof To see this, assume that there exists an interval alphabet Γ based on Σ and $\widehat{L} \subseteq \Gamma^\infty$ such that $L = tw(\widehat{L})$. Let n be the least common multiple of the denominators of the rational bounds mentioned in Γ. Then for every interval bound $r \neq \infty$ mentioned in Γ, we have $r = \frac{k}{n}$ for some non-negative integer k. Now consider the timed word σ given by:

$$\sigma = (b,1)(a,1+\frac{1}{2n})(b,2+\frac{1}{4n}).$$

Clearly $\sigma \in L$. Hence $L = tw(\widehat{L})$ would imply that there exists $\widehat{\sigma} = (b,I)(a,I')(b,I'')$ in $\widehat{L}$ such that $\sigma \in tw(\widehat{\sigma})$. We now see that $\frac{1}{2n} \in I'$. Further, $1-\frac{1}{4n} \in I''$ because $(2+\frac{1}{4n})-(1+\frac{1}{2n}) = 1-\frac{1}{4n}$.

Let $I' = \langle l, r\rangle$. Since $\frac{1}{2n} \in \langle l, r\rangle$, we know that $l \leq \frac{1}{2n}$ and hence it must be the case that $l = 0$. This lets us conclude that $\frac{1}{4n} \in I'$. Now consider the timed word σ' given by:

$$\sigma' = (b,1)(a,1+\frac{1}{4n})(b,2).$$

We claim that $\sigma' \in tw(\widehat{\sigma})$. To see this we just have to verify that $2-(1+\frac{1}{4n}) = 1-\frac{1}{4n} \in I''$ and we already know that $\frac{1}{4n} \in I'$. But then $\sigma' \notin L$ and this is a contradiction. $\square$

2.4 Closure under boolean operations

We now show that the class of regular interval languages is closed under boolean operations.

Closure under union is straightforward. Let L and M be two regular interval languages over Σ. Then we know there exist two interval alphabets Γ_1 and Γ_2 based on Σ, and regular subsets $\widehat{L}$ of Γ_1^∞, and $\widehat{M}$ of Γ_2^∞, such that $L = tw(\widehat{L})$ and $M = tw(\widehat{M})$. Now consider the alphabet $\Gamma = \Gamma_1 \cup \Gamma_2$ and the subset $\widehat{L} \cup \widehat{M}$ of Γ^∞. By classical automata theory, we know that $\widehat{L} \cup \widehat{M}$ is a regular subset of Γ^∞. Further, by definition of the tw map, it follows that $tw(\widehat{L} \cup \widehat{M}) = tw(\widehat{L}) \cup tw(\widehat{M})$. Thus $L \cup M = tw(\widehat{L}) \cup tw(\widehat{M})$ is a regular interval language.

For closure under complementation the notion of a *proper* interval set will prove useful. This notion will also play an important role in subsequent sections.

We say a set of intervals $\mathcal{I} \subseteq \mathcal{I}\mathbb{R}$ is *proper* if it forms a finite partition of $\mathbb{R}^{\geq 0}$. Thus, if $\mathcal{I}$ is a proper interval set, then for each $t \in \mathbb{R}^{\geq 0}$ there exists an $I \in \mathcal{I}$ such that $t \in I$, and for each $I, I' \in \mathcal{I}$, $I \cap I' \neq \emptyset$ implies $I = I'$. An interval alphabet Γ will be termed proper if for each $a \in \Sigma$ the set $\Gamma_a = \{I \mid (a, I) \in \Gamma\}$ is a proper interval set. We say an interval set $\mathcal{I}$ *covers* an interval set $\mathcal{I}'$ if every interval in $\mathcal{I}'$ is the union of some subset of intervals in $\mathcal{I}$. Finally, an interval alphabet Γ covers the interval alphabet Γ' if Γ_a covers Γ'_a for each $a \in \Sigma$.

Each interval alphabet Γ induces, in a canonical way, a proper interval alphabet, denoted $prop(\Gamma)$, with the property that it covers Γ. It is given by

$$prop(\Gamma) = \{(a, I) \mid a \in \Sigma,\ I \in prop(\Gamma_a)\}$$

where for each a, the set $prop(\Gamma_a)$ is obtained from Γ_a by the procedure outlined below.

Let $\mathcal{I}$ be a non-empty finite set of intervals (if it is empty, we simply set $prop(\mathcal{I}) = \{[0, \infty)\}$). Let $V = \{0, v_1, v_2, \ldots, v_n, \infty\}$ where for $1 \leq i \leq n$, $v_i \in V$ iff there exists $I \in \mathcal{I}$ with v_i as the left or right end of I. Without loss of generality, we assume that $n \geq 1$ and $0 < v_1 < v_2 \cdots < v_n \neq \infty$. Now define $prop(\mathcal{I})$ via:

$$prop(\mathcal{I}) = \{[v_j, v_j],\ (v_j, v_{j+1}) \mid 0 \leq j \leq n\}$$

where we set $v_0 = 0$ and $v_{n+1} = \infty$. It is easy to verify that $prop(\mathcal{I})$ is a proper interval set which covers $\mathcal{I}$.

The following is an important property of proper interval alphabets.

Lemma 2.11 *Let Γ be a proper interval alphabet based on Σ. Then for each $\sigma \in T\Sigma^\infty$ there exists a* unique *word $\widehat{\sigma} \in \Gamma^\infty$ such that $\sigma \in tw(\widehat{\sigma})$.*

Proof Let $\sigma \in T\Sigma^\infty$. Since Γ is proper we know that for each $a \in \Sigma$ and $t \in \mathbb{R}^{>0}$ there exists a unique $I \in \Gamma_a$ such that $t \in I$. Consider the word $\widehat{\sigma} \in \Gamma^\infty$ given by its set of prefixes which we define as follows. For each prefix τ of σ we define a corresponding prefix $\widehat{\tau}$ of $\widehat{\sigma}$. The prefix corresponding to ϵ is ϵ itself. The prefix corresponding to the non-empty prefix $\tau(a, t)$ of σ is $\widehat{\tau}(a, I)$ where I is the unique interval in Γ_a such that $(t - time(\tau)) \in I$. It is easy to verify that $\sigma \in tw(\widehat{\sigma})$.

For uniqueness of $\widehat{\sigma}$ suppose $\sigma \in tw(\widehat{\sigma}_1)$ and $\sigma \in tw(\widehat{\sigma}_2)$ for some $\widehat{\sigma}_1, \widehat{\sigma}_2 \in \Gamma^\infty$. If $\widehat{\sigma}_1 \neq \widehat{\sigma}_2$ then there must exist prefixes $\widehat{\tau}_1(a, I)$ of $\widehat{\sigma}_1$ and $\widehat{\tau}_2(a, I')$ of $\widehat{\sigma}_2$, such that $|\widehat{\tau}_1| = |\widehat{\tau}_2|$ and $I \neq I'$. Let $\tau(a, t)$ be a prefix of σ such that $|\tau| = |\widehat{\tau}_1|$. Then we know $(t - time(\tau))$ belongs to both I and I'. Since Γ_a is proper, this would mean $I = I'$ which contradicts our assumption. Thus we must have $\widehat{\sigma}_1 = \widehat{\sigma}_2$. $\square$

We next observe that proper interval alphabets suffice to capture regular interval languages.

Lemma 2.12 *Let Γ be an interval alphabet based on Σ and let $\widehat{L}$ be a regular subset of Γ^∞. Let Γ' be a proper interval alphabet which covers Γ (note that $prop(\Gamma)$ is at least one such alphabet). Then there exists a regular subset $\widehat{L}'$ of $(\Gamma')^\infty$ such that $tw(\widehat{L}') = tw(\widehat{L})$.*

Proof Since $\widehat{L}$ is a regular subset of Γ^∞ there exists a Büchi automaton $\mathcal{A}$ over Γ such that $L_{sym}(\mathcal{A}) = \widehat{L}$. Let $\mathcal{A} = (Q, \longrightarrow, Q_{in}, F, G)$. Consider the automaton $\mathcal{B} = (Q, \longrightarrow', Q_{in}, F, G)$ over Γ' where $\longrightarrow'$ is given by

$$q \overset{(a,I')}{\longrightarrow'} q' \text{ iff } \exists q \overset{(a,I)}{\longrightarrow} q' \text{ with } I' \subseteq I.$$

Let $\widehat{L}' = L_{sym}(\mathcal{B})$. Then it is routine to argue that $tw(\widehat{L}') = tw(\widehat{L})$. □

Using properties of proper alphabets we can now show closure under complementation. Let L be an regular interval language over Σ. Without loss of generality (using Lemma 2.12), we can assume that there exists a proper interval alphabet Γ based on Σ and a regular subset $\widehat{L}$ of Γ^∞ such that $L = tw(\widehat{L})$. We now claim that $T\Sigma^\infty - L = tw(\Gamma^\infty - \widehat{L})$. We can prove this easily using Lemma 2.11 as follows. Let $\sigma \in T\Sigma^\infty - L$. By Lemma 2.11 we know there is a unique $\widehat{\sigma} \in \Gamma^\infty$ such that $\sigma \in tw(\widehat{\sigma})$. Now $\widehat{\sigma}$ cannot belong to $\widehat{L}$, as then we would have $\sigma \in tw(\widehat{L})$. So $\widehat{\sigma} \in \Gamma^\infty - \widehat{L}$ and hence $\sigma \in tw(\Gamma^\infty - \widehat{L})$. Conversely, let $\sigma \in tw(\Gamma^\infty - \widehat{L})$. Then $\sigma \in tw(\widehat{\sigma})$ for some $\widehat{\sigma} \notin \widehat{L}$. But this would mean that $\sigma \notin L$ for otherwise the uniqueness of $\widehat{\sigma}$ would be violated. This completes the proof of the claim above.

Now since regular languages are closed under complement, we know that $\Gamma^\infty - \widehat{L}$ is regular, and hence $T\Sigma^\infty - L$ is a regular interval language.

Closure under union and complementation tell us that regular interval languages are closed under intersection too. It will be helpful, however, to see an explicit construction of this. Let L_1 and L_2 be regular interval languages over Σ. So there exist interval alphabets Γ_1 and Γ_2, and regular sets $\widehat{L}_1 \subseteq \Gamma_1^\infty$ and $\widehat{L}_2 \subseteq \Gamma_2^\infty$ such that $L_1 = tw(\widehat{L}_1)$ and $L_2 = tw(\widehat{L}_2)$. Now, let $\Gamma = prop(\Gamma_1 \cup \Gamma_2)$. Applying Lemma 2.12 we know that there exist regular subsets $\widehat{M}_1$ and $\widehat{M}_2$ of Γ^∞ such that $L_1 = tw(\widehat{L}_1) = tw(\widehat{M}_1)$ and $L_2 = tw(\widehat{L}_2) = tw(\widehat{M}_2)$. We now claim that $L_1 \cap L_2 = tw(\widehat{M}_1 \cap \widehat{M}_2)$. This is again easy to verify using Lemma 2.11 and an argument similar to the complementation case above. Since $\widehat{M}_1 \cap \widehat{M}_2$ is regular, we conclude that $L_1 \cap L_2$ is a regular interval language.

The results above are summarised in the following theorem:

Theorem 2.13 *The class of regular interval languages over an alphabet Σ is closed under the boolean operations of union, intersection and complemen-*

tation. □

2.5 A logical characterisation

In this section we introduce a monadic second order logic interpreted over timed words, which characterises the class of regular interval languages. This logic is called TMSO(Σ) and is parameterised by the alphabet Σ.

Here and in the logics to follow, we assume a supply of individual variables $x, y, \ldots$, and set-variables $X, Y, \ldots$. These variables will range over prefixes (respectively sets of prefixes) of the timed word in question. We will make use of the predicates $Q_a(x)$ (one for each $a \in \Sigma$) and $\Delta(x, I)$, where x is an individual variable and I is an element of $\mathcal{IR}$. The syntax of TMSO(Σ) is given by:

$$\varphi ::= (x \in X) \mid (x < y) \mid Q_a(x) \mid \Delta(x, I) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi \mid \exists X\varphi.$$

A structure for a formula of the logic will be a pair $(\sigma, \mathbb{I})$ where $\sigma \in T\Sigma^{\infty}$ and $\mathbb{I}$ is an interpretation which assigns to each individual variables a prefix of σ, and to each set variable a set of prefixes of σ. We depart slightly from classical monadic logics by using prefixes instead of natural numbers to play the role of positions in a word. Once again, this is more convenient for us given that we are dealing with both finite and infinite words. Correspondingly, $<$ will be interpreted as the the strict prefix relation $\prec$ on finite words.

The satisfaction relation $\sigma \models_{\mathbb{I}} \varphi$ for atomic formulas φ is given as follows:

$$\begin{array}{lll}
\sigma \models_{\mathbb{I}} (x \in X) & \text{iff} & \mathbb{I}(x) \in \mathbb{I}(X) \\
\sigma \models_{\mathbb{I}} (x < y) & \text{iff} & \mathbb{I}(x) \prec \mathbb{I}(y) \\
\sigma \models_{\mathbb{I}} Q_a(x) & \text{iff} & \mathit{action}(\mathbb{I}(x)) = a \\
\sigma \models_{\mathbb{I}} \Delta(x, I) & \text{iff} & \mathbb{I}(x) \text{ is of the form } \tau(a,t) \text{ and } (t - \mathit{time}(\tau)) \in I.
\end{array}$$

The operators $\neg$, $\vee$, and the existential quantifiers $\exists x$ and $\exists X$ are interpreted in the usual manner, as described below. Let $\mathbb{I}$ be an interpretation for variables with respect to σ. Let τ be a prefix of σ. We will use the notation $\mathbb{I}[\tau/x]$ to denote the interpretation which maps x to τ and agrees with $\mathbb{I}$ on all other individual and set variables. Similarly, for a set of prefixes S of σ, the notation $\mathbb{I}[S/X]$ will denote the interpretation which sends X to S, and agrees with $\mathbb{I}$ on all other variables.

$$\begin{array}{lll}
\sigma \models_{\mathbb{I}} \neg\varphi & \text{iff} & \sigma \not\models_{\mathbb{I}} \varphi \\
\sigma \models_{\mathbb{I}} (\varphi \vee \varphi') & \text{iff} & \sigma \models_{\mathbb{I}} \varphi \text{ or } \sigma \models_{\mathbb{I}} \varphi' \\
\sigma \models_{\mathbb{I}} \exists x\varphi & \text{iff} & \text{there exists } \tau \preceq \sigma \text{ such that } \sigma \models_{\mathbb{I}[\tau/x]} \varphi \\
\sigma \models_{\mathbb{I}} \exists X\varphi & \text{iff} & \text{there exists } S \subseteq \mathit{prf}(\sigma) \text{ such that } \sigma \models_{\mathbb{I}[S/X]} \varphi.
\end{array}$$

Given a sentence φ in TMSO(Σ) we define $L(\varphi) = \{\sigma \in T\Sigma^\infty \mid \sigma \models \varphi\}$.

Example 2.14 Let $\Sigma = \{a, b\}$. Then the following TMSO(Σ)-sentence describes the language accepted by the automaton $\mathcal{A}$ in Example 2.7.

$$\begin{aligned}\varphi_1 \quad = \quad & \exists x(Q_b(x) \wedge \Delta(x, [1,2]) \wedge \\ & \quad \forall y((y < x) \Rightarrow (Q_a(y) \wedge \Delta(y, [1,1]))) \wedge \\ & \quad \forall y((x < y) \Rightarrow Q_a(y))).\end{aligned}$$

□

Theorem 2.15 *Let $L \subseteq T\Sigma^\infty$. Then L is a regular interval language iff $L = L(\varphi)$ for some sentence φ in* TMSO(Σ).

We will devote the rest of this section to the proof of this theorem. We begin with Büchi's monadic second order logic characterisation of regular languages, which we will make use of crucially. We recall that for an alphabet A, Büchi's monadic second order logic (denoted here by MSO(A)) is given as follows:

$$\varphi ::= (x \in X) \mid (x < y) \mid Q_a(x) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x \varphi \mid \exists X \varphi.$$

A structure for this logic is a pair of the form $(\sigma, \mathbb{I})$ where $\sigma \in A^\infty$ and $\mathbb{I}$ assigns to individual and set variables, non-empty prefixes and sets of non-empty prefixes of σ respectively. The semantics of the logic is given in a similar manner to that of TMSO(Σ). In particular, the atomic formula $Q_a(x)$—here a is required to be in A—is interpreted as follows:

$$\sigma \models_{\mathbb{I}} Q_a(x) \text{ iff } \mathbb{I}(x) = \tau a \text{ for some } \tau \preceq \sigma.$$

As usual, for a sentence φ in MSO(A) we set

$$L(\varphi) = \{\sigma \in A^\infty \mid \sigma \models \varphi\}.$$

Büchi's result states that a language $L \subseteq A^\infty$ is accepted by a Büchi automaton over the alphabet A iff $L = L(\varphi)$ for some sentence φ in MSO(A).

Now, given a formula $\varphi \in$ TMSO(Σ) we show how to translate it to a formula $t\text{-}s(\varphi) \in$ MSO(Γ), for a suitably defined interval alphabet Γ. The translation will preserve—in a sense to be made precise—the timed models of φ. (The name *t-s* is a mnemonic for "timed-to-symbolic".) Let Γ be any proper interval alphabet over Σ such that for each $a \in \Sigma$, Γ_a covers

$$voc(\varphi) = \{I \mid \varphi \text{ has a subformula of the form } \Delta(x, I)\}.$$

Note that $\Sigma \times prop(voc(\varphi))$ is at least one such Γ. Then $t\text{-}s(\varphi)$ (w.r.t. Γ) is obtained from φ by replacing sub-formulas of the form $Q_a(x)$ by the formula

$$\bigvee_{(b,I)\in\Gamma,\ b=a} Q_{(b,I)}(x),$$

and sub-formulas of the form $\Delta(x, I)$ by the formula

$$\bigvee_{(a,I')\in\Gamma,\ I'\subseteq I} Q_{(a,I')}(x).$$

Lemma 2.16 *Let $\varphi \in \mathrm{TMSO}(\Sigma)$ and let Γ be a proper interval alphabet based on Σ with the property that for each $a \in \Sigma$, Γ_a covers $voc(\varphi)$. Let $\widehat{\sigma} \in \Gamma^\infty$ and $\sigma \in T\Sigma^\infty$ be such that $\sigma \in tw(\widehat{\sigma})$. Suppose further that $\mathbb{I}$ is an interpretation for variables with respect to σ, and $\widehat{\mathbb{I}}$ is the corresponding interpretation for variables w.r.t. $\widehat{\sigma}$, given by $\widehat{\mathbb{I}}(x) = \widehat{\tau}$ where $\widehat{\tau} \preceq \widehat{\sigma}$ such that $|\widehat{\tau}| = |\mathbb{I}(x)|$, and $\widehat{\mathbb{I}}(X) = \{\widehat{\mathbb{I}}(x) \mid x \in X\}$. Then*

1. *$\sigma \models_{\mathbb{I}} \varphi$ iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\varphi)$.*
2. *If φ is a sentence, then $L(\varphi) = tw(L(t\text{-}s(\varphi)))$.*

Proof

1. We prove the statement by induction on the structure of φ. We begin with the atomic formulas as base cases.

   Case $\varphi = (x \in X)$: We know $\sigma \models_{\mathbb{I}} (x \in X)$ iff $\mathbb{I}(x) \in \mathbb{I}(X)$. This in turn is true iff $\widehat{\mathbb{I}}(x) \in \widehat{\mathbb{I}}(X)$, which is true iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} (x \in X)$.

   Case $\varphi = (x < y)$: We know $\sigma \models_{\mathbb{I}} (x < y)$ iff $\mathbb{I}(x) \prec \mathbb{I}(y)$. But this is true iff $\widehat{\mathbb{I}}(x) \prec \widehat{\mathbb{I}}(y)$, which is true iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} (x < y)$.

   Case $\varphi = Q_a(x)$: We know $\sigma \models_{\mathbb{I}} Q_a(x)$ iff $action(\mathbb{I}(x)) = a$. But since $\sigma \in tw(\widehat{\sigma})$, we know that this holds iff $\widehat{\mathbb{I}}(x) = \widehat{\tau}(a, I)$ for some $\widehat{\tau}$ and I such that $(a, I) \in \Gamma$. This in turn holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} \bigvee_{(b,I)\in\Gamma,\ b=a} Q_{(b,I)}(x)$.

   Case $\varphi = \Delta(x, I)$: Let $\sigma \models_{\mathbb{I}} \Delta(x, I)$. Then we know that $\mathbb{I}(x) = \tau(a, t)$ for some τ, a, t such that $(t - time(\tau)) \in I$. Further, since $\sigma \in tw(\widehat{\sigma})$, we know that $\widehat{\mathbb{I}}(x) = \widehat{\tau}(a, I')$ for some I' such that $(a, I') \in \Gamma$, and $(t - time(\tau)) \in I'$. Using the fact that Γ is proper and covers $voc(\varphi)$, and $(t - time(\tau)) \in I \cap I'$, it must be the case that $I' \subseteq I$. Hence

   $$\widehat{\sigma} \models_{\widehat{\mathbb{I}}} \bigvee_{(a,I')\in\Gamma,\ I'\subseteq I} Q_{(a,I')}(x).$$

Conversely, let

$$\widehat{\sigma} \models_{\widehat{\mathbb{I}}} \bigvee_{(a,I')\in\Gamma,\ I'\subseteq I} Q_{(a,I')}(x).$$

Then $\widehat{\mathbb{I}}(x) = \widehat{\tau}(a, I')$ for some $(a, I') \in \Gamma$ with $I' \subseteq I$. Since $\sigma \in tw(\widehat{\sigma})$ it must be the case that $\mathbb{I}(x) = \tau(a, t)$ such that $(t - time(\tau)) \in I'$. Thus $(t - time(\tau)) \in I$, and it follows that $\sigma \models_{\mathbb{I}} \Delta(x, I)$.

For the induction step, assume the statement holds for formulas φ' and φ''.

Case $\neg\varphi'$: $\sigma \models_{\mathbb{I}} \neg\varphi'$ iff $\sigma \not\models_{\mathbb{I}} \varphi'$. By the induction hypothesis, this holds iff $\widehat{\sigma} \not\models_{\widehat{\mathbb{I}}} t\text{-}s(\varphi')$. Since $\neg t\text{-}s(\varphi') = t\text{-}s(\neg\varphi')$, this holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\neg\varphi')$.

Case $(\varphi' \vee \varphi'')$: $\sigma \models_{\mathbb{I}} (\varphi' \vee \varphi'')$ iff $\sigma \models_{\mathbb{I}} \varphi'$ or $\sigma \models_{\mathbb{I}} \varphi''$. By the induction hypothesis, this holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\varphi')$ or $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\varphi'')$. Since $t\text{-}s(\varphi' \vee \varphi'') = (t\text{-}s(\varphi') \vee t\text{-}s(\varphi''))$, this holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\varphi' \vee \varphi'')$.

Case $\exists x\varphi'$: $\sigma \models_{\mathbb{I}} \exists x\varphi'$ iff there exists a prefix τ of σ such that $\sigma \models_{\mathbb{I}[\tau/x]} \varphi'$. Let $\widehat{\tau}$ be the prefix of $\widehat{\sigma}$ such that $|\widehat{\tau}| = |\tau|$. By the induction hypothesis, it follows that $\widehat{\sigma} \models_{\widehat{\mathbb{I}}[\widehat{\tau}/x]} t\text{-}s(\varphi')$. Once again, since $t\text{-}s(\exists x\varphi') = \exists x(t\text{-}s(\varphi'))$, this holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\exists x\varphi')$.

Case $\exists X\varphi'$: $\sigma \models_{\mathbb{I}} \exists X\varphi'$ iff there exists a set of prefixes S of σ such that $\sigma \models_{\mathbb{I}[S/X]} \varphi'$. Let $\widehat{S}$ be the set of prefixes of $\widehat{\sigma}$ given by $\{\widehat{\tau} \mid \tau \in S, |\widehat{\tau}| = |\tau|\}$. By the induction hypothesis, it follows that $\widehat{\sigma} \models_{\widehat{\mathbb{I}}[\widehat{S}/X]} t\text{-}s(\varphi')$. Once again, since $t\text{-}s(\exists X\varphi') = \exists X(t\text{-}s(\varphi'))$, this holds iff $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} t\text{-}s(\exists X\varphi')$.

2. This is easy to see once we have (1) above. Let $\sigma \models \varphi$. Then, again using properties of proper alphabets, there exists a $\widehat{\sigma} \in \Gamma^\infty$ such that $\sigma \in tw(\widehat{\sigma})$. Using (1) above, we have $\widehat{\sigma} \in L(t\text{-}s(\varphi))$ and hence $\sigma \in tw(L(t\text{-}s(\varphi)))$. Conversely, if $\sigma \in tw(\widehat{\sigma})$ and $\widehat{\sigma} \models t\text{-}s(\varphi)$, then by (1) again we have that $\sigma \models \varphi$, and hence $\sigma \in L(\varphi)$.

$\square$

We now show how we can associate a formula $s\text{-}t(\widehat{\varphi}) \in \text{TMSO}(\Sigma)$ with a formula $\widehat{\varphi} \in \text{MSO}(\Gamma)$, such that the translated formula preserves timed models. The formula $s\text{-}t(\widehat{\varphi})$ is obtained by replacing atomic sub-formulas in $\widehat{\varphi}$ of the form $Q_{(a,I)}(x)$ by the formula

$$Q_a(x) \wedge \Delta(x, I).$$

We show the following result:

Lemma 2.17 *Let Γ be a proper interval alphabet based on Σ and let $\widehat{\varphi} \in$ MSO(Γ). Let $\widehat{\sigma} \in \Gamma^\infty$ and $\sigma \in T\Sigma^\infty$ such that $\sigma \in tw(\widehat{\sigma})$. Suppose further that $\widehat{\mathbb{I}}$ is an interpretation for variables w.r.t. $\widehat{\sigma}$ and let $\mathbb{I}$ be the corresponding interpretation w.r.t. σ—i.e. $\mathbb{I}(x) = \tau$, where $\tau \preceq \sigma$ such that $|\tau| = |\widehat{\mathbb{I}}(x)|$, and similarly for set variables. Then*

1. $\sigma \models_{\mathbb{I}} s\text{-}t(\widehat{\varphi})$ *iff* $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} \widehat{\varphi}$.

2. *If φ is a sentence, then we have* $L(s\text{-}t(\widehat{\varphi})) = tw(L(\widehat{\varphi}))$.

Proof

1. We proceed by induction on the structure of $\widehat{\varphi}$.

   Case $\widehat{\varphi} = Q_{(a,I)}(x)$: Thus $s\text{-}t(\widehat{\varphi}) = Q_a(x) \wedge \Delta(x, I)$. Let $\sigma \models_{\mathbb{I}} Q_a(x) \wedge \Delta(x, I)$. Then $action(\mathbb{I}(x)) = a$ and $\mathbb{I}(x)$ is of the form $\tau(a, t)$ with $(t - time(\tau)) \in I$. But this would imply that $\widehat{\mathbb{I}}(x) = \tau(a, I')$ for some I' such that $(t - time(\tau)) \in I'$. Since Γ is proper, this would mean $I = I'$. Hence $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} Q_{(a,I)}(x)$.

   Conversely if $\widehat{\sigma} \models_{\widehat{\mathbb{I}}} Q_{(a,I)}(x)$ then we know that $\widehat{\mathbb{I}}(x) = \widehat{\tau}(a, I)$ for some $\widehat{\tau}$. But since $\sigma \in tw(\widehat{\sigma})$ we must have $\mathbb{I}(x) = \tau(a, t)$ for some τ, t such that $(t - time(\tau)) \in I$. This means that $\sigma \models_{\mathbb{I}} Q_a(x) \wedge \Delta(x, I)$.

   The remaining cases are routine.

2. For a sentence $\widehat{\varphi}$ to prove that $L(s\text{-}t(\widehat{\varphi})) = tw(L(\widehat{\varphi}))$, let $\sigma \models s\text{-}t(\widehat{\varphi})$. Then, since Γ is a proper alphabet, there exists a $\widehat{\sigma} \in \Gamma^\infty$ such that $\sigma \in tw(\widehat{\sigma})$. By (1) above, we know that $\widehat{\sigma} \models \widehat{\varphi}$. Thus $\widehat{\sigma} \in L(\widehat{\varphi})$ and $\sigma \in tw(L(\widehat{\varphi}))$.

   Conversely, suppose $\sigma \in tw(L(\widehat{\varphi}))$. Then $\sigma \in tw(\widehat{\sigma})$ for some $\widehat{\sigma} \in \Gamma^\infty$ such that $\widehat{\sigma} \models \widehat{\varphi}$. By (1) again, we know that $\sigma \models s\text{-}t(\widehat{\varphi})$, and hence $\sigma \in L(s\text{-}t(\widehat{\varphi}))$.

□

We are now in a position to provide a proof of Theorem 2.15. Let L be a regular interval language over Σ. Using Lemma 2.12 we know that there exists a proper interval alphabet Γ based on Σ and a regular subset $\widehat{L}$ of Γ^∞ such that $L = tw(\widehat{L})$. Büchi's theorem tells us that there exists an MSO(Γ)-sentence $\widehat{\varphi}$ such that $L(\widehat{\varphi}) = \widehat{L}$. Hence $L = tw(L(\widehat{\varphi}))$. Thus, by Lemma 2.17, we have a TMSO(Σ)-sentence, namely $\varphi = s\text{-}t(\widehat{\varphi})$, such that $L = L(\varphi)$.

Conversely, let φ be a TMSO(Σ)-sentence. Then, using Lemma 2.16, we know that there exists a proper interval alphabet Γ and a formula $\widehat{\varphi} = t\text{-}s(\varphi)$

in MSO(Γ), such that $L(\varphi) = tw(L(\widehat{\varphi}))$. Using Büchi's theorem once more, we are assured of an interval automaton $\mathcal{A}$ over Γ such that $L_{sym}(\mathcal{A}) = L(\widehat{\varphi})$. Thus $\mathcal{A}$ is such that

$$L(\mathcal{A}) = tw(L_{sym}(\mathcal{A})) = tw(L(\widehat{\varphi})) = L(\varphi).$$

This completes the proof of the theorem.

Chapter 3

Product interval automata

As seen in the last chapter, the class of interval automata are a rather restricted class of timed automata which can be captured by timed automata with a single clock which is reset along every transition. Here we increase the expressiveness of interval automata by considering networks of such automata. We have an alphabet of actions which is distributed over locations. Each location runs an interval automaton over its local alphabet. Communication takes place between these automata by enforcing that locations synchronize on common actions. In terms of timed automata, product interval automata will correspond to multi-clock automata, but where the reading and resetting of clocks is controlled by the topology of the network. It is through the distribution of actions that we gain the added expressive power. As we demonstrate in Chapter 5, product interval automata are indeed capable of modelling some non-trivial timed systems.

In Sections 3.5 and 3.6 we will show that product interval automata are closed under boolean operations, and admit a logical characterisation as well.

3.1 Preliminaries

Let $\mathcal{P} = \{1, 2, \ldots, k\}$ be a finite set of agents, or locations. A $\mathcal{P}$-distributed alphabet is a family $\widetilde{\Sigma} = \{\Sigma_i\}_{i \in \mathcal{P}}$ where each Σ_i is a finite set of actions. We set $\Sigma = \bigcup_{i \in \mathcal{P}} \Sigma_i$ and call it the global alphabet induced by $\widetilde{\Sigma}$. In general, for a $\mathcal{P}$-distributed alphabet $\widetilde{\Delta}$ we will follow the convention that $\Delta = \bigcup_{i \in \mathcal{P}} \Delta_i$. The set of agents that participate in each occurrence of the action a will be denoted by $loc(a)$ and is given by: $loc(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$.

Through the rest of this chapter we fix such a set of agents $\mathcal{P}$ and a $\mathcal{P}$-distributed alphabet $\widetilde{\Sigma}$.

We will be considering timed languages in a distributed setting. Towards this end we must re-define the notion of a timed word over a *distributed* alphabet. This is mainly to allow us to model the simultaneous occurrence of *independent* actions. Actions a and b are said to be independent if $loc(a) \cap loc(b) = \emptyset$.

Definition 3.1 *A timed word σ over $\widetilde{\Sigma}$ is an element of $(\Sigma \times \mathbb{R}^{>0})^\infty$ such that:*

(i) if $\tau(a,t)(b,t')$ is a prefix of σ then $t \leq t'$ (non-decreasing).

(ii) if $\tau(a,t)\tau'(b,t')$ is a prefix of σ with $t = t'$, then $loc(a) \cap loc(b) = \emptyset$ (simultaneous actions must be independent).

(iii) if σ is infinite, then for each $t \in \mathbb{R}^{>0}$ there exists a prefix $\tau(a,t')$ of σ such that $t < t'$ (progressiveness).

We let $T\widetilde{\Sigma}^*$ and $T\widetilde{\Sigma}^\omega$ denote the set of finite and infinite timed words over $\widetilde{\Sigma}$ respectively, and set $T\widetilde{\Sigma}^\infty = T\widetilde{\Sigma}^* \cup T\widetilde{\Sigma}^\omega$.

Let $\sigma \in T\widetilde{\Sigma}^\infty$. Then $\sigma \restriction i$ is the i-projection of σ. It is the timed word over Σ_i obtained by erasing from σ all appearances of letters of the form (a,t) with $a \notin \Sigma_i$. It is easy to check that $\sigma \restriction i$ does indeed belong to $T\Sigma_i^\infty$.

For a finite timed word τ, we will use $time_i(\tau)$ to denote the time of occurrence of the last i-action in τ. More formally:

Definition 3.2 *Let $\tau \in T\widetilde{\Sigma}^*$. Then $time_i(\tau)$ is given inductively by:*

- *$time_i(\epsilon) = 0$.*
- *$time_i(\tau(a,t)) = t$ if $a \in \Sigma_i$, and equals $time_i(\tau)$ otherwise.*

3.2 Product interval automata

Definition 3.3 *A* product interval automaton *over $\widetilde{\Sigma}$ is a structure $(\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q_{in})$, where for each i, $\mathcal{A}_i$ is a structure $(Q_i, \longrightarrow_i, F_i, G_i)$ where*

- *Q_i is a finite set of states*
- *$\longrightarrow_i$ is a finite subset of $Q_i \times (\Sigma_i \times \mathcal{IR}) \times Q_i$*
- *$F_i, G_i \subseteq Q_i$ are, respectively, finitary and infinitary acceptance state sets.*

$Q_{in} \subseteq Q = Q_1 \times \cdots \times Q_k$ is a set of global initial states.

Let $\mathcal{A} = (\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q_{in})$ be a product interval automaton over $\widetilde{\Sigma}$ and let $\sigma \in T\widetilde{\Sigma}^{\infty}$. Then a run of $\mathcal{A}$ over σ is a map $\rho : \mathit{prf}(\sigma) \to Q$ such that

1. $\rho(\epsilon) \in Q_{in}$
2. for each prefix $\tau(a, t)$ of σ we have
   (a) for each $i \in \mathit{loc}(a)$, there exists a transition $\rho(\tau)[i] \xrightarrow{(a,I)}_i \rho(\tau(a,t))[i]$ with $(t - \mathit{time}_i(\tau)) \in I$.
   (b) for each $i \notin \mathit{loc}(a)$ we have $\rho(\tau)[i] = \rho(\tau(a,t))[i]$.

A run ρ of $\mathcal{A}$ on σ is accepting iff for each $i \in \mathcal{P}$ either

(i) $\sigma \restriction i$ is finite and $\rho(\tau)[i] \in F_i$ for any prefix τ of σ such that $\tau \restriction i = \sigma \restriction i$, or

(ii) $\sigma \restriction i$ is infinite and $\rho(\tau)[i] \in G_i$ for infinitely many $\tau \in \mathit{prf}(\sigma)$.

We set $L(\mathcal{A})$ to be the set of words in $T\widetilde{\Sigma}^{\infty}$ accepted by $\mathcal{A}$ (i.e. those on which $\mathcal{A}$ has an accepting run).

Notice that the components $\mathcal{A}_i$ of $\mathcal{A}$ are interval automata over Σ_i, except that they have no start states. The global initial states determine which combination of states the component automata can start in. Thus, if $\mathcal{A}_i$ are interval automata over Σ_i respectively, with each $\mathcal{A}_i = (Q_i, \longrightarrow_i, Q^i_{in}, F_i, G_i)$, and $Q_{in} \subseteq Q_1 \times \cdots \times Q_k$, then we will often use $(\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q_{in})$ to denote the product interval automaton $(\{\mathcal{B}_i\}_{i \in \mathcal{P}}, Q_{in})$, where $\mathcal{B}_i = (Q_i, \longrightarrow_i, F_i, G_i)$.

Example 3.4 Consider the language L over $\Sigma = \{a, b\}$ given in Lemma 2.10. We saw that L is not a regular interval language over Σ. However we can easily give a product interval automaton $\mathcal{A}$ over the distributed alphabet $\widetilde{\Sigma} = \{\{a, b\}, \{b\}\}$ such that $L(\mathcal{A}) = L$. Such an automaton is given in Figure 3.1. □

3.3 Checking emptiness

Unlike interval automata, product interval automata cannot be checked for emptiness by simply examining the symbolic language they accept. We can however use the region construction method of [AD94]. Rather than do this construction explicitly, we will show how to simulate a product interval automaton using a timed Büchi automaton. This will then give us a way of checking emptiness for our automata.

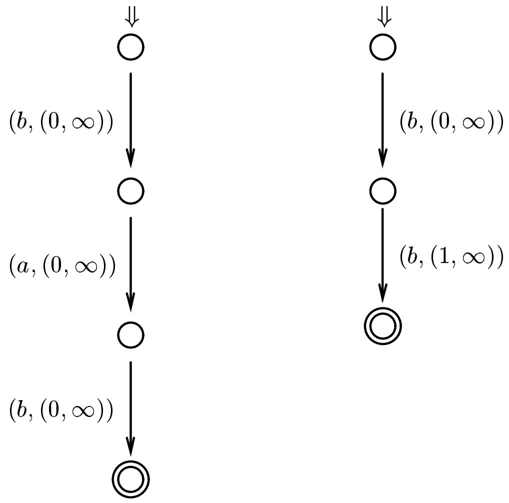

Figure 3.1: A product interval automaton over $\widetilde{\Sigma} = \{\{a,b\},\{b\}\}$.

Due to the fact that our automata accept "non-decreasing" timed words, as opposed to the strictly increasing time sequences of [AD94], we will need to make use of a more generous semantics for timed automata. We enlarge the universe of timed words to include timed sequences over Σ which may be non-decreasing. Thus, we say $\sigma \in \Sigma \times \mathbb{R}^{>0}$ is a non-decreasing timed word if it satisfies the following conditions:

(i) if $\tau(a,t)(b,t')$ is a prefix of σ then $t \leq t'$ (non-decreasing).

(ii) if σ is infinite, then for each $t \in \mathbb{R}^{>0}$ there exists a prefix $\tau(a,t')$ of σ such that $t < t'$ (progressiveness).

Let $T_{nd}\Sigma^{\infty}$ denote the set of non-decreasing timed words over Σ.

We also extend the definition of a run of a timed automaton (Section 2.1) to such words too. Let $L_{nd}(\mathcal{B})$ denote the set of non-decreasing timed words accepted by a timed automaton $\mathcal{B}$.

The region automaton $R(\mathcal{B})$ must be modified slightly to accept $untime(L_{nd}(\mathcal{B}))$. This can be done by allowing every region to be a time successor of itself.

Now let $\mathcal{A} = (\{\mathcal{A}_i\}_{i\in\mathcal{P}}, Q_{in})$ be a product interval automaton, with each $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, G_i)$. We assume here each local finitary final state is "terminal" in the sense that there are no outgoing edges from them. This does not involve any loss of generality.

As a first step we define a TBA $\mathcal{B}$ with the property that $L_{nd}(\mathcal{B}) \cap T\widetilde{\Sigma}^{\infty} = L(\mathcal{A})$. We define $\mathcal{B} = (S, \longrightarrow, S_{in}, C, F, G)$ where:

- The set of states is $S = (\prod_{i=1}^{k} Q_i) \times \{0,\ldots,k\}$.

- The set of clocks is $C = \{x_i \mid i \in \mathcal{P}\}$.
- Its transition relation is given as follows. We have $(q, l) \xrightarrow[X]{a,g} (q', m)$ iff the following conditions hold:
    1. $X = \{x_i \mid i \in loc(a)\}$,
    2. for each $i \in loc(a)$ there exists transitions $q[i] \xrightarrow{(a,I_i)}_i q'[i]$ such that $g = \bigwedge_{i \in loc(a)}(x_i \in I_i)$.
    3. for each $i \notin loc(a)$, $q[i] = q'[i]$.
    4. $m = (l+1) \bmod (k+1)$ if $q'[l] \in F_l \cup G_l$ or $l = 0$; otherwise $m = l$.
- The initial states are given by $S_{in} = Q_{in} \times \{0\}$.
- The finitary final states F are given by $F = (\prod_{i=1}^{k} F_i) \times \{0, \ldots, k\}$.
- The infinitary final states are given by $G = \{(q, l) \in S \mid l = 0\}$.

It is not difficult to argue that $L_{nd}(\mathcal{B}) \cap T\widetilde{\Sigma}^{\infty} = L(\mathcal{A})$.

Next we get rid of words in $L_{nd}(\mathcal{B})$ which are *not* in $T\widetilde{\Sigma}^{\infty}$. To do this we intersect $\mathcal{B}$ with a TBA $\mathcal{B}'$ which accepts precisely the language $T\widetilde{\Sigma}^{\infty}$. $\mathcal{B}'$ will have a single clock, and its set of states will be $\{0,1\}^k$. The clock is used to check whether an action is a zero-time one, while the bits in the state keep track of the components which have already taken part in the last stretch of zero-time actions.

Formally, we define $\mathcal{B}' = (S', \longrightarrow', C', S'_{in}, F', G')$ where $S' = \{0,1\}^k$, and $C' = \{x\}$, $S'_{in} = \{\vec{0}\}$ where $\vec{0} = (0, \ldots, 0)$. The transitions are one of two forms:

- $v \xrightarrow[\{x\}]{a,\,(x \in (0,\infty))}{}' \vec{0}$, or,
- $v \xrightarrow[\{x\}]{a,\,(x \in [0,0])}{}' v'$ where v and v' are such that each of the following are satisfied:
    - $v[i] = 0$ for each $i \in loc(a)$.
    - $v'[i] = 1$ if $i \in loc(a)$; $v'[i] = v[i]$ otherwise.

  The final states are given by $F' = G' = S'$.

Let $\mathcal{B}''$ be the TBA obtained by intersecting $\mathcal{B}$ and $\mathcal{B}'$. Then $\mathcal{B}''$ is the TBA promised above, with $L_{nd}(\mathcal{B}'') = L(\mathcal{A})$.

Using the check for emptiness of the language accepted by TBA's outlined in Section 2.1, we can check if $L(\mathcal{A})$ is empty or not.

We now analyse the time complexity of checking the emptiness of $\mathcal{A}$ via this route. The number of states and edges in $\mathcal{B}''$ can be seen to be at most $2^{O(k)}$ times the number of global states and edges of $\mathcal{A}$. Thus the number of states in $\mathcal{B}''$ is at most

$$2^{O(k)} \cdot |Q|$$

and the number of edges is at most

$$2^{O(k)} \cdot |E|$$

where $|Q| = \prod_{i=1}^{k} |Q_i|$ and $|E| = \prod_{i=1}^{k} |E_i|$, with Q_i, E_i being the state and edge set of the i-th component of $\mathcal{A}$. Further, the constants used in the clock constraints are the same as in $\mathcal{A}$, and the number of clocks is $k+1$.

Thus, using the time bound obtained in Section 2.1, the emptiness check for $\mathcal{A}$ takes time

$$|\Sigma| \cdot O(|Q| + |E|) \cdot 2^{O(k)} \cdot k! \cdot \prod_{i=1}^{k}(c_i + 1).$$

(The factor of $|\Sigma|$ comes in as we need to examine the distribution of Σ while generating the global transition relation of $\mathcal{A}$.) A loose upper bound for the above expression can be seen to be $|\Sigma| \cdot 2^{O(|\mathcal{A}|^2)}$.

3.4 Product interval languages

In this section we give an alternate characterisation of languages accepted by product interval automata in terms of a timed version of the parallel composition operator $\otimes$. For convenience, we will continue to use the same symbol $\otimes$ to denote the timed version also. The characterisation we obtain will play an important role in the subsequent sections.

Let $L_i \subseteq T\Sigma_i^\infty$ for each $i \in \mathcal{P}$. Then the *direct product* of $L_1, \ldots, L_k$, written $\otimes(L_1, \ldots, L_k)$, is defined as:

$$\otimes(L_1, \ldots, L_k) = \{\sigma \in T\widetilde{\Sigma}^\infty \mid \sigma\restriction i \in L_i \text{ for each } i \in \mathcal{P}\}.$$

$L \subseteq T\widetilde{\Sigma}^\infty$ is a *regular direct product interval language* over $\widetilde{\Sigma}$ if $L = \otimes(L_1, \ldots, L_k)$ for some regular interval languages L_i over Σ_i. Finally, we say $L \subseteq T\widetilde{\Sigma}^\infty$ is a *regular product interval language* over $\widetilde{\Sigma}$ if L is the finite union of regular direct product interval languages over $\widetilde{\Sigma}$.

The aim of this section is to establish the following result:

Theorem 3.5 *Let $L \subseteq T\widetilde{\Sigma}^{\infty}$. Then L is a regular product interval language over $\widetilde{\Sigma}$ iff L is accepted by a product interval automaton over $\widetilde{\Sigma}$.*

We first prove a couple of results that will be helpful in proving the theorem.

Lemma 3.6 *Let $\mathcal{A} = (\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q_{in})$ be a product interval automaton over $\widetilde{\Sigma}$. Let $Q_{in} = \{q_1, \ldots, q_m\}$ for some $m \geq 1$. For $j \in \{1, \ldots, m\}$ let $\mathcal{A}^j$ denote the product interval automaton $(\{\mathcal{A}_i\}_{i \in \mathcal{P}}, \{q_j\})$. Then*

$$L(\mathcal{A}) = \bigcup_{j \in \{1, \ldots, m\}} L(\mathcal{A}^j).$$

Proof Follows easily from the definition of the language accepted by a product interval automaton. □

Lemma 3.7 *For each $i \in \mathcal{P}$ let $\mathcal{A}_i = (Q_i, \longrightarrow_i, Q^i_{in}, F_i, G_i)$ be an interval automaton over Σ_i. Let $\mathcal{A}$ be the product interval automaton*

$$(\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q^1_{in} \times \cdots \times Q^k_{in}).$$

Then

$$L(\mathcal{A}) = \otimes(L(\mathcal{A}_1), \ldots, L(\mathcal{A}_k)).$$

Proof We first show that $L(\mathcal{A}) \subseteq \otimes(L(\mathcal{A}_1), \ldots, L(\mathcal{A}_k))$. Let $\sigma \in L(\mathcal{A})$. Then we know that there exists an accepting run ρ of $\mathcal{A}$ on σ. We can use ρ to define, for each $i \in \mathcal{P}$, a run ρ_i of the interval automaton $\mathcal{A}_i$ on $\sigma \restriction i$. We do this by setting, for each prefix τ_i of $\sigma \restriction i$, $\rho_i(\tau_i) = \rho(\tau')[i]$ where τ' is any prefix of σ such that $\tau' \restriction i = \tau_i$. It is easy to check that if τ' and τ'' are two prefixes of σ such that $\tau' \restriction i = \tau'' \restriction i$ then $\rho(\tau')[i] = \rho(\tau'')[i]$, and hence ρ_i is a well-defined map from $prf(\sigma \restriction i)$ to Q_i. To check that ρ_i is indeed a run of $\mathcal{A}_i$ on $\sigma \restriction i$ we must verify that

1. $\rho_i(\epsilon) \in Q^i_{in}$.

   This is true since $\rho(\epsilon) \in Q^1_{in} \times \cdots \times Q^k_{in}$.

2. For each prefix $\tau_i(a, t)$ of $\sigma \restriction i$, $\rho_i(\tau_i) \xrightarrow{(a,I)}_i \rho_i(\tau_i(a, t))$ for some I such that $t - time(\tau_i) \in I$.

   To verify this, let $\tau_i(a, t)$ be a prefix of $\sigma \restriction i$. Then there must exist a prefix τ of σ such that $\tau(a, t)$ is a prefix of σ and $\tau \restriction i = \tau_i$. Using the fact that ρ is a run of $\mathcal{A}$ on σ we have that there exists an I such that $\rho(\tau)[i] \xrightarrow{(a,I)}_i \rho(\tau(a, t))[i]$ and $t - time_i(\tau) \in I$. But it is easy to see that $time_i(\tau) = time(\tau_i)$. Hence we have the required property.

Further, each ρ_i is an accepting run of $\mathcal{A}_i$ on $\sigma \restriction i$. This is again easy to verify given that ρ is accepting. Thus $\sigma \restriction i \in L(\mathcal{A}_i)$ for each i, and hence $\sigma \in \otimes(L(\mathcal{A}_1), \ldots, L(\mathcal{A}_k))$.

Conversely, suppose $\sigma \in \otimes(L(\mathcal{A}_1), \ldots, L(\mathcal{A}_k))$. So there exist accepting runs ρ_i of $\mathcal{A}_i$ on $\sigma \restriction i$ for each $i \in \mathcal{P}$. Using these ρ_i's we can piece together a run ρ of $\mathcal{A}$ on σ, given by: $\rho(\tau)[i] = \rho_i(\tau \restriction i)$. Again, it is routine to check that ρ is indeed a run of $\mathcal{A}$ on σ, and is accepting. □

Returning to the proof of Theorem 3.5, suppose now that L is a regular product interval language over $\widetilde{\Sigma}$. Then there exist $L_i^j \subseteq T\Sigma_i^\infty$ for $i \in \mathcal{P}$ and $j \in \{1, \ldots, m\}$ $(m \geq 1)$ such that each L_i^j is a regular interval language over Σ_i and

$$L = \bigcup_{j \in \{1,\ldots,m\}} \otimes(L_1^j, \ldots, L_k^j).$$

Since each L_i^j is a regular interval language over Σ_i, there exist interval automata $\mathcal{A}_i^j$ over Σ_i such that $L(\mathcal{A}_i^j) = L_i^j$. Let each $\mathcal{A}_i^j$ be of the form $(Q_i^j, \longrightarrow_i^j, (Q_{in})_i^j, F_i^j, G_i^j)$. Now let $\mathcal{B}_i$ denote the disjoint union of the automata $\mathcal{A}_i^1, \ldots, \mathcal{A}_i^m$ (viewed as labelled graphs). Let us use $(\mathcal{B}_i, (Q_{in})_i^j)$ to denote the interval automaton with the underlying structure of $\mathcal{B}_i$ and $(Q_{in})_i^j$ as the set of initial states. Then it is easy to see that $L(\mathcal{B}_i, (Q_{in})_i^j) = L(\mathcal{A}_i^j)$. Thus we have

$$\begin{aligned} L &= \textstyle\bigcup_{j \in \{1,\ldots,m\}} \otimes(L_1^j, \ldots, L_k^j) \\ &= \textstyle\bigcup_{j \in \{1,\ldots,m\}} \otimes(L(\mathcal{B}_1, (Q_{in})_1^j), \ldots, L(\mathcal{B}_k, (Q_{in})_k^j)) \\ &= \textstyle\bigcup_{j \in \{1,\ldots,m\}} L(\{\mathcal{B}_i\}_{i \in \mathcal{P}}, (Q_{in})_1^j \times \cdots \times (Q_{in})_k^j) \text{ (using Lemma 3.7)} \\ &= L(\{\mathcal{B}_i\}_{i \in \mathcal{P}}, Q_{in})) \end{aligned}$$

where $Q_{in} = \bigcup_{j \in \{1,\ldots,m\}} (Q_{in})_1^j \times \cdots \times Q_{in})_k^j$. This last step follows from Lemma 3.6. Thus L is accepted by a product interval automaton.

The converse direction follows in a similar manner. Let $\mathcal{A} = (\{\mathcal{A}_i\}_{i \in \mathcal{P}}, Q_{in})$ be a product interval automaton over $\widetilde{\Sigma}$, with each $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, G_i)$. Let $L = L(\mathcal{A})$. We will show that L is a regular product interval language over $\widetilde{\Sigma}$. Let $Q_{in} = \{q_1, \ldots, q_m\}$ $(m \geq 1)$. Then, by Lemma 3.6, we know

$$L(\mathcal{A}) = \bigcup_{j=1}^{m} L(\{\mathcal{A}_i\}_{i \in \mathcal{P}}, \{q_j\}).$$

Let $\mathcal{B}_i^j$ be the interval automaton over Σ_i given by

$$\mathcal{B}_i^j = (Q_i, \longrightarrow_i, \{q_j[i]\}, F_i, G_i),$$

for $i \in \mathcal{P}$, $j \in \{1, \ldots, m\}$. Then, by Lemma 3.7, we have

$$L = \bigcup_{j=1}^{m} \otimes(L(\mathcal{B}_1^j), \ldots, L(\mathcal{B}_k^j)).$$

Since for each i and j, $L(\mathcal{B}_i^j)$ is a regular interval language over Σ_i, we conclude that L is a regular product interval language over $\widetilde{\Sigma}$.

3.5 Closure under boolean operations

The fact that regular product interval languages are closed under union follows directly from the definition of regular product interval languages as finite unions of regular direct product interval languages.

To show closure under intersection and complementation, it is sufficient to show that these operations on regular *direct* product interval languages do not take us out of the class of regular product interval languages.

Let $L = \otimes(L_1, \ldots, L_k)$ and $M = \otimes(M_1, \ldots, M_k)$ with each L_i and M_i regular interval languages over Σ_i, respectively. Then it is easy to verify that $L \cap M = \otimes((L_1 \cap M_1), \ldots, (L_k \cap M_k))$. This is once again a regular direct product interval language since regular interval languages are closed under intersection.

To show that $\overline{L} = T\widetilde{\Sigma}^\infty - L$ is a regular product interval language, note that we can write $\overline{L}$ as

$$\overline{L} = \bigcup_{j \in \mathcal{P}} \otimes(W_1^j, \ldots, W_k^j)$$

where for each $i, j \in \mathcal{P}$,

$$W_i^j = \begin{cases} T\Sigma_i^\infty - L_i & \text{if } i = j \\ T\Sigma_i^\infty & \text{otherwise.} \end{cases}$$

Since each W_i^j is a regular interval language, we can conclude that $\overline{L}$ is a regular product interval language.

Thus, we have

Theorem 3.8 *The class of regular product interval languages over $\widetilde{\Sigma}$ is closed under the boolean operations of union, intersection, and complementation.* □

3.6 A logical characterisation

We present here the logic $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ which captures the class of regular product interval languages over $\widetilde{\Sigma}$.

The formulas in this logic will comprise boolean combinations of TMSO assertions (cf. Section 2.5) about individual components. The formulas in the logic will be interpreted over timed words over $\widetilde{\Sigma}$. An assertion about the actions of component i will be interpreted as a $\mathrm{TMSO}(\Sigma_i)$ sentence over the projection of the word to Σ_i.

The formulas of $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ are given by the following syntax:

$$\varphi ::= (\alpha)(i) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi)$$

where for each formula $(\alpha)(i)$ we require α to be a sentence in $\mathrm{TMSO}(\Sigma_i)$. The notation $(\alpha)(i)$ is meant to indicate that the sentence α comes from the logic $\mathrm{TMSO}(\Sigma_i)$. We introduce the operator $\wedge$ as a first class operator in the logic as a matter of convenience.

We note that the formulas in this logic are all sentences—i.e. they have no free variables.

A model for a $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ sentence is a timed word in $T\widetilde{\Sigma}^{\infty}$. For a word $\sigma \in T\widetilde{\Sigma}^{\infty}$ and a sentence $\varphi \in \mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$, the satisfaction relation $\sigma \models \varphi$ is given inductively as follows.

$$\begin{array}{lll}
\sigma \models (\alpha)(i) & \text{iff} & \sigma{\restriction}i \models \alpha \text{ (as a } \mathrm{TMSO}(\Sigma_i) \text{ formula)} \\
\sigma \models \neg\alpha & \text{iff} & \sigma \not\models \alpha \\
\sigma \models (\varphi \vee \varphi') & \text{iff} & \sigma \models \varphi \text{ or } \sigma \models \varphi' \\
\sigma \models (\varphi \wedge \varphi') & \text{iff} & \sigma \models \varphi \text{ and } \sigma \models \varphi'
\end{array}$$

For a sentence $\varphi \in \mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ we set $L(\varphi) = \{\sigma \in T\widetilde{\Sigma}^{\infty} \mid \sigma \models \varphi\}$.

Theorem 3.9 *Let $L \subseteq T\widetilde{\Sigma}^{\infty}$. Then L is a regular product interval language iff $L = L(\varphi)$ for some sentence $\varphi \in \mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$.*

To prove this theorem we will make use of the fact that regular interval languages are characterised by the logic TMSO.

We first observe a straightforward consequence of the semantics of $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$.

Proposition 3.10 *For each i in $\mathcal{P}$, let φ_i be a sentence in $\mathrm{TMSO}(\Sigma_i)$. Consider the $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ formula $((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k))$. Then*

$$L((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k)) = \otimes(L(\varphi_1), \ldots, L(\varphi_k)).$$

Proof Let $\sigma \in T\widetilde{\Sigma}^\infty$ and let $\sigma \models ((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k))$. Then, we know that $\sigma \models (\varphi_i)(i)$ for each $i \in \mathcal{P}$. Hence, for each $i \in \mathcal{P}$, $\sigma \restriction i \models \varphi_i$ and consequently, $\sigma \restriction i \in L(\varphi_i)$. Thus, $\sigma \in \otimes(L(\varphi_1), \ldots, L(\varphi_k))$.

Conversely, let $\sigma \in \otimes(L(\varphi_1), \ldots, L(\varphi_k))$. Then $\sigma \restriction i \in L(\varphi_i)$ for each $i \in \mathcal{P}$. That is, $\sigma \restriction i \models \varphi_i$ for each $i \in \mathcal{P}$. This means that $\sigma \models ((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k))$. □

Now given a regular direct product interval language $L = \otimes(L_1, \ldots, L_k)$ we know from Theorem 2.15 that there exist sentences $\varphi_i \in \mathrm{TMSO}(\Sigma_i)$ such that $L(\varphi_i) = L_i$. Using Proposition 3.10, we have $L((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k)) = L$. Thus regular direct product interval languages can be captured in our logic. Regular product interval languages are finite unions of regular direct product interval languages, and hence can be captured using the $\vee$ operator in our logic.

Conversely, given a sentence φ in $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$, we can write φ in disjunctive normal form by first driving in negation symbols (note that $\neg((\alpha)(i)) \equiv (\neg\alpha)(i)$) and then distributing $\wedge$ over $\vee$:

$$\varphi \equiv \bigvee_{j=1}^{m} (\bigwedge_{i=1}^{l_j} \gamma_i^j).$$

Here each γ_i^j is of the form $(\alpha_i^j)(p)$, with $\alpha_i^j \in \mathrm{TMSO}(\Sigma_p)$ for some $p \in \mathcal{P}$. Further, for $j = 1, \ldots, m$ and $i = 1, \ldots, k$, let X_i^j be the set of $\mathrm{TMSO}(\Sigma_i)$ sentences which occur in $\alpha_1^j, \ldots, \alpha_{l_j}^j$. Let β_i^j be the conjunction of formulas in X_i^j, with the convention that $\bigwedge \emptyset = \top$. It is easy to verify that

$$\varphi \equiv \bigvee_{j=1}^{m} (\bigwedge_{i=1}^{k} \beta_i^j).$$

From the semantics of $\vee$ it follows that

$$L(\varphi) = \bigcup_{j=1}^{m} (L(\bigwedge_{i=1}^{k} \beta_i^j)).$$

Once again, using Proposition 3.10 we have:

$$L(\varphi) = \bigcup_{j=1}^{m} (\otimes(L(\beta_1^j), \cdots, L(\beta_k^j))).$$

Since for each $i \in \mathcal{P}$, β_i^j is a $\mathrm{TMSO}(\Sigma_i)$ sentence, we know that $L(\beta_i^j)$ is a regular interval language over Σ_i. Thus, it follows that $L(\varphi)$ is a regular product interval language. This completes the proof of Theorem 3.9.

Chapter 4

Timed product-LTL

The Propositional Linear Time Temporal Logic (LTL) proposed by Pnueli [P77] is a specification logic which is well-suited for reasoning about safety and liveness properties of a wide class of reactive systems. Some of the advantages of LTL as a specification language are that it is a fairly natural logic to use, and it has an elementary decision procedure, in contrast to the logic MSO which has a non-elementary decision procedure. Further, LTL is known to be *expressively complete*: it is expressively equivalent to the first-order fragment of MSO [K68, GP+80].

In this chapter we attempt to lift these advantages to the timed setting. We formulate a timed version of LTL called timed product-LTL and denoted $\mathrm{TLTL}^{\otimes}$. An important motivation for studying this logic is that it is expressively complete, being expressively equivalent to the first-order fragment of the logic $\mathrm{TMSO}^{\otimes}$ introduced in Chapter 3.

$\mathrm{TLTL}^{\otimes}$ formulas will comprise boolean combinations of assertions over individual components in a timed logic called TLTL. The structure and semantics of $\mathrm{TLTL}^{\otimes}$ are based on the logic product-LTL proposed by Thiagarajan in the untimed setting [Thi95]. In [Thi95], the author's main motivation was to examine a subset of LTL in which one can make trace-consistent assertions about concurrent systems. Such assertions have the property that their truth is invariant across interleaved runs of a partially ordered computation, and hence constitute a class of properties which are amenable to partial-order based verification methods [V92]. The motivation in our case however, as mentioned earlier, is that $\mathrm{TLTL}^{\otimes}$ is a natural temporal logic to study in the framework of product interval automata. In fact, in our setting, product timed LTL turns out to be more expressive than timed LTL over a single global alphabet.

In the next section we introduce the logic TLTL and study its satisfiability problem. We will make use of this in Section 4.2 where we study the satisfiability and model checking problems for the logic TLTL$^{\otimes}$.

Finally, in Section 4.3 we prove the expressive completeness of TLTL$^{\otimes}$.

4.1 TLTL over a single component

We begin with our version of timed action-based LTL, denoted TLTL. Let Σ be an alphabet of actions. Then the formulas of TLTL(Σ) (parameterised by the alphabet Σ) are given by:

$$\varphi ::= \top \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \langle a, I\rangle\varphi \mid O\varphi \mid (\varphi U \varphi).$$

Here we require $a \in \Sigma$ and $I \in \mathcal{I}\mathbb{R}$.

The semantics of TLTL(Σ) is given as follows, with the models being timed words over Σ. In what follows, $\sigma \in T\Sigma^{\infty}$, and $\tau \in T\Sigma^{*}$ with $\tau \in$ *prf*(σ).

$$\begin{array}{lll}
\sigma, \tau \models \top & & \\
\sigma, \tau \models \neg\alpha & \text{iff} & \sigma, \tau \not\models \alpha \\
\sigma, \tau \models \alpha \vee \beta & \text{iff} & \sigma, \tau \models \alpha \text{ or } \sigma, \tau \models \beta \\
\sigma, \tau \models \langle a, I\rangle\alpha & \text{iff} & \exists t : \tau(a,t) \preceq \sigma \text{ and } t - \mathit{time}(\tau) \in I, \text{ and } \sigma, \tau(a,t) \models \alpha \\
\sigma, \tau \models O\alpha & \text{iff} & \exists \tau(a,t) \preceq \sigma \text{ with } \sigma, \tau(a,t) \models \alpha \\
\sigma, \tau \models \alpha U \beta & \text{iff} & \exists \delta \in \mathit{prf}(\sigma) \text{ with } \tau \preceq \delta, \text{ such that } \sigma, \delta \models \beta \text{ and} \\
 & & \forall \gamma : \tau \preceq \gamma \prec \delta,\ \sigma, \gamma \models \alpha.
\end{array}$$

We say $\sigma \models \varphi$ iff $\sigma, \epsilon \models \varphi$. Define $L(\varphi) = \{\sigma \in T\Sigma^{\infty} \mid \sigma \models \varphi\}$.

We mention here some of the standard abbreviations used in temporal logic. The formula $\Diamond\alpha$ (read as "future α" or "eventually α") is defined as $\Diamond\alpha = \top U \alpha$. $\Box\alpha$ ("globally α") is defined as $\Box\alpha = \neg\Diamond\neg\alpha$.

We now give a construction of an interval automaton which accepts the set of models of a given TLTL(Σ) formula. The construction follows the classical construction of [VWS83]. The manner in which the results are phrased is motivated by their application in the construction of a similar automaton for TLTL$^{\otimes}$.

Before we proceed, we would like to note here that a simpler route to follow would be to translate a given formula of TLTL(Σ) into an equivalent formula which mentions only intervals taken from a *proper* interval set. We can then use the classical construction to associate an appropriate interval automaton with the given formula. However, this method could lead to an exponential blow-up in the size of the translated formula. This blow-up is avoided in the method we adopt below.

Theorem 4.1 *Let X be a non-empty set of formulas of* TLTL(Σ). *Then we can construct a structure $\mathcal{A}_X = (Q, \longrightarrow, F, G)$ (an interval automaton without start states) such that for each non-empty subset Y of X there exists $(Q_{in})_Y^X \subseteq Q$ such that*

1. *$L(\bigwedge Y) = L(\mathcal{A}_X, (Q_{in})_Y^X)$ where by $(\mathcal{A}_X, (Q_{in})_Y^X)$ we mean the interval automaton $(Q, \longrightarrow, (Q_{in})_Y^X, F, G)$. Thus, by suitably choosing start states for $\mathcal{A}_X$ we can accept exactly the models of the conjunction of a subset of formulas in X.*
2. *The number of states in $\mathcal{A}_X$ is at most $2^{O(\sum_{\varphi \in X} |\varphi|)}$.*
3. *The largest bound mentioned in $\mathcal{A}_X$ is at most the largest bound mentioned in the formulas in X.*

Proof Let $\mathcal{I}$ be the set of intervals mentioned in the formulas in X. Let $\mathcal{I}'$ be a proper interval set covering $\mathcal{I}$. Using a method similar to the one outlined in section 2.4 we can construct $\mathcal{I}'$ such that the size of $\mathcal{I}'$ is at most $2 \cdot |\mathcal{I}|$, and the largest integer constant mentioned in $\mathcal{I}'$ is the largest integer constant mentioned in $\mathcal{I}$.

For a TLTL formula φ we will denote the subformula closure of φ by $sfc(\varphi)$. Formally, $sfc(\varphi)$ is defined inductively as follows:

- $sfc(\top) = \{\top\}$
- $sfc(\neg\alpha) = \{\neg\alpha\} \cup sfc(\alpha)$.
- $sfc((\alpha \vee \beta)) = \{(\alpha \vee \beta)\} \cup sfc(\alpha) \cup sfc(\beta)$.
- $sfc(\langle a, I\rangle\alpha) = \{\langle a, I\rangle\alpha\} \cup sfc(\alpha)$.
- $sfc(O\alpha) = \{O\alpha\} \cup sfc(\alpha)$.
- $sfc((\alpha U \beta)) = \{(\alpha U \beta)\} \cup sfc(\alpha) \cup sfc(\beta)$.

For a set of formulas X we write $sfc(X)$ to denote the set $\bigcup_{\alpha \in X} sfc(\alpha)$.

We define $CL(X)$, the Fisher-Ladner closure of a set of formulas X, to be

$$CL(X) = S \cup \{\neg\alpha \mid \alpha \in S\} \cup \{O(\alpha U \beta) \mid \alpha U \beta \in S\},$$

where $S = sfc(X)$.

It is easy to verify that $|CL(X)| \leq 3 \cdot \sum_{\alpha \in X} |\alpha|$.

An *atom* of X is a maximal "propositionally" consistent subset of $CL(X)$. Formally, a subset A of $CL(X)$ is an atom of X iff

1. If $\top \in CL(X)$ then $\top \in A$.
2. $\forall \alpha \in CL(X)$, $\neg\alpha \in A$ iff $\alpha \notin A$. (Here we identify $\neg\neg\alpha$ with α.)
3. $\forall (\alpha \vee \beta) \in CL(X)$, $(\alpha \vee \beta) \in A$ iff $\alpha \in A$ or $\beta \in A$.
4. $\forall (\alpha U \beta) \in CL(X)$, $(\alpha U \beta) \in A$ iff $\beta \in A$, or, both α, $O(\alpha U \beta) \in A$.

We can now define the automaton $\mathcal{A}_X$. Take Q to be the set of atoms of X. The transition relation $\longrightarrow$ is given by the following rule. We have $A \xrightarrow{(a,I')} B$ iff each of the following is satisfied:

1. $I' \in \mathcal{I}'$
2. if $\langle b, I \rangle \alpha \in A$ then
   (i) $b = a$
   (ii) $I \cap I' \neq \emptyset$
   (iii) $\alpha \in B$.
3. if $\langle a, I \rangle \alpha \in CL(X)$ with $I \cap I' \neq \emptyset$ and $\alpha \in B$, then $\langle a, I \rangle \alpha \in A$.
4. For all $O\alpha \in CL(X)$, $O\alpha \in A$ iff $\alpha \in B$.

The set of finitary final states F consists of atoms which have no "next-state" formulas—i.e. formulas of the form $\langle a, I \rangle \alpha$ or $O\alpha$.

For the infinitary final states we have a generalized Büchi condition $\mathcal{G} = \{G_1, \ldots, G_m\}$ where $m \geq 0$ is the number of until formulas in $CL(X)$. Let $\{\alpha_1 U \beta_1, \ldots, \alpha_m U \beta_m\}$ be the set of until formulas in $CL(X)$. Then for each $i \in \{1, \ldots, m\}$ we define $G_i = \{A \mid \alpha_i U \beta_i \notin A \text{ or } \beta_i \in A\}$.

Now let Y be a non-empty subset of X, $\varphi = \bigwedge Y$, and $(Q_{in})^X_Y = \{A \mid Y \subseteq A\}$. We will show that $L(\varphi) = L((\mathcal{A}_X, (Q_{in})^X_Y))$.

We first show that $L(\varphi) \subseteq L((\mathcal{A}_X, (Q_{in})^X_Y))$. Let σ be a model for φ. Let $\rho : \mathit{prf}(\sigma) \to Q$ be given by $\rho(\tau) = \{\alpha \in CL(X) \mid \sigma, \tau \models \alpha\}$. It is easy to verify that for each $\tau \in \mathit{prf}(\sigma)$, $\rho(\tau)$ is an atom. Further, for each $\tau(a,t) \in \mathit{prf}(\sigma)$ we have $\rho(\tau) \xrightarrow{(a,I')} \rho(\tau(a,t))$ where $I' \in \mathcal{I}'$ such that $t - \mathit{time}(\tau) \in I'$. This too can be readily verified, keeping in mind that we are working with a proper interval set $\mathcal{I}'$. Also, we can see that $\rho(\epsilon) \in (Q_{in})^X_Y$, since $\sigma, \epsilon \models \varphi$ and hence $\sigma, \epsilon \models \alpha$ for each $\alpha \in Y$, and hence $Y \subseteq \rho(\epsilon)$. We have verified so far that ρ is indeed a run of $(\mathcal{A}_X, (Q_{in})^X_Y)$ on σ. To argue that ρ is accepting, we first consider the case when σ is finite. Clearly $\rho(\sigma)$ cannot contain any formula of the form $\langle a, I \rangle \alpha$ or $O\alpha$. For the case when σ is infinite, we need to show that ρ meets each of the acceptance conditions G_i.

Suppose to the contrary that there exists an i for which G_i is not satisfied. Thus there exists a prefix τ of σ such that for all $\delta \in prf(\sigma)$ such that $\tau \preceq \delta$ we have $\sigma, \delta \models \alpha_i U \beta_i$ and $\sigma, \delta \not\models \beta_i$. From the semantics of U this is clearly not possible. Thus ρ is an accepting run of $(\mathcal{A}_X, (Q_{in})_Y^X)$ on σ, and hence $\sigma \in L((\mathcal{A}_X, (Q_{in})_Y^X))$.

Conversely, to argue that $L((\mathcal{A}_X, (Q_{in})_Y^X)) \subseteq L(\varphi)$, we first show the following:

Claim 4.2 *Let $\sigma \in T\Sigma^\infty$ and let ρ be an accepting run of $(\mathcal{A}_X, (Q_{in})_Y^X)$ on σ. Let $\alpha \in CL(X)$. Then for each $\tau \in prf(\sigma)$ we have $\sigma, \tau \models \alpha$ iff $\alpha \in \rho(\tau)$.*

We can prove the claim by induction on the structure of α. For $\alpha = \top$, we know $\top \in \rho(\tau)$ for all $\tau \in prf(\sigma)$.

For $\alpha = \neg\beta$, by the induction hypothesis, we know that $\sigma, \tau \models \neg\beta$ iff $\sigma, \tau \not\models \beta$ iff $\beta \notin \rho(\tau)$ iff $\neg\beta \in \rho(\tau)$.

For $\alpha = \langle a, I\rangle\beta$, let $\sigma, \tau \models \langle a, I\rangle\beta$. Then $\tau(a,t) \in prf(\sigma)$ with $t - time(\tau) \in I$, and $\sigma, \tau(a,t) \models \beta$. Thus, by induction hypothesis, $\beta \in \rho(\tau(a,t))$. From clause 3 of the definition of the transition relation $\longrightarrow$, we must have $\langle a, I\rangle\beta \in \rho(\tau)$. Conversely, let $\langle a, I\rangle\beta \in \rho(\tau)$. Let $\tau(b,t) \in prf(\sigma)$. Then, since ρ is a run on σ, there exists $I' \in \mathcal{I}'$ such that $\rho(\tau) \xrightarrow{(b,I')} \rho(\tau(a,t))$, and $t - time(\tau) \in I'$. But since $\langle a, I\rangle\beta \in \rho(\tau)$, by the definition of the transition relation $\longrightarrow$, it must be the case that $a = b$ and $I \cap I' \neq \emptyset$. Since $\mathcal{I}'$ is proper, it must be the case that $I' \subseteq I$, and hence $t - time(\tau) \in I'$ implies $t - time(\tau) \in I$. This tells us that $\sigma, \tau \models \langle a, I\rangle\beta$.

The arguments for the $\vee$, U and O connectives are similar to the classical case and are omitted here.

We have used a generalized Büchi condition above. We can go over to a standard Büchi condition as shown in Section 2.1 with an $O(m)$ blow-up in the number of states. Since $m \leq \sum_{\varphi \in X} |\varphi|$, the number of states is still $2^{O(\sum_{\varphi \in X} |\varphi|)}$ as claimed in the statement of the theorem. $\square$

Given $\varphi \in$ TLTL(Σ) we know that φ is satisfiable iff the language accepted by $\mathcal{A}_{\{\varphi\}}$ is non-empty. Using the results from section 2.3 we see that $\mathcal{A}_{\{\varphi\}}$ can be checked for emptiness in $2^{O(|\varphi|)}$ time. Thus we have:

Corollary 4.3 *The satisfiability of a formula φ in* TLTL *can be checked in $2^{O(|\varphi|)}$ time.*

4.2 Product TLTL

We now give the syntax and semantics of TLTL$^{\otimes}$. Let $\widetilde{\Sigma} = \{\Sigma_i\}$ be a distributed alphabet. The syntax of the logic TLTL$^{\otimes}(\widetilde{\Sigma})$ is given by:

$$\varphi ::= (\varphi)(i) \mid \neg\varphi \mid (\varphi \wedge \varphi') \mid (\varphi \vee \varphi')$$

where, as in section 3.6, we require of $(\varphi)(i)$ that $i \in \mathcal{P}$ and $\varphi \in \mathrm{TLTL}(\Sigma_i)$. Thus $(\varphi)(i)$ is an arbitrary i-type formula φ, tagged with the index i. Once again we introduce $\wedge$ as a first class operator in the logic for convenience in working out proofs. The expressiveness of the logic is unaffected even if we drop the $\wedge$ operator from the syntax.

Models for TLTL$^{\otimes}(\widetilde{\Sigma})$ formulas are timed words in $T\widetilde{\Sigma}^{\infty}$. Let $\varphi \in$ TLTL$^{\otimes}(\widetilde{\Sigma})$ and let $\sigma \in T\widetilde{\Sigma}^{\infty}$. The satisfaction relation $\sigma \models \varphi$ is defined inductively as follows:

$\sigma \models (\varphi)(i)$	iff	$\sigma\lceil i \models \varphi$ (as a TLTL(Σ_i) formula).
$\sigma \models \neg\varphi$	iff	$\sigma \not\models \varphi$
$\sigma \models (\varphi \wedge \varphi')$	iff	$\sigma \models \varphi$ and $\sigma \models \varphi'$.
$\sigma \models (\varphi \vee \varphi')$	iff	$\sigma \models \varphi$ or $\sigma \models \varphi'$.

Once again, we let $L(\varphi)$ denote the set $\{\sigma \in T\widetilde{\Sigma}^{\infty} \mid \sigma \models \varphi\}$.

Example 4.4 *The following formula over the distributed alphabet* $\widetilde{\Sigma} = (\{a, b\}, \{b\})$ *recognises the language* L *of Lemma 2.10.*

$$(\langle b, (0, \infty)\rangle\langle a, (0, \infty)\rangle\langle b, (0, \infty)\rangle\top)(1) \ \wedge \ (\langle b, (0, \infty)\rangle\langle b, (1, \infty)\rangle\top)(2).$$

$\square$

From the semantics of TLTL$^{\otimes}(\widetilde{\Sigma})$ the following lemma is immediate:

Lemma 4.5 *Let* $\varphi_1, \ldots, \varphi_k$ *be formulas in* TLTL$(\Sigma_1), \ldots,$ TLTL(Σ_k) *respectively. Then,*

$$L((\varphi_1)(1) \wedge \cdots \wedge (\varphi_k)(k)) = \otimes(L(\varphi_1), \ldots, L(\varphi_k)).$$

Next we show how we can associate with a given TLTL$^{\otimes}(\widetilde{\Sigma})$ formula a product interval automaton which recognizes exactly the models of the formula. Let $\varphi \in$ TLTL$^{\otimes}(\widetilde{\Sigma})$. As done for TMSO$^{\otimes}(\widetilde{\Sigma})$ in Theorem 3.9, we can write φ as

$$\varphi \equiv \bigvee_{j=1}^{m}(\bigwedge_{i=1}^{l_j} \alpha_i^j).$$

with each α_i^j a TLTL(Σ_p)-formula for some $p \in \mathcal{P}$. Further, for $j = 1, \ldots, m$ and $i = 1, \ldots, k$, let X_i^j be the set of i-type formulas which occur in $\alpha_1^j, \ldots, \alpha_{l_j}^j$. Let β_i^j be the conjunction of formulas in X_i^j, with the convention that $\bigwedge \emptyset = \top$. It follows that

$$\varphi \equiv \bigvee_{j=1}^{m} (\bigwedge_{i=1}^{k} \beta_i^j).$$

From the semantics of $\vee$ it follows that

$$L(\varphi) = \bigcup_{j=1}^{m} (L(\bigwedge_{i=1}^{k} \beta_i^j)).$$

From Lemma 4.5 we have:

$$L(\varphi) = \bigcup_{j=1}^{m} (\otimes(L(\beta_1^j), \cdots, L(\beta_k^j))).$$

Now using theorem 4.1 we can construct for each $i = 1, \ldots, k$ an interval automaton $\mathcal{A}_{X_i}$ (where $X_i = \bigcup_{j=1}^{m} X_i^j$), such that for each $j = 1, \ldots, m$ the interval automaton $(\mathcal{A}_{X_i}, (Q_{in})_{X_i^j}^{X_i})$ accepts $L(\beta_i^j)$. Using Lemma 3.7 it follows that

$$L(\varphi) = \bigcup_{j=1}^{m} L((\{\mathcal{A}_{X_i}\}_{i=1}^{k}, ((Q_{in})_{X_1^j}^{X_1} \times \cdots \times (Q_{in})_{X_k^j}^{X_k}))).$$

Now note that the product interval automata

$$(\{\mathcal{A}_{X_i}\}_{i=1}^{k}, ((Q_{in})_{X_1^j}^{X_1} \times \cdots \times (Q_{in})_{X_k^j}^{X_k})) \text{ for } j = 1, \ldots, m$$

are the same except for the start states. It then follows easily that

$$L(\varphi) = L(\mathcal{A}_\varphi)$$

where

$$\mathcal{A}_\varphi = (\{\mathcal{A}_{X_i}\}_{i=1}^{k}, \bigcup_{j=1}^{m} ((Q_{in})_{X_1^j}^{X_1} \times \cdots \times (Q_{in})_{X_k^j}^{X_k})).$$

Thus the satisfiability problem for TLTL$^\otimes$ can be solved as follows. Given a formula $\varphi \in$ TLTL$^\otimes(\tilde{\Sigma})$ we can generate the product interval automaton $\mathcal{A}_\varphi$ and then check the emptiness of $\mathcal{A}_\varphi$ as outlined in section 3.3.

To analyse the time complexity of checking satisfiability of φ, note that each component $\mathcal{A}_{X_i}$ of $\mathcal{A}_\varphi$ can be generated in time $2^{O(\sum_{\alpha \in X_i} |\alpha|)}$ using Theorem 4.1. Further, it is not difficult to see that $\prod_{i=1}^{k} 2^{O(\sum_{\alpha \in X_i} |\alpha|)} = 2^{O(|\varphi|)}$.

Thus, if Q_i and E_i are the set of states and edges respectively of the component $\mathcal{A}_{X_i}$, we have:

$$\prod_{i=1}^{k} |Q_i| = 2^{O(|\varphi|)} \text{ and } \prod_{i=1}^{k} |E_i| = 2^{O(|\varphi|)}.$$

The global initial states can be generated in time

$$2^{O(|\varphi|)} \cdot \prod_{i=1}^{k} |Q_i| = 2^{O(|\varphi|)}.$$

Finally note that $\prod_{i=1}^{k} c_i$ is also at most $2^{O(|\varphi|)}$, where c_i for $i \in \{1, \ldots, k\}$ is the largest interval bound appearing in an i-type formula. Now using these values in the time bounds obtained in section 3.3 for checking emptiness of product interval automata, we see that the satisfiability of φ can be decided in time

$$|\Sigma| \cdot 2^{O(|\varphi|)} \cdot 2^{O(k)} \cdot k!.$$

As regards the space complexity of satisfiability we have the following result:

Theorem 4.6 *Given $\widetilde{\Sigma}$ and a formula $\varphi \in$ TLTL$^{\otimes}(\widetilde{\Sigma})$, the problem of checking whether φ is satisfiable is* PSPACE*-complete.*

Proof The satisfiability problem for LTL is known to be PSPACE-complete [SC85]. PSPACE-hardness for TLTL$^{\otimes}$ follows easily by reducing the satisfiability problem for LTL to the one-component case of TLTL$^{\otimes}$. To show that the satisfiability check can be done in PSPACE, we argue that it can be done non-deterministic PSPACE. This will give us a PSPACE algorithm, since non-deterministic PSPACE is known to be equal to PSPACE.

1. The region graph $R(\mathcal{A}_\varphi)$ is an implicitly defined graph whose adjacency relation can be checked in space polynomial in $|\varphi| + |\widetilde{\Sigma}|$. Note that the description of each state-region pair is at most polynomial in the size of the input. In particular, each region can be described in $O(k^2) \cdot |\varphi|$ space, and time successors of a region can be generated in space linear in the description of the region (see [AD94]).

2. Reachability on an n-node graph can be done in non-deterministic $\log(n)$ space. The number of states in $R(\mathcal{A}_\varphi)$ can be seen to be

$$2^{O(|\varphi|)} \cdot 2^{O(k)} \cdot k!.$$

   Thus, the logarithm of the number of states in $R(\mathcal{A})$ is polynomial in $|\varphi|$ and $|\widetilde{\Sigma}|$, and hence we can check reachability in $R(\mathcal{A}_\varphi)$ non-deterministically in polynomial space in the size of φ and $\widetilde{\Sigma}$.

3. We can check emptiness of $R(\mathcal{A}_\varphi)$ using the algorithm for reachability. We guess a start state and a final state to begin with. We then check that the final state is reachable from the start state, and that the final state is reachable from itself.

$\square$

Next suppose we consider a real-time program Pr modeled by a product interval automaton $\mathcal{A}_{Pr}$, and a formula φ of $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$. Then Pr is said to meet the specification φ iff $L(\mathcal{A}_{Pr}) \subseteq L(\varphi)$. The model checking problem for $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$ is to determine whether Pr meets the specification φ.

Theorem 4.7 *The model checking problem for* $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$ *is* PSPACE-*complete.*

Proof PSPACE-hardness follows from the fact that the satisfiability problem for $\mathrm{TLTL}^{\otimes}$ can be reduced to the model checking problem. This is because the question of whether φ is satisfiable can be reduced to whether $\mathcal{A}_{univ} \not\models \neg\varphi$, where $\mathcal{A}_{univ}$ is a product interval automaton which recognizes the language $T\widetilde{\Sigma}^{\infty}$.

To see that the problem can be solved in PSPACE, we must check the emptiness of the intersection of $\mathcal{A}_{Pr}$ and $\mathcal{A}_\varphi$ in space polynomial in $|\mathcal{A}_{Pr}|+|\varphi|$. This is a similar argument to the one we have sketched for Theorem 4.6 above. $\square$

4.3 Expressive completeness of $\mathrm{TLTL}^{\otimes}$

In this chapter we show that $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$ is expressively equivalent to the first-order fragment of $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$. Once again, the proof of this result will factor through a well known result in the classical setting relating LTL and MSO [K68, GP+80].

Let FO[TMSO(A)] denote the first-order fragment of the logic TMSO(A). FO[TMSO(A)] is obtained from TMSO(A) by disallowing the use of quantification over set variables. Thus the syntax of the logic is given by

$$\varphi ::= (x \in X) \mid (x < y) \mid Q_a(x) \mid \Delta(x, I) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi.$$

The semantics is the same as that of TMSO(A).

The first-order fragment of $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$, denoted $\mathrm{FO}[\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})]$ is obtained by taking boolean combinations of the first-order fragment of $\mathrm{TMSO}(\Sigma_i)$ for each $i \in \mathcal{P}$. Thus the syntax of $\mathrm{FO}[\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})]$ is given by

$$\varphi ::= (\alpha)(i) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi)$$

where in each formula $(\alpha)(i)$, α is a sentence in $\mathrm{FO}[\mathrm{TMSO}(\Sigma_i)]$.

It will be useful to first recall the definition of LTL and the result concerning its expressive completeness.

Let A be an alphabet of actions. Then the formulas of the so-called "action-based" LTL, denoted $\mathrm{LTL}(A)$, are given by the syntax:

$$\varphi ::= \top \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \langle a \rangle \varphi \mid O\varphi \mid (\varphi U \varphi).$$

In the formula $\langle a \rangle \varphi$ we require $a \in A$.

The semantics of $\mathrm{LTL}(A)$ is given as follows, with the models being finite and infinite words over A. In what follows, $\sigma \in A^{\infty}$, and $\tau \in A^*$ with $\tau \in \mathit{prf}(\sigma)$.

$$\begin{array}{lll}
\sigma, \tau \models \top & & \\
\sigma, \tau \models \neg\alpha & \text{iff} & \sigma, \tau \not\models \alpha \\
\sigma, \tau \models \alpha \vee \beta & \text{iff} & \sigma, \tau \models \alpha \text{ or } \sigma, \tau \models \beta \\
\sigma, \tau \models \langle a \rangle \alpha & \text{iff} & \tau a \preceq \sigma \text{ and } \sigma, \tau a \models \alpha \\
\sigma, \tau \models O\alpha & \text{iff} & \exists a \in A: \ \tau a \preceq \sigma \text{ and } \sigma, \tau a \models \alpha \\
\sigma, \tau \models \alpha U \beta & \text{iff} & \exists \delta \in \mathit{prf}(\sigma) \text{ with } \tau \preceq \delta, \text{ such that } \sigma, \delta \models \beta \text{ and} \\
& & \forall \gamma : \tau \preceq \gamma \prec \delta, \ \sigma, \gamma \models \alpha.
\end{array}$$

We will say that $\sigma \models \varphi$ iff $\sigma, \epsilon \models \varphi$. Following the convention in this thesis, we set $L_{sym}(\varphi) = \{\sigma \in A^{\infty} \mid \sigma \models \varphi\}$.

Thus, looking back at the syntax of $\mathrm{TLTL}(\Sigma)$ formulas, we see that they are simply $\mathrm{LTL}(\Gamma)$ formulas for some interval alphabet Γ based on Σ. Of course, we must bear in mind that $\mathrm{TLTL}(\Sigma)$ formulas are interpreted over *timed* words over Σ.

Let $\mathrm{FO}[\mathrm{MSO}(A)]$ denote the first-order fragment of the logic $\mathrm{MSO}(A)$. As before, $\mathrm{FO}[\mathrm{MSO}(A)]$ is obtained from the logic $\mathrm{MSO}(A)$ defined in Section 2.5, by disallowing existential quantification over set variables. Then a well known result due to the work of Kamp, and Gabbay et. al, is:

Theorem 4.8 ([K68, GP+80]) $\mathrm{LTL}(A)$ *is expressively equivalent to* $\mathrm{FO}[\mathrm{MSO}(A)]$.

Returning to our aim of proving expressive completeness for $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$, we first show that $\mathrm{TLTL}(A)$, for an arbitrary alphabet A, corresponds to the first-order fragment of $\mathrm{TMSO}(A)$.

The following lemma will play a pivotal role in this.

Lemma 4.9 *1. Let Γ be a proper interval alphabet based on A. Let $\widehat{\varphi}$ be a formula in* LTL(Γ). *Thus $\widehat{\varphi}$ is also a formula in* TLTL(A)*; let us call it φ. Then*

$$L(\varphi) = tw(L_{sym}(\widehat{\varphi})).$$

2. Given a formula φ in TLTL(A)*, there exists a proper interval alphabet Γ based on A and a formula φ' in* TLTL(A) *such that $L(\varphi) = L(\varphi')$ and φ' is also a formula in* LTL(Γ).

Proof

1. The proof of this is very similar to our earlier arguments which make use of the properties of proper interval sets.

   Let $\sigma \in TA^{\infty}$ and let $\widehat{\sigma} \in \Gamma^{\infty}$ such that $\sigma \in tw(\widehat{\sigma})$. Further, let τ, $\widehat{\tau}$ be prefixes of σ and $\widehat{\sigma}$ respectively, such that $|\tau| = |\widehat{\tau}|$. Then we claim that $\sigma, \tau \models \varphi$ iff $\widehat{\sigma}, \widehat{\tau} \models \widehat{\varphi}$.

   To prove the claim, we proceed by induction on the structure of φ. For the base case of $\varphi = \top$, it is true that $\sigma, \tau \models \top$ iff $\widehat{\sigma}, \widehat{\tau} \models \top$.

   Now suppose $\varphi = \langle a, I\rangle\alpha$. Let $\sigma, \tau \models \langle a, I\rangle\alpha$. Then there exists $t \in \mathbb{R}^{>0}$ such that $\tau(a,t) \preceq \sigma$, and $t - time(\tau) \in I$, and $\sigma, \tau(a,t) \models \alpha$. Since $\sigma \in tw(\widehat{\sigma})$ we know that there exists $I' \in \Gamma_a$ such that $\widehat{\tau}(a, I') \preceq \widehat{\sigma}$ and $t - time(\tau) \in I'$. Since Γ is a proper interval alphabet, and both $I, I' \in \Gamma_a$, we must have $I = I'$. Further, by the induction hypothesis, we know that $\widehat{\sigma}, \widehat{\tau}(a, I) \models \alpha$. This proves that $\widehat{\sigma}, \widehat{\tau} \models \langle a, I\rangle\alpha$. Conversely, suppose $\widehat{\sigma}, \widehat{\tau} \models \langle a, I\rangle\alpha$. Then by the semantics of LTL(Γ) we must have $\widehat{\tau}(a, I) \preceq \widehat{\sigma}$ and $\widehat{\sigma}, \widehat{\tau}(a, I) \models \alpha$. Since $\sigma \in tw(\widehat{\sigma})$ we know that there exists $t \in \mathbb{R}^{>0}$ such that $\tau(a,t) \preceq \sigma$ and $t - time(\tau) \in I$. Also, by the induction hypothesis, we know that $\sigma, \tau(a,t) \models \alpha$. Thus, we have that $\sigma, \tau \models \langle a, I\rangle\alpha$.

   The remaining cases are easily taken care of and we omit the details here.

   Now using the above claim, we can prove part 1 of the theorem. Let $\sigma \in L(\varphi)$. So $\sigma, \epsilon \models \varphi$. Since Γ is a proper interval alphabet, by Lemma 2.11, there exists a $\widehat{\sigma} \in \Gamma^{\infty}$ such that $\sigma \in tw(\widehat{\sigma})$. By the claim, we now have that $\widehat{\sigma}, \epsilon \models \widehat{\varphi}$. Hence $\widehat{\sigma} \in L_{sym}(\widehat{\varphi})$ and $\sigma \in tw(L_{sym}(\widehat{\varphi}))$.

   Conversely, suppose $\sigma \in tw(\widehat{\sigma})$ and $\widehat{\sigma} \in L_{sym}(\widehat{\varphi})$. Thus $\widehat{\sigma}, \epsilon \models \widehat{\varphi}$. Using the claim again, we can conclude that $\sigma, \epsilon \models \varphi$. Thus $\sigma \in L(\varphi)$. This completes the proof of part 1 of the theorem.

2. Let $\varphi \in \mathrm{TLTL}(A)$ and let Γ be a proper interval alphabet based on A, such that for each $a \in A$, Γ_a covers $voc_a(\varphi)$. Then, to obtain φ' we simply replace formulas of the form $\langle a, I\rangle\alpha$ in φ by the formula

$$\bigvee_{b=a,\, I' \subseteq I,\, (b,I')\in\Gamma} \langle b, I'\rangle\alpha'$$

where, inductively, α' is obtained from α in a similar manner. It is easy to verify that $L(\varphi) = L(\varphi')$.

$\square$

Theorem 4.10 $\mathrm{TLTL}(A)$ *is expressively equivalent to the the first-order fragment of* $\mathrm{TMSO}(A)$.

Proof Let $\varphi_0 \in \mathrm{TLTL}(A)$. Then from Lemma 4.9 part 2, we know that there exists a $\mathrm{TLTL}(A)$ formula φ_1 and a proper interval alphabet Γ such that $L(\varphi_0) = L(\varphi_1)$ and φ_1 is also a $\mathrm{LTL}(\Gamma)$ formula. Let us refer to φ_1 as $\widehat{\varphi}_1$ when we consider it as a $\mathrm{LTL}(\Gamma)$ formula. Then, by part 1 of the same lemma, we have $L(\varphi_1) = tw(L_{sym}(\widehat{\varphi}_1))$. Now, by Theorem 4.8, we know that there exists a sentence $\widehat{\varphi}_2$ in $\mathrm{FO}[\mathrm{MSO}(\Gamma)]$ such that $L(\widehat{\varphi}_2) = L_{sym}(\widehat{\varphi}_1)$. Now consider the sentence $\varphi_3 = s\text{-}t(\widehat{\varphi}_2)$ (cf. Section 2.5). Since $\widehat{\varphi}_2$ uses only first order quantification, the translated formula φ_3 is a $\mathrm{FO}[\mathrm{TMSO}(A)]$ sentence. Further, Γ is a proper interval alphabet, and hence by Lemma 2.17 we know that $L(\varphi_3) = tw(L(\widehat{\varphi}_2))$. Thus φ_3 is the required $\mathrm{TMSO}(A)$ sentence with $L(\varphi_0) = L(\varphi_3)$.

Conversely, let φ_0 be a sentence in $\mathrm{FO}[\mathrm{TMSO}(A)]$. Then there exists a proper interval alphabet Γ based on A such that Γ_a covers $voc(\varphi_0)$ for each $a \in A$. Consider the $\mathrm{MSO}(\Gamma)$ sentence $\widehat{\varphi}_0 = t\text{-}s(\varphi_0)$ with respect to the interval alphabet Γ (cf. Section 2.5). By Lemma 2.16 $L(\varphi_0) = tw(L(\widehat{\varphi}_0))$. Further, by the nature of the translation, $\widehat{\varphi}_0$ is a sentence in $\mathrm{FO}[\mathrm{MSO}(\Gamma)]$. Now, again appealing to Theorem 4.8, we know that there exists an $\mathrm{LTL}(\Gamma)$ formula $\widehat{\varphi}_1$ such that $L_{sym}(\widehat{\varphi}_1) = L(\widehat{\varphi}_0)$. But by Lemma 4.9 part 1, we know that $\widehat{\varphi}_1$ is also an $\mathrm{LTL}(A)$ formula (call it φ_1) such that $L(\varphi_1) = tw(L_{sym}(\widehat{\varphi}_1))$. Thus φ_1 is the required formula in $\mathrm{TLTL}(A)$ such that $L(\varphi_1) = L(\varphi_0)$. $\square$

Using the above theorem, we can now prove:

Theorem 4.11 $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$ *is expressively equivalent to* $\mathrm{FO}[\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})]$.

Proof Let $\varphi \in \mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$. We define a sentence $l\text{-}m(\varphi)$ in $\mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$ such that $L(\varphi) = L(l\text{-}m(\varphi))$. The sentence $l\text{-}m(\varphi)$ is obtained by replacing each subformula of the form $(\alpha)(i)$ in φ by the sentence $(\alpha')(i)$ where α' is a sentence in $\mathrm{TMSO}(\Sigma_i)$ which is equivalent to the $\mathrm{TLTL}(\Sigma_i)$ formula α. Note that the existence of such an α' is guaranteed by Theorem 4.10. The fact that $L(\varphi) = L(l\text{-}m(\varphi))$ now follows easily by inductive argument on the structure of φ.

Conversely, for a sentence $\varphi \in \mathrm{TMSO}^{\otimes}(\widetilde{\Sigma})$, we define a formula $m\text{-}l(\varphi)$ in $\mathrm{TLTL}^{\otimes}(\widetilde{\Sigma})$ such that $L(\varphi) = L(m\text{-}l(\varphi))$. The formula $m\text{-}l(\varphi)$ is obtained by replacing each subformula of the form $(\alpha)(i)$ in φ by the formula $(\alpha')(i)$ where α' is a formula in $\mathrm{TLTL}(\Sigma_i)$ which is equivalent to the $\mathrm{TMSO}(\Sigma_i)$ formula α. Once again the existence such an α' is guaranteed by Theorem 4.10. The fact that $L(\varphi) = L(m\text{-}l(\varphi))$ now follows easily by inductive argument on the structure of φ. $\square$

Chapter 5

Modelling asynchronous circuits

The aim of this chapter is to show that product interval automata are expressive enough to model an important class of timed behaviours, namely that of asynchronous digital circuits. In [MP95] Maler and Pnueli model the timing behaviour of circuits using a network of timed automata that communicate via shared variables. Their model is based on the non-deterministic inertial delay model for gates [BS94]. With this model as our starting point, we describe the behaviour of a circuit using timed words (in contrast to the *signals* used in [MP95]). We then show that for a given circuit, we can define a product interval automaton which recognizes the language of timed words generated by the circuit.

We conclude the chapter with some examples of properties we can verify using this model. The properties will be expressed in the logic $\mathrm{TLTL}^{\otimes}$.

5.1 The non-deterministic delay model

A k-wire circuit (see Fig. 5.1) is modelled as a tuple

$$\chi = (X, F, D, b_0)$$

where $X = \{x_1, \ldots, x_k\}$ is a set of wires, $F = \{f_1, \ldots, f_k\}$ is a set of gates modeled as functions from $\{0,1\}^k$ to $\{0,1\}$, and $D = \{(l_1, u_1), \ldots, (l_k, u_k)\}$ is set of pairs of positive integers (u_i could be ∞ as a special case) with $l_i \leq u_i$ for each i. The pair (l_i, u_i) is meant to model the *delay* and the *latency* of the gate f_i. For simplicity the delay and the latency are assumed to be modeled by the same pair of values. Roughly speaking, a change in the input signal must hold for at least l_i units of time for it to be reflected

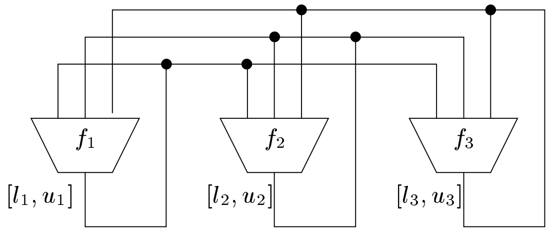

Figure 5.1: A 3-wire circuit

in the output signal of the gate, and if a change holds for u_i units of time, it *must* be reflected in the output of the gate. This models the latency of the gate. Further, the amount of delay in reflecting a change in the input signal must again lie in the interval $[l_i, u_i]$. These notions will be formalised below. b_0 here is an element of $\{0,1\}^k$ and will represent the initial values on the wires of the circuit.

Some simplifying assumptions have been made in this model. All gates are assumed to be of fan-in k and all wires are fed into a gate as input. Further, inputs to the circuit need to be modelled using the initial state b_0 of the circuit, or as NOT-gates within the circuit, with suitable delay values.

In [MP95] Maler and Pnueli used infinite "signals" to describe the behaviour of a circuit. A signal is a piece-wise constant, right-continuous map from $\mathbb{R}^{\geq 0}$ to the alphabet set, with countably many boundary points (or discontinuities). In our framework, we will use finite and infinite timed words over the alphabet $\Sigma = \{0,1\}^k$. A signal with finitely many discontinuities can be represented as a finite timed word (these are the so called "stable" signals). If they have infinitely many discontinuities, they can be represented as infinite timed words.

We now proceed to give the behaviour of a circuit in terms of timed words. We fix a k-wire circuit χ as above. Recall that we use the alphabet $\Sigma = \{0,1\}^k$.

We will need the following notions:

- Let $\tau \in T\Sigma^*$. We define $action(\tau)$ (w.r.t. χ) to be a if $\tau = \tau'(a,t)$ for some τ' and t, and we set $action(\epsilon) = b_0$. This is the analogue of the *time* function used earlier in the thesis.

- Let $s \in \{0,1\}^k$. The "hidden" value vector of s will be denoted by $h(s)$ and is given by $h(s)(i) = f_i(s)$. For each i, $h(s)(i)$ will represent the value computed by gate i, which may or may not be propagated to the output of the gate.

- Let $s \in \{0,1\}^k$. Then $excited(s)$ is a subset of $\{1,\dots,k\}$ given by $i \in excited(s)$ iff $s(i) \neq h(s)(i)$. We will say that s is i-excited if $i \in excited(s)$, and we will say that s is excited if $excited(s) \neq \emptyset$, Finally, we will say $\tau \in T\Sigma^*$ is excited (i-excited) if $action(\tau)$ is excited (i-excited).

- Let $s, s' \in \{0,1\}^k$. Then we define

$$switches(s,s') = \{i \in \{1,\dots,k\} \mid s'(i) \neq s(i)\}.$$

  These are the gates which have switched in going from s to s'.

- For $s, s' \in \{0,1\}^k$ we define $qtoe(s,s') \subseteq \{1,\dots,k\}$ given by $i \in qtoe(s,s')$ iff s' is i-excited and s is not i-excited. $qtoe$ is a mnemonic for "quiescent to excited."

  Similarly, $i \in etoq(s,s')$ iff s is i-excited and s' is not i-excited.

- Let $\tau \in T\Sigma^*$ and i be such that τ is i-excited. Let τ' be the smallest prefix of τ such that

  1. τ' is i-excited, and,
  2. For each τ'' such that $\tau' \preceq \tau'' \preceq \tau$ we have that τ'' is i-excited and further that $action(\tau'')(i) = action(\tau)(i)$.

  Then we define $etime_i(\tau) = time(\tau')$. Thus $etime_i(\tau)$ is defined when gate-i is excited in τ, and it is the time at which gate-i last became excited, without having switched in between.

We define when a finite timed word $\tau \in T\Sigma^*$ is *valid* (w.r.t. χ), inductively on the length of τ. The empty word ϵ is valid. A word of the form $\tau(a,t)$ is valid iff the following conditions hold. Let $action(\tau) = b$. Then

1. τ must be valid.
2. For each $i \in \{1,\dots,k\}$, if $i \in switches(b,a)$ then we must have $i \in excited(b)$. Thus, for a gate to switch, it must be in an excited state.
3. For each $i \in switches(b,a)$ we must have $(t - etime_i(\tau)) \in [l_i, u_i]$.
4. Suppose $i \in excited(b)$ and $i \notin switches(b,a)$. Then
   (a) if $i \notin excited(a)$ (i.e. gate i becomes quiescent) then we must have $(t - etime_i(\tau)) < u_i$.

(b) if $i \in \mathit{excited}(a)$ (i.e. gate i remains excited) then also we must have $(t - \mathit{etime}_i(\tau)) < u_i$.

We say an infinite timed word in $T\Sigma^\omega$ is valid iff every finite prefix of it is valid.

We now define $L(\chi)$, the behaviour of the circuit χ. For a finite word τ in $T\Sigma^*$ we have $\tau \in L(\chi)$ iff τ is valid w.r.t χ and τ is not excited. For an infinite word $\sigma \in T\Sigma^\omega$, we have $\sigma \in L(\chi)$ iff it is valid w.r.t. to χ.

5.2 Modelling a circuit as a PIA

We now show how we can model the circuit χ as a product interval automaton $\mathcal{A}_\chi$ over the distributed alphabet $\widetilde{\Sigma}'$ with $\Sigma' \subseteq \Sigma \times \Sigma$. This automaton will have the property that $L(\mathcal{A}_\chi) = L(\chi)$, modulo a (geometric) projection applied to the alphabet. Thus we will have $L(\chi) = \varsigma(L(\mathcal{A}_\chi))$ where ς is a substitution which maps actions of the form (s, s') to s'.

We define our distributed alphabet $\widetilde{\Sigma}'$ as follows. The set of actions Σ' is the set of all $(s, s') \in \Sigma \times \Sigma$ which satisfy:

1. $\mathit{switches}(s, s') \neq \emptyset$
2. $i \in \mathit{switches}(s, s')$ implies $i \in \mathit{excited}(s)$.

We will have $k + 1$ locations and the distribution of Σ' over the locations is given by:

$$\mathit{loc}((s, s')) = \{k + 1\} \cup \mathit{switches}(s, s') \cup \mathit{etoq}(s, s') \cup \mathit{qtoe}(s, s').$$

(Note that we have no independent actions here as $k + 1 \in \mathit{loc}(a)$ for all $a \in \Sigma'$. Hence $T\widetilde{\Sigma}'^\infty = T\Sigma'^\infty$.)

The components $\mathcal{A}_i$, for $i = 1, \ldots, k$, will correspond to the ith gate. The $(k + 1)$-th component, $\mathcal{A}_{k+1}$, will be free of any timing constraints and will keep track of the current vector of output values of the circuit.

We define $\mathcal{A}_\chi = (\{\mathcal{A}_i\}_{i=1}^{k+1}, Q_{in})$ with each $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, G_i)$ given as follows:

- For $i \in \{1, \ldots, k\}$ we have:
  - $Q_i = \{e, q\}$
  - $\longrightarrow_i$ is given by:
    - $e \xrightarrow{((s,s'),[l_i,u_i])}_i e$ provided $i \in \mathit{switches}(s, s')$ and $i \in \mathit{excited}(s')$.

  * $e \overset{((s,s'),[l_i,u_i])}{\longrightarrow_i} q$ provided $i \in \mathit{switches}(s,s')$ and $i \notin \mathit{excited}(s')$.
  * $e \overset{((s,s'),(0,u_i))}{\longrightarrow_i} q$ provided $i \notin \mathit{switches}(s,s')$ and $i \in \mathit{etoq}(s,s')$.
  * $q \overset{((s,s'),(0,\infty))}{\longrightarrow_i} e$ provided $i \in \mathit{qtoe}(s,s')$.
- $F_i = \{q\}$
- $G_i = \{e,q\}$

- For $\mathcal{A}_{k+1}$ we have:
  - $Q_{k+1} = \{0,1\}^k$
  - $\longrightarrow_{k+1}$ is given by:

$$s \overset{((t,t'),I)}{\longrightarrow_{k+1}} s' \text{ iff } s = t,\ s' = t',\text{ and } I = (0,\infty).$$

  - $F_{k+1} = \{s \in Q_{k+1} \mid \mathit{excited}(s) = \emptyset\}$.
  - $G_{k+1} = Q_{k+1}$.

Finally, the set of initial states Q_{in} will be a singleton $\{q_{in}\}$ with $q_{in} = \{(d_1,\ldots,d_k,b_0)\}$ where for each $i \in \{1,\ldots,k\}$ $d_i = e$, if b_0 is i-excited, and q otherwise.

5.3 Proof of correctness

In this section we prove the claim that $\varsigma(L(\mathcal{A}_\chi)) = L(\chi)$. It will be more convenient for us to follow a slightly different route and re-define the behaviour of χ in terms of timed words over Σ'. We do this by mapping each Σ-timed word σ to a Σ'-timed word $\varsigma'(\sigma)$ in a natural way. We will then show that $\varsigma'(L(\chi)) = L(\mathcal{A}_\chi)$.

Let $\sigma \in T\Sigma^\infty$. Then we define $\varsigma(\sigma)$ by defining for each prefix τ of σ a corresponding prefix $\widehat{\tau}$ of $\varsigma(\sigma)$. For $\tau = \epsilon$ we define $\widehat{\tau} = \epsilon$. For a nonempty prefix $\tau = \tau'(a,t)$ we define $\widehat{\tau} = \widehat{\tau'}((b,a),t)$ where $b = \mathit{action}(\tau)$.

We extend the notions introduced earlier (like validity) in the natural way for these Σ'-timed words as well. In what follows, we will use $L(\chi)$ to mean $\varsigma'(L(\chi))$,

Theorem 5.1 $L(\chi) = L(\mathcal{A}_\chi)$

Proof We first show that $L(\chi)$ is contained in $L(\mathcal{A}_\chi)$.

First note that if σ is a valid Σ' timed word, and τ is a prefix of σ such that τ is i-excited, then $\mathit{time}_i(\tau) = \mathit{etime}_i(\tau)$. This is easy to see by induction on the length of τ.

Now let $\sigma \in L(\chi)$. Consider the map $\rho : \mathit{prf}(\sigma) \to Q$, where $Q = \Pi_{i=1}^{k+1} Q_i$, given by:

- $\rho(\epsilon) = q_{in}$
- and for each prefix $\tau(a,t)$ of σ, with $a = (s,s')$, $\rho(\tau(a,t))$ is given by $\rho(\tau(a,t))(k+1) = s'$, and for $i \in \{1,\ldots,k\}$, $\rho(\tau(a,t))(i) = e$ if $i \in \mathit{excited}(s')$ and q otherwise.

We claim that ρ is a run of $\mathcal{A}_\chi$ on σ. Let $\tau((s,s'),t)$ be a prefix of σ, and let $i \in \mathit{loc}((s,s'))$. Then we can show that there exists an interval I such that

- $\rho(\tau)(i) \xrightarrow{((s,s'),I)}_i \rho(\tau((s,s'),t))(i)$
- $t - \mathit{time}_i(\tau) \in I$.

For example, if $i \in \mathit{switches}(s,s')$ and $i \notin \mathit{excited}(s')$ then $\rho(\tau)(i) = e$ and $\rho(\tau((s,s'),t))(i) = q$ and we know that $e \xrightarrow{((s,s'),[l_i,u_i])}_i q$. Further, using the preceding observation, we know that $t - \mathit{time}_i(\tau) = t - \mathit{etime}_i(\tau)$ which in turn lies in the interval $[l_i, u_i]$ by the validity of σ. The other cases can be argued similarly.

Further, it is easy to verify that for $i \notin \mathit{loc}((s,s'))$ we have $\rho(\tau)(i) = \rho(\tau((s,s'),t))(i)$. Thus ρ is a run on σ.

We now argue that ρ is accepting. If σ is finite, then by definition of $L(\chi)$, σ is not i-excited for any i, and hence $\rho(\sigma)(i) = q$ for all $i \in \{1,\ldots,k\}$, and ρ is accepting. If σ is infinite, then assume that ρ is not accepting. Then, by the nature of the acceptance conditions, there must exist an $i \in \{1,\ldots,k\}$ such that $\mathcal{A}_i$ participates only finitely often in σ, say up to τ, and $\rho(\tau)(i) = e$. By the definition of ρ this means that τ'' is i-excited for all prefixes τ'' of σ such that $\tau \preceq \tau''$. But σ is progressive, and hence there exists a prefix τ' of σ such that $\mathit{time}(\tau') - \mathit{time}(\tau) > u_i$, which in turn implies $\mathit{time}(\tau') - \mathit{etime}_i(\tau') > u_i$. This contradicts the validity of σ. Thus ρ must be an accepting run.

Conversely, to argue that $L(\mathcal{A}_\chi) \subseteq L(\chi)$, we first argue that if ρ is a run of $\mathcal{A}_\chi$ on a word σ, then σ is *weakly* valid. We say that a word $\sigma \in T\Sigma^*$ is weakly valid if it satisfies all the clauses of the definition of validity, except possibly clause 4b. As before, an infinite word will be termed weakly valid iff every finite prefix of it is weakly valid.

Let $\sigma \in T\Sigma'^\infty$ and let ρ be a run of $\mathcal{A}_\chi$ on σ. We show that for each τ prefix of σ

- if τ is i-excited, then $\mathit{etime}_i(\tau) = \mathit{time}_i(\tau)$.
- For $i \in \{1,\ldots,k\}$ $\rho(\tau)(i) = e$ iff τ is i-excited.

We prove the claim by induction on $|\tau|$. The claim clearly holds for $\tau = \epsilon$. Now let $\tau = \tau'(a,t)$ be a prefix of σ with $a = (s,s')$. The following cases arise:

- τ' is i-excited, $i \in \mathit{switches}(s,s')$, and $i \in \mathit{excited}(s')$. By the induction hypothesis we know that $\rho(\tau)(i) = e$. Further, $i \in \mathit{loc}((s,s'))$ and the only applicable i transition is $e \xrightarrow{((s,s'),[l_i,u_i])}_i e$. Thus $\rho(\tau(a,t))(i) = e$. Further, $\mathit{etime}_i(\tau(a,t)) = t = \mathit{time}_i(\tau(a,t))$.

- τ' is i-excited, $i \in \mathit{switches}(s,s')$, and $i \notin \mathit{excited}(s')$. Once again, by the induction hypothesis we know that $\rho(\tau')(i) = e$. Further, $i \in \mathit{loc}((s,s'))$ and the only applicable i transition is $e \xrightarrow{((s,s'),[l_i,u_i])}_i q$. Thus $\rho(\tau'(a,t))(i) = q$. Since τ is not i-excited, we are done.

- τ' is i-excited and $i \notin \mathit{switches}(s,s')$. By the induction hypothesis we know that $\rho(\tau')(i) = e$ and since $i \notin \mathit{loc}(a)$ we have $\rho(\tau)(i) = e$. Further, by the induction hypothesis, $\mathit{etime}_i(\tau') = \mathit{time}_i(\tau')$, and since $\mathit{etime}_i(\tau'(a,t)) = \mathit{etime}_i(\tau')$, we have $\mathit{etime}_i(\tau'(a,t)) = \mathit{time}_i(\tau'(a,t))$.

- τ' is not i-excited. then $i \in \mathit{qtoe}(s,s')$ and $i \in \mathit{loc}(a)$. Further, by the induction hypothesis, $\rho(\tau')(i) = q$. The only applicable i-transition is $q \xrightarrow{((s,s'),(0,\infty))}_i e$. Thus $\rho(\tau'(a,t))(i) = e$. Further, $\mathit{etime}_i(\tau'(a,t)) = t = \mathit{time}_i(\tau'(a,t))$.

Using the claim proved above, it is easy to verify that such a σ is weakly valid.

We now argue that if $\mathcal{A}_\chi$ has an accepting run ρ on σ, then σ must be valid. Suppose to the contrary that σ is not valid. Thus, using the first part above, σ is weakly-valid, but not valid. It follows that there must exist an $i \in \{1,\ldots,k\}$ and a prefix τ of σ, such that τ is i-excited and $i \notin \mathit{loc}(a)$ for all prefixes $\tau'(a,t)$ of σ such that $\tau \preceq \tau'$. Thus $\sigma{\restriction}i$ is finite, and equals $\tau{\restriction}i$. Since $\rho(\tau)(i) = e$, $\rho(\tau)(i) \notin F_i$, and ρ cannot be accepting. Thus σ must be valid.

Finally, if σ is finite then, since ρ is accepting, $\rho(\sigma)(i) = q$. Using the claim above, we have that σ is not i-excited for any i.

This completes the proof that $\sigma \in L(\chi)$. $\square$

5.4 Properties expressed in TLTL$^\otimes$

We list below some properties of circuits that we can specify in our logic TLTL$^\otimes\Sigma'$. We can then check whether a given circuit satisfies these properties using the method for model-checking TLTL$^{\otimes}$ formulas given in Chapter 4.

- The formula φ below specifies that every possible behaviour of a circuit is eventually stable.

$$\varphi = (\bigvee_{excited(s')=\emptyset} \Diamond\langle(s,s'),(0,\infty)\rangle\top)\ (k+1).$$

- This formula asserts that every behaviour of a circuit is unstable:

$$(\Box \bigvee_{a\in\Sigma'} \langle a,(0,\infty)\rangle\top)\ (k+1).$$

- For a circuit in which the initial value b_0 is such that $b_0(i) = 0$ the formula

$$(\neg\Diamond(\bigvee_{i\in switches(s,s')} \langle(s,s'),(0,\infty)\rangle\top))\ (i)$$

specifies that gate i is stuck at 0. If $b_0(i) = 1$ the same property is specified by the formula

$$(\Diamond \bigvee_{s'(i)=0} \langle(s,s'),(0,\infty)\rangle(\neg\Diamond \bigvee_{i\in switches(s,s')} \langle(s,s'),(0,\infty)\rangle\top))\ (i).$$

- We can specify that gate i always switches within d time units of becoming excited. This is specified by the formula

$$(\Box(\varphi_1 \Rightarrow \varphi_2))(i)$$

if $i \notin excited(b_0)$, and by the formula

$$(\Box(\varphi_1 \Rightarrow \varphi_2))(i) \wedge (\varphi_3)(i)$$

if $i \in excited(b_0)$, where

$$\begin{array}{lcl} \varphi_1 & = & \bigvee_{i\in excited(s')}\langle(s,s'),(0,\infty)\rangle\top \\ \varphi_2 & = & \bigvee_{a\in\Sigma'}\langle a,(0,\infty)\rangle(\bigvee_{i\in switches(t,t')}\langle(t,t'),(0,d]\rangle\top)\ (i) \\ \varphi_3 & = & \bigvee_{i\in switches(s,s')}\langle(s,s'),(0,d]\rangle\top. \end{array}$$

Chapter 6

Distributed interval automata

Our goal here is to enlarge the class of product interval languages introduced in Chapter 3. The synchronisation of actions in product interval automata is "locally" specified. It is possible to specify the synchronisations in a global manner as done in the asynchronous automata of Zielonka [Z87]. Here we will prefer to use the term *distributed automata* instead of asynchronous automata.

It is well known in classical trace theory that synchronized products of automata are strictly less expressive than distributed automata. Expectedly, we gain in expressive power by considering the distributed version of product interval automata. We call these automata *distributed interval automata*.

As in the case of product interval automata, distributed interval automata also admit a logical characterisation and enjoy the desired closure properties.

The proof of these results will make use of several well-known results in classical trace theory. We begin by introducing these results along with the necessary background material.

6.1 Preliminaries

In this chapter and the next we will confine ourselves to infinite behaviours. This is purely for the sake of convenience, as we can then avoid dealing with multiple acceptance conditions in our automata.

For the rest of this section, we fix a $\mathcal{P}$-distributed alphabet $\widetilde{\Delta}$, with $\mathcal{P} = \{1, \ldots, k\}$.

Recall that a Mazurkiewicz trace alphabet is a pair (A, I) where A is a finite alphabet and I is an irreflexive and symmetric relation on A. I is called an independence relation on A. A distributed alphabet $\widetilde{\Delta}$ induces in a natural manner a Mazurkiewicz trace alphabet $(\Delta, I_{\widetilde{\Delta}})$ where $I_{\widetilde{\Delta}} \subseteq \Delta \times \Delta$

is the independence relation induced by $\widetilde{\Delta}$, given by

$$a \, I_{\widetilde{\Delta}} \, b \text{ iff } loc(a) \cap loc(b) = \emptyset.$$

We will use $D_{\widetilde{\Delta}}$ to denote the dependence relation $(\Delta \times \Delta) - I_{\widetilde{\Delta}}$.

$\widetilde{\Delta}$ induces an equivalence relation $\approx_{\widetilde{\Delta}}$ on Δ^ω given by $\sigma \approx_{\widetilde{\Delta}} \sigma'$ iff for each $i \in \mathcal{P}$, $\sigma \restriction i = \sigma' \restriction i$. As usual, $\sigma \restriction i$ is obtained by erasing from σ all occurrences of letters a such that $a \notin \Delta_i$.

We say $L \subseteq \Delta^\omega$ is $\approx_{\widetilde{\Delta}}$-*consistent* iff for all $\sigma, \sigma' \in \Delta^\omega$, whenever $\sigma \approx_{\widetilde{\Delta}} \sigma'$ then $\sigma \in L$ iff $\sigma' \in L$. $L \subseteq \Delta^\omega$ is then an *ω-regular consistent language* over $\widetilde{\Delta}$ iff L is an ω-regular subset of Δ^ω and L is $\approx_{\widetilde{\Delta}}$-consistent. (We say a subset L of Δ^ω is ω-regular iff it is accepted by a Büchi automaton over Δ.) We denote the class of ω-regular consistent languages over a distributed alphabet $\widetilde{\Delta}$ by $\mathrm{RCL}(\widetilde{\Delta})$.

Next we introduce distributed automata. We will need some notation to begin with. Let $\widetilde{\Delta}$ be a $\mathcal{P}$-distributed alphabet. For convenience, we will denote a $\mathcal{P}$-family of sets $\{X_i\}_{i \in \mathcal{P}}$ by $\{X_i\}$. Let $\{S_i\}$ be a $\mathcal{P}$-family of local state sets. Suppose P is a non-empty subset of $\mathcal{P}$. Then a P-state (relative to the family $\{S_i\}$) is a map $s : P \to \bigcup_{i \in P} S_i$ such that $s(i) \in S_i$ for each $i \in P$. We let S_P stand for the set of P-states. We will refer to $\mathcal{P}$-states as global states. An a-state is just a $loc(a)$-state. We let S_a denote the set of a-states.

Definition 6.1 *A distributed (Büchi) automaton over $\widetilde{\Delta}$ is structure $\mathcal{A} = (\{S_i\}, \{\longrightarrow_a\}_{a \in \Delta}, S_{in}, \{F_i\}, \{G_i\})$ which satisfies:*

- *Each S_i is a finite set of i-local states.*
- *$\longrightarrow_a \subseteq S_a \times S_a$ for each a.*
- *$S_{in} \subseteq S_{\mathcal{P}}$ is a set of global initial states.*
- *F_i, G_i are subsets of S_i for each i.*

It will be convenient to first define a global transition relation $\longrightarrow_{\mathcal{A}}$ for $\mathcal{A}$. $\longrightarrow_{\mathcal{A}}$ is a subset of $S_{\mathcal{P}} \times \Delta \times S_{\mathcal{P}}$ given by: $s \overset{a}{\longrightarrow}_{\mathcal{A}} s'$ iff there exists an a-transition $s_a \longrightarrow_a s'_a$ such that $s(i) = s_a(i)$ and $s'(i) = s'_a(i)$ for every $i \in loc(a)$. Further, $s(i) = s'(i)$ for every $i \notin loc(a)$.

A run of $\mathcal{A}$ on a word $\widehat{\sigma} \in \Delta^\omega$ is a map $\rho : prf(\widehat{\sigma}) \to S_{\mathcal{P}}$ such that:

- $\rho(\epsilon) \in S_{in}$
- for each prefix $\widehat{\tau}a$ of $\widehat{\sigma}$ we have $\rho(\widehat{\tau}) \overset{a}{\longrightarrow}_{\mathcal{A}} \rho(\widehat{\tau}a)$.

Further, ρ is said to be accepting iff for each $i \in \mathcal{P}$ either

(i) $\widehat{\sigma} \restriction i$ is finite and $\rho(\widehat{\tau})(i) \in F_i$ for some prefix $\widehat{\tau}$ of $\widehat{\sigma}$ such that $\widehat{\tau} \restriction i = \widehat{\sigma} \restriction i$, or

(ii) $\widehat{\sigma} \restriction i$ is infinite and $\rho(\widehat{\tau})(i) \in G_i$ for infinitely many $\widehat{\tau} \in prf(\widehat{\sigma})$.

As usual we will say a word $\widehat{\sigma}$ is accepted by $\mathcal{A}$ iff there exists an accepting run of $\mathcal{A}$ on $\widehat{\sigma}$. We set $L_{sym}(\mathcal{A})$ to be the subset Δ^ω accepted by $\mathcal{A}$.

In the setting of finite words, Zielonka [Z87] has shown that distributed automata accept precisely the class of regular consistent languages. Gastin and Petit [GP92] later extended this result to languages of infinite words as well.

Thus the result due to Gastin and Petit [GP92] specialised to our setting is:

Theorem 6.2 ([GP92]) *Let $L \subseteq \Delta^\omega$. Then L is an ω-regular consistent language over $\widetilde{\Delta}$ iff L is accepted by a distributed automaton over $\widetilde{\Delta}$.*

In fact, Gastin and Petit use a slightly different acceptance condition for distributed automata. Their condition can however be seen to be equivalent to the one we use here. The reader is referred to [MT96] for a discussion on this.

Next, we introduce *traces* which are compact ways of representing $\approx_{\widetilde{\Delta}}$ equivalence classes of words.

Definition 6.3 *Let $\widetilde{\Delta}$ be a distributed alphabet. An (infinite)* trace *T over $\widetilde{\Delta}$ is a triple $(E, \leq, \lambda)$ where*

- *$(E, \leq)$ is a countably infinite, finitary partial order. By finitary we mean that for every $e \in E$ the set $\{f \in E \mid f < e\}$ is finite.*
- *$\lambda : E \to \Delta$ is a labelling function which satisfies:*
  - *$e \lessdot e' \Rightarrow \lambda(e) \, D_{\widetilde{\Delta}} \, \lambda(e')$. (We say $e \lessdot e'$ iff $e < e'$ and for all $f \in E$, $e \leq f \leq e'$ implies either $f = e$ or $f = e'$.)*
  - *$\lambda(e) \, D_{\widetilde{\Delta}} \, \lambda(e') \Rightarrow e \leq e'$ or $e' \leq e$.*

Let $\mathrm{Tr}(\widetilde{\Delta})$ denote the set of traces over $\widetilde{\Delta}$. We wish to point out here that for convenience we will not be distinguishing between isomorphic traces. Thus we will say $(E, \leq, \lambda) = (E', \leq', \lambda')$ if $(E, \leq)$ is isomorphic to $(E', \leq')$ via a map $\phi : E \to E'$ and $\lambda(e) = \lambda'(\phi(e))$ for all $e \in E$.

Traces over $\widetilde{\Delta}$ and $\approx_{\widetilde{\Delta}}$ equivalence classes represent each other via the maps *w-t* and *t-w* which we now define.

The map ("words to traces") $w\text{-}t : \Delta^\omega \to \mathrm{Tr}(\widetilde{\Delta})$ associates a trace with a word $\sigma \in \Delta^\omega$, as follows:

$$w\text{-}t(\sigma) = (E, \leq, \lambda)$$

where

- $E = prf(\sigma) - \{\epsilon\}$.
- $\leq$ is the least partial order contained in $E \times E$ satisfying: if $\tau a \preceq \tau' b$ with $a \, D_{\widetilde{\Delta}} \, b$, then $\tau a \leq \tau' b$.
- $\lambda(\tau a) = a$.

The map $t\text{-}w : \mathrm{Tr}(\widetilde{\Delta}) \to 2^{(\Delta^\omega)}$ is defined as:

$$t\text{-}w((E, \leq, \lambda)) = \{\lambda(\delta) \mid \delta \text{ is a linearization of } (E, \leq)\}.$$

(We denote the canonical extension of λ to words in Δ^ω by λ itself.)

We extend the maps *w-t* and *t-w* to act naturally on subsets of Δ^ω and $\mathrm{Tr}(\widetilde{\Delta})$ respectively. In particular, for $L \subseteq \mathrm{Tr}(\widetilde{\Delta})$ we have

$$t\text{-}w(L) = \bigcup_{T \in L} t\text{-}w(T).$$

Let $\mathrm{RTrL}(\widetilde{\Delta})$ denote the class of trace languages which are images, under the map *w-t*, of ω-regular consistent languages over $\widetilde{\Delta}$. That is,

$$\mathrm{RTrL}(\widetilde{\Delta}) = \{w\text{-}t(L) \mid L \in \mathrm{RCL}(\widetilde{\Delta})\}.$$

Then the classes $\mathrm{RCL}(\widetilde{\Delta})$ and $\mathrm{RTrL}(\widetilde{\Delta})$ represent each other via the maps *w-t* and *t-w* in the following sense

Proposition 6.4 (see [Thi95]) *1. For any L in $\mathrm{RCL}(\widetilde{\Delta})$, $t\text{-}w(w\text{-}t(L)) = L$, and*

2. For any L in $\mathrm{RTrL}(\widetilde{\Delta})$, $w\text{-}t(t\text{-}w(L)) = L$.

We now recall some results on the logical definability of ω-regular consistent languages. In [Tho89] Thomas shows that regular consistent languages (over finite words) can be characterised by a monadic second order logic in which the '$\leq$' is interpreted as the partial ordering of action occurrences along a word (and not the *prefix* ordering that we have used till now). Ebinger and Muscholl [EM96] later extended this result to the setting of infinite words.

The formulas of $\mathrm{MSOTr}(\widetilde{\Delta})$, the monadic second order logic of traces over $\widetilde{\Delta}$, are given by the following syntax:

$$\varphi ::= (x \leq y) \mid Q_a(x) \mid (x \in X) \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x \varphi \mid \exists X \varphi.$$

Here a is required to be in Δ, and as usual, x and X are individual and set variables respectively.

The formulas of $\mathrm{MSOTr}(\widetilde{\Delta})$ are interpreted over traces in $\mathrm{Tr}(\widetilde{\Delta})$. Given a trace $T = (E, \leq, \lambda)$ over $\widetilde{\Delta}$, and an interpretation $\mathbb{I}$ which assigns elements of E to the first-order variables $\{x, y, \ldots\}$ and subsets of E to the second-order variables $\{X, Y, \ldots\}$, the atomic formulas are interpreted as follows:

$$\begin{array}{lll} T \models_{\mathbb{I}} (x \leq y) & \text{iff} & \mathbb{I}(x) \leq \mathbb{I}(y) \text{ in the partial order } (E, \leq) \\ T \models_{\mathbb{I}} Q_a(x) & \text{iff} & \lambda(\mathbb{I}(x)) = a \\ T \models_{\mathbb{I}} (x \in X) & \text{iff} & \mathbb{I}(x) \in \mathbb{I}(X) \end{array}$$

The logical connectives $\neg$, $\vee$, and $\exists$ (for both first-order and second-order variables) are interpreted in the usual manner.

For a sentence $\varphi \in \mathrm{MSOTr}(\widetilde{\Delta})$, let $L(\varphi)$ denote the set of traces which satisfy φ. That is

$$L(\varphi) = \{T \in \mathrm{Tr}(\widetilde{\Delta}) \mid T \models \varphi\}.$$

Theorem 6.5 ([EM96]) *$L \in \mathrm{RTrL}(\widetilde{\Delta})$ iff $L = L(\varphi)$ for some sentence $\varphi \in \mathrm{MSOTr}(\widetilde{\Delta})$.*

6.2 Distributed interval automata

Once again we fix a $\mathcal{P}$-distributed alphabet $\widetilde{\Sigma} = \{\Sigma_p\}_{p \in \mathcal{P}}$ with $\mathcal{P} = \{1, 2, \ldots, k\}$. Let $a \in \Sigma$.

An a-event of $\widetilde{\Sigma}$ is a pair (a, J) with $J : loc(a) \to \mathcal{I}\mathbb{R}$. The set of events of $\widetilde{\Sigma}$, denoted $E(\widetilde{\Sigma})$, is defined to be:

$$E(\widetilde{\Sigma}) = \{(a, J) \mid a \in \Sigma, \text{ and } (a, J) \text{ is an } a\text{-event}\}.$$

A *distributed interval alphabet* based on $\widetilde{\Sigma}$ is a distributed alphabet $\widetilde{\Gamma} = (\Gamma_1, \ldots, \Gamma_k)$ such that

1. Γ is a finite subset of $E(\widetilde{\Sigma})$ (following the earlier convention, we set $\Gamma = \bigcup_{i \in \mathcal{P}} \Gamma_i$).

2. $loc((a, J)) = loc(a)$ for all $(a, J) \in \Gamma$. That is, the distribution of a-events in $\widetilde{\Gamma}$ respects the distribution of letters in $\widetilde{\Sigma}$.

Let $\widetilde{\Gamma}$ be a distributed interval alphabet based on $\widetilde{\Sigma}$. Once again we can associate a set of timed words with a symbolic word $\widehat{\sigma} \in \Gamma^\infty$.

Definition 6.6 *We define $tw(\widehat{\sigma})$ to be set of all $\sigma \in T\widetilde{\Sigma}^\infty$ such that*

1. $|\sigma| = |\widehat{\sigma}|$.
2. *for each $\tau(a, t) \preceq \sigma$ and $\widehat{\tau}(b, J) \preceq \widehat{\sigma}$ such that $|\tau| = |\widehat{\tau}|$, we have $a = b$, and for each $i \in loc(a)$, $(t - time_i(\tau)) \in J(i)$.*

As before, for $L \subseteq \Gamma^\infty$ we set

$$tw(L) = \bigcup_{\widehat{\sigma} \in L} tw(\widehat{\sigma}).$$

Definition 6.7 *A distributed interval automaton over $\widetilde{\Sigma}$ is just a distributed automaton over a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$.*

A distributed interval automaton $\mathcal{A}$ accepts a timed language $L(\mathcal{A}) \subseteq T\widetilde{\Sigma}^\omega$ given by $L(\mathcal{A}) = tw(L_{sym}(\mathcal{A}))$. We will term such languages *ω-regular distributed interval languages* over $\widetilde{\Sigma}$, and denote this class DIL($\widetilde{\Sigma}$).

As in the case of interval automata and product interval automata, we can define the notion of a run of $\mathcal{A}$ directly on a timed word in $T\widetilde{\Sigma}^\omega$. The timed language accepted in this manner can be seen to coincide with the one given above using the *tw* map.

We remark here that in the the light of Theorem 6.2 an alternate way of arriving at the class DIL($\widetilde{\Sigma}$) is by applying the map *tw* to languages in RCL($\widetilde{\Gamma}$), for some distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$.

Example 6.8 Fig 6.1 shows a distributed interval automaton $\mathcal{A}$ over the distributed alphabet $\widetilde{\Sigma} = (\{a, a', d\}, \{b, b', d\})$. The distributed interval alphabet we have in mind is

$$\widetilde{\Gamma} = (\{(a, \langle I \rangle), (a', \langle I \rangle), (d, \langle I, I \rangle)\}, \{(b, \langle I \rangle), (b', \langle I \rangle), (d, \langle I, I \rangle)\}),$$

where I is the interval $[1, 1]$. We have written the second components of the a-events as ordered tuples instead of functions. Thus $(d, \langle I, I \rangle)$ represents the d-event (d, J) where $J(1) = J(2) = I$.

$\mathcal{A}$ is defined as

$$\mathcal{A} = (\{S_i\}, \{\longrightarrow_{\widehat{a}}\}_{\widehat{a} \in \Gamma}, S_{in}, \{F_i\}, \{G_i\})$$

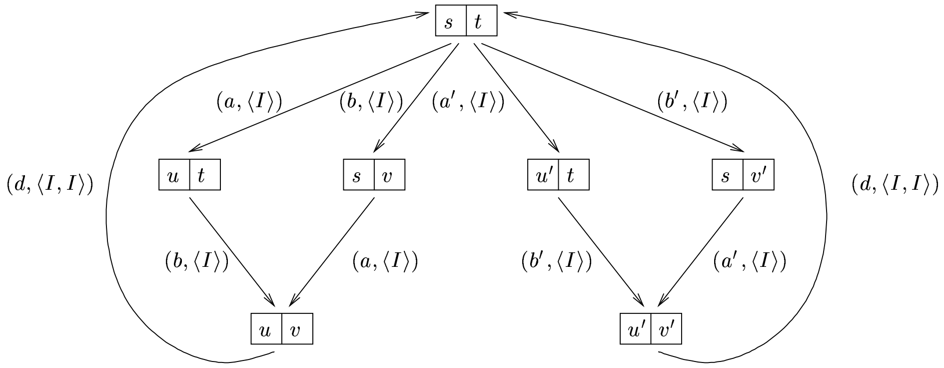

Figure 6.1: A distributed interval automaton over $\widetilde{\Sigma} = (\{a, a', d\}, \{b, b', d\})$.

where $S_1 = \{s, u, u'\}$ and $S_2 = \{t, v, v'\}$, $S_{in} = \{(s, t)\}$, $F_i = \emptyset$ and $G_i = S_i$ for $i \in \mathcal{P} = \{1, 2\}$. The transition relation can be inferred from the figure. For example, $\longrightarrow_{(a,\langle I\rangle)} = \{(s, u)\}$ and $\longrightarrow_{(d,\langle I,I\rangle)} = \{((u, v), (s, t)), ((u', v'), (s, t))\}$.

It is not difficult to see that

$$L_{sym}(\mathcal{A}) = (([(a, I)(b, I)] + [(a', I)(b', I)])(d, I))^{\omega}.$$

Here we use $[x]$ to denote the $\approx_{\widetilde{\Gamma}}$-equivalence class of x, and the symbol '+' to denote the union operation of regular expressions. L^{ω} denotes the infinite concatenation $L \cdot L \cdots$.

$L(\mathcal{A})$—the language of timed words accepted by $\mathcal{A}$—can be seen to be the set of all words in $T\widetilde{\Sigma}^{\omega}$ of the form

$$(x_1, 1)(y_1, 1)(d, 2)(x_2, 3)(y_2, 3)(d, 4)(x_3, 5)(y_3, 5)(d, 6)\cdots$$

where for each $i \in \mathbb{N} - \{0\}$, $x_i \in \{a, a'\}$, $y_i \in \{b, b'\}$, and $x_i = a$ iff $y_i = b$. $\Box$

We now wish to argue that ω-regular product interval languages are strictly weaker than ω-regular distributed interval languages.

Proposition 6.9 *The class of ω-regular product interval languages over $\widetilde{\Sigma}$ is strictly contained in the class of ω-regular distributed interval languages over $\widetilde{\Sigma}$.*

Proof Let $L \subseteq T\widetilde{\Sigma}^{\omega}$ be an ω-regular product interval language. So there exits a product interval automaton $\mathcal{A}$ over $\widetilde{\Sigma}$ with $L(\mathcal{A}) = L$. Let $\mathcal{A} = (\{\mathcal{A}_i\}_{i\in\mathcal{P}}, Q_{in})$, with each $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i, G_i)$. We define a distributed interval automaton $\mathcal{B}$ over $\widetilde{\Sigma}$ such that $L(\mathcal{B}) = L(\mathcal{A})$. Let $\widetilde{\Gamma}$ be the distributed interval alphabet based on $\widetilde{\Sigma}$ given by

$$\Gamma = \{(a, J) \in E(\widetilde{\Sigma}) \mid \forall i \in loc(a)\ \exists q \xrightarrow{(a,I)}_i q' \text{ with } I = J(i)\}.$$

We define $\mathcal{B}$ to be the distributed automaton over $\widetilde{\Gamma}$ given by:

$$\mathcal{B} = (\{Q_i\}, \{\longrightarrow_{\widehat{a}}\}_{\widehat{a} \in \Gamma}, Q_{in}, \{F_i\}, \{G_i\})$$

where the transition relation is defined as follows. Let $(a, J) \in \Gamma$, and let s, s' be (a, J)-states. Then $s \longrightarrow_{(a,J)} s'$ iff for each $i \in loc(a)$, $s(i) \xrightarrow{(a,J(i))}_i s'(i)$. It is not difficult to verify that $L(\mathcal{B}) = L(\mathcal{A})$.

To show the strict inclusion, we argue below that the language L of Example 6.8 is not a regular product interval language over $\widetilde{\Sigma} = (\{a, a', d\}, \{b, b', d\})$.

Suppose L were a regular product interval language. Then there exist regular direct product interval languages $L_1, \ldots, L_m$ $(m \geq 1)$ over $\widetilde{\Sigma}$, such that $L = \bigcup_{i=1}^{m} L_i$. It is easy to see that there must exist some L_i, and timed words σ, σ' in L_i such that $\sigma \restriction 1 \neq \sigma' \restriction 1$. Since we can interchange adjacent x_i's and y_i's and still remain within L_i, without loss of generality we can write σ and σ' as: $\sigma = \tau(a, t)(b, t)(d, t+1)\delta$ and $\sigma' = \tau(a', t)(b', t)(d, t+1)\delta'$, where $\tau \in T\widetilde{\Sigma}^*$ and $\delta, \delta' \in T\widetilde{\Sigma}^\omega$.

Further, we can construct a $\delta'' \in T\widetilde{\Sigma}^\omega$ such that $\delta'' \restriction 1 = \delta' \restriction 1$ and $\delta'' \restriction 2 = \delta \restriction 2$.

Now consider the word $\sigma'' = \tau(a', t)(b, t)(d, t+1)\delta''$. σ'' must belong to L_i, since $\sigma'' \restriction 1 = \sigma' \restriction 1$ and $\sigma'' \restriction 2 = \sigma \restriction 2$. But this is a contradiction, since $\sigma'' \notin L$.

This proves that L cannot be a regular product interval language over $\widetilde{\Sigma}$. $\square$

6.3 Checking emptiness

To check the emptiness of the timed language accepted by a distributed interval automaton we follow a similar route to the one adopted for product interval automaton in Section 3.3.

Let

$$\mathcal{A} = (\{S_i\}, \{\longrightarrow_{\widehat{a}}\}_{\widehat{a} \in \Gamma}, S_{in}, \{F_i\}, \{G_i\})$$

be a distributed interval automaton over $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$. (Once again we assume that each local finitary final state is terminal.) We define a TBA $\mathcal{B}$ with the property that $L_{nd}(\mathcal{B}) \cap T\widetilde{\Sigma}^\infty = L(\mathcal{A})$. Let $\mathcal{B} = (Q, \longrightarrow, Q_{in}, C, F, G)$ where:

- The set of states is $Q = (\prod_{i=1}^{k} S_i) \times \{0, \ldots, k\}$.
- The set of clocks is $C = \{x_i \mid i \in \mathcal{P}\}$.
- Its transition relation is given as follows. We have $(q, l) \xrightarrow[X]{a,g} (q', m)$ iff the following conditions hold:

  1. $X = \{x_i \mid i \in loc(a)\}$,
  2. there exists $(a, J) \in \Gamma$ and (a, J)-states s, s' such that $s \longrightarrow_{(a,J)} s'$, $q[i] = s(i)$ and $q'[i] = s'(i)$ for each $i \in loc(a)$, and for each $i \notin loc(a)$ $q[i] = q'[i]$. Further, $g = \bigwedge_{i \in loc(a)} (x_i \in J(i))$.
  3. $m = (l+1) \bmod (k+1)$ if $q'(l) \in F_l \cup G_l$ or $l = 0$; otherwise $m = l$.

- The initial states are given by $Q_{in} = S_{in} \times \{0\}$.
- $F = \emptyset$
- The infinitary final states are given by $G = \{(q, l) \in Q \mid l = 0\}$.

It is not difficult to argue that $L_{nd}(\mathcal{B}) \cap T\widetilde{\Sigma}^{\infty} = L(\mathcal{A})$.

Finally, we intersect $\mathcal{B}$ with the automaton $\mathcal{B}'$ of Section 3.3 to get the automaton $\mathcal{B}''$ with the property that $L_{nd}(\mathcal{B}'') = L(\mathcal{A})$.

The emptiness check via this route can be seen to take time

$$O(|S| + |E|) \cdot 2^{O(k)} \cdot k! \cdot \prod_{i=1}^{k} (c_i + 1)$$

where S and E are the sets of global states and transitions of $\mathcal{A}$ respectively, and c_i is the largest (normalised) i-type interval bound appearing in $\mathcal{A}$.

Before we close this section we prove a property of distributed interval languages (this applies to product interval languages as well) which allows us a mild saving in the emptiness check for the corresponding automata.

A *non-progressive* timed word over $\widetilde{\Sigma}$ is an element of $(\Sigma \times \mathbb{R}^{>0})^{\omega}$ such that:

(i) if $\tau(a, t)(b, t')$ is a prefix of σ then $t \leq t'$ (non-decreasing).

(ii) if $\tau(a, t)\tau'(b, t')$ is a prefix of σ with $t = t'$, then $loc(a) \cap loc(b) = \emptyset$ (simultaneous actions must be independent).

Thus a non-progressive timed word need not satisfy the progressiveness condition (iii) of Definition 3.1. Let $T_{np}\widetilde{\Sigma}^{\omega}$ denote the set of non-progressive timed words over $\widetilde{\Sigma}$.

We extend the definition of the tw map to non-progressive timed words as well. Let $\widetilde{\Gamma}$ be a distributed interval alphabet based on $\widetilde{\Sigma}$. For a word $\widehat{\sigma}$ in Γ^ω let $tw_{np}(\widehat{\sigma})$ denote the set of non-progressive timed words in $T_{np}\widetilde{\Sigma}^\omega$ which satisfy Definition 6.6.

Proposition 6.10 *Let $\widetilde{\Gamma}$ be a distributed interval alphabet based on $\widetilde{\Sigma}$ and let $\widehat{\sigma} \in \Gamma^\infty$. Then $untime(tw_{np}(\widehat{\sigma})) = untime(tw(\widehat{\sigma}))$.*

Proof We assume for convenience that the rational interval bounds which occur in Γ are integers. It is easy to adapt the proof to the case of non-integral bounds. The proof we give here is based on a similar argument in [AD94] to show that the progressiveness condition added to the region automaton is enough to ensure the existence of progressive timed word along any run which accepts a non-progressive timed word.

Let $\sigma = (\alpha, \eta)$ be a non-progressive timed word in $tw_{np}(\widehat{\sigma})$. If σ itself is progressive, we have nothing to prove. Suppose σ is not progressive. We construct a timed word σ' which is progressive and belongs to $tw(\widehat{\sigma})$.

Now since σ is not progressive, its sequence of time stamps η is monotonically increasing and bounded, which means that it must converge to a value $l \in \mathbb{R}^{>0}$. This would further imply that there exists a natural number l_0 such that for all $n \geq l_0$ we have $\eta(n) - \eta(l_0) < 0.5$. Also, let $X \subseteq \mathcal{P}$ be the set of components which participate infinitely often in σ; then we can assume without loss of generality that for each $n \geq l_0$, $loc(\alpha(n)) \in X$. Since the agents in X participate infinitely often in σ, there exists an infinite sequence of natural numbers $l_1, l_2, \ldots$ such that $l_0 < l_1 < l_2 < \cdots$ and for each $n \geq 0$ and $i \in X$ there exists m with $l_n < m < l_{n+1}$ and $\alpha(m) \in \Sigma_i$.

Now define σ' as (α, η') where η' is given as follows. Let $n \in \mathbb{N} - \{0\}$. Then

$$\eta'(n) \quad = \quad \begin{cases} \eta'(n-1) + (\eta(n) - \eta(n-1)) & \text{if} \quad n \notin \{l_1, l_2, \ldots\} \\ \eta'(n-1) + (\eta(n) - \eta(n-1)) + 0.5 & \text{if} \quad n \in \{l_1, l_2, \ldots\} \end{cases}$$

where we set $\eta'(0) = 0$.

σ' is clearly progressive since its time stamps increase by at least 0.5 time units at infinitely many points along the word. Further, $untime(\sigma') = untime(\sigma) = \alpha$. Thus we need only verify that $\sigma' \in tw(\widehat{\sigma})$.

We first observe an immediate property of our construction. Let τ, τ' be prefixes of σ and σ' respectively, such that $|\tau| = |\tau'|$. Let $i \in loc(action(\tau))$. Then $time_i(\tau') = time_i(\tau) + 0.5$ if $|\tau| \in \{l_1, l_2, \ldots\}$, and $time_i(\tau') = time_i(\tau)$ otherwise. Further, if $|\tau| \in \{l_1, l_2, \ldots,\}$ then $0 < time_i(\tau) < 0.5$. Since our interval bounds are assumed to be integral, it follows that $time_i(\tau') \in I$ iff

$time_i(\tau) \in I$ for any interval I mentioned in Γ. Since $\sigma \in tw_{np}(\widehat{\sigma})$ it follows that $\sigma' \in tw(\widehat{\sigma})$.

It now follows immediately that $untime(tw_{np}(\widehat{\sigma})) = untime(tw(\widehat{\sigma}))$ $\square$

As a consequence of this proposition, while checking the emptiness of the TBA $\mathcal{B}''$ above, we need not add the extra state information in the region automaton $R(\mathcal{B}'')$ to ensure progressive runs.

6.4 Closure under Boolean Operations

Here we wish to show that class of distributed interval languages are closed under boolean operations. To be precise, we fix, as usual, a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_p\}_{p\in\mathcal{P}}$ with $\mathcal{P} = \{1, 2, \ldots, k\}$. Closure under union is straightforward, so here we will concentrate on complementation.

Once more the notion of a proper alphabet will play an important role. We extend the notion of proper interval alphabets introduced in Chapter 2 to distributed interval alphabets too. We say a distributed interval alphabet $\widetilde{\Gamma}$ over $\widetilde{\Sigma}$ is *proper* if the set

$$\Gamma_{a,i} = \{I \mid \exists (a, J) \in \Gamma, \text{ with } J(i) = I\}$$

is a proper interval set for each $a \in \Sigma$ and $i \in loc(a)$. As before, we will say a distributed interval alphabet $\widetilde{\Gamma}$ covers another distributed interval alphabet $\widetilde{\Delta}$ (with both $\widetilde{\Gamma}$ and $\widetilde{\Delta}$ based on $\widetilde{\Sigma}$) iff $\Gamma_{a,i}$ covers $\Delta_{a,i}$ for each $a \in \Sigma$ and $i \in loc(a)$.

Given a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$, we can construct a proper distributed interval alphabet $\widetilde{\Delta}$ based on $\widetilde{\Sigma}$ such that $\widetilde{\Delta}$ covers $\widetilde{\Gamma}$. This can be done by defining

$$\Delta_i = \{(a, J) \in E(\widetilde{\Sigma}) \mid a \in \Sigma_i \text{ and } \forall i \in loc(a),\ J(i) \in prop(\Gamma_{a,i})\}.$$

The following proposition shows that proper alphabets suffice to capture the class $\mathrm{DIL}(\widetilde{\Sigma})$.

Proposition 6.11 *Let $\widetilde{\Gamma}$ be a distributed interval alphabet over $\widetilde{\Sigma}$ and let $\widehat{L} \in \mathrm{RCL}(\widetilde{\Gamma})$. Let $\widetilde{\Delta}$ be a proper distributed interval alphabet covering $\widetilde{\Gamma}$. Then there exists a $\widehat{M} \in \mathrm{RCL}(\widetilde{\Delta})$ such that $tw(\widehat{M}) = tw(\widehat{L})$.*

Proof Since $\widehat{L}$ is in $\mathrm{RCL}(\widetilde{\Gamma})$, by Theorem 6.2 we know that there exists a distributed automaton $\mathcal{A}$ over $\widetilde{\Gamma}$ such that $L_{sym}(\mathcal{A}) = \widehat{L}$. Let $\mathcal{A} =$

$(\{S_i\}, \{\longrightarrow_{(a,J)}\}_{(a,J)\in\Gamma}, S_{in}, \{F_i\}, \{G_i\})$. Consider now the distributed automaton

$$\mathcal{B} = (\{S_i\}, \{\Longrightarrow_{(a,J)}\}_{(a,J)\in\Delta}, S_{in}, \{F_i\}, \{G_i\})$$

over $\widetilde{\Delta}$ given by: $s_a \Longrightarrow_{(a,J')} s'_a$ iff we have $s_a \longrightarrow_{(a,J)} s'_a$ with $J'(i) \subseteq J(i)$ for each $i \in loc(a)$.

Taking $\widehat{M} = L_{sym}(\mathcal{B})$ it is quite easy to prove that $tw(\widehat{M}) = tw(\widehat{L})$. $\Box$

Proper distributed interval alphabets also enjoy a property analogous to Lemma 2.11 for interval alphabets:

Proposition 6.12 *Let $\widetilde{\Gamma}$ be a proper distributed interval alphabet based on $\widetilde{\Sigma}$ and let $\sigma \in T\widetilde{\Sigma}^\omega$. Then there exists a unique $\widehat{\sigma} \in \Gamma^\omega$ such that $\sigma \in tw(\widehat{\sigma})$.*

Proof Consider the word $\widehat{\sigma} \in \Gamma^\omega$ defined as follows. Let $i \in \mathbb{N} - \{0\}$. Let $\tau(a,t)$ be the prefix of σ such that $|\tau| = i - 1$. Then we define $\widehat{\sigma}(i) = (a, J)$ where for each $j \in loc(a)$, $J(j)$ is the unique interval I in $\Gamma_{a,j}$ such that $(t - time_j(\tau)) \in I$. By construction, we have that $\sigma \in tw(\widehat{\sigma})$.

To argue that $\widehat{\sigma}$ is the only word in Γ^ω for which $\sigma \in tw(\widehat{\sigma})$, let $\widehat{\sigma}' \in \Gamma^\omega$ with $\sigma \in tw(\widehat{\sigma}')$. We show that for every $i \in \mathbb{N}$, $\widehat{\sigma}'(i) = \widehat{\sigma}(i)$. Let $i \in \mathbb{N}$ and let $\widehat{\sigma}(i) = (b, J)$ and $\widehat{\sigma}'(i) = (b', J')$. Let $\tau(a,t)$ be the prefix of σ such that $|\tau| = i - 1$. Then clearly $b = b' = a$. Further, for each $j \in loc(a)$ we have $(t - time_j(\tau)) \in J(j)$ and $(t - time_j(\tau)) \in J'(j)$. But both $J(j)$ and $J'(j)$ belong to $\Gamma_{a,j}$. Since $\Gamma_{a,j}$ is a proper interval set, we must have $J(j) = J'(j)$. This proves that $\widehat{\sigma}' = \widehat{\sigma}$. $\Box$

Now let L be a distributed interval language over $\widetilde{\Sigma}$. So there exists a proper distributed interval alphabet $\widetilde{\Gamma}$ over $\widetilde{\Sigma}$, and a distributed automaton $\mathcal{A}$ over $\widetilde{\Gamma}$, such that $L = tw(L_{sym}(\mathcal{A}))$. By the complementation results established in [KMS95, M94] we know that there exists a distributed interval automaton $\mathcal{B}$ over $\widetilde{\Gamma}$ such that $L_{sym}(\mathcal{B}) = \Gamma^\omega - L_{sym}(\mathcal{A})$.

It now follows using Proposition 6.12 that

$$L(\mathcal{B}) = tw(\Gamma^\omega - L_{sym}(\mathcal{A})) = T\widetilde{\Sigma}^\omega - tw(L_{sym}(\mathcal{A})) = T\widetilde{\Sigma}^\omega - L.$$

This lets us conclude that the class $\mathrm{DIL}(\widetilde{\Sigma})$ is closed under complementation.

It is easy to see that the class of distributed interval languages over $\widetilde{\Sigma}$ is closed under union. Closure under union and complementation give us closure under intersection too. Thus, we have the following result:

Theorem 6.13 *The class $\mathrm{DIL}(\widetilde{\Sigma})$ of ω-regular distributed interval languages over $\widetilde{\Sigma}$ is closed under the boolean operations of union, intersection and complementation.*

6.5 Timed trace languages

The aim of this section is to obtain a representation result for distributed interval languages via timed trace languages. This result will be along the lines of the representation of ω-regular consistent languages via ω-regular trace languages stated in Proposition 6.4.

We begin with the notion of a *timed trace.*

Definition 6.14 *A* timed trace *over* $\widetilde{\Sigma}$ *is a pair* (T, θ) *where*

- $T = (E, \leq, \lambda)$ *is a trace over* $\widetilde{\Sigma}$
- $\theta : E \to \mathbb{R}^{>0}$ *satisfying*
  - $e < e' \Rightarrow \theta(e) <' \theta(e')$ *where* $<'$ *is the usual ordering over reals (we will call this* $<$ *itself, since the intended ordering will be clear from the context), and*
  - *there exists a linearization* δ *of* $(E, \leq)$*, such that* δ *respects* θ *(i.e.* $\theta(e) < \theta(e') \Rightarrow e < e'$ *in* δ*) and* $\theta(\delta)$ *is progressive—i.e. it forms a divergent sequence of reals.*

Let $\mathrm{TTr}(\widetilde{\Sigma})$ denote the set of timed traces over $\widetilde{\Sigma}$.

Let (T, θ) be a timed trace over a distributed alphabet $\widetilde{\Sigma}$, with $T = (E, \leq, \lambda)$. For each $e \in E$ and $i \in \mathcal{P}$ the set

$$\downarrow_i e = \{e' \in E \mid e' < e \text{ and } \lambda(e') \in \Sigma_i\}$$

is a finite, totally ordered subset of E. When this set is non-empty we define $last_i((T, \theta), e)$ to be the maximum element of $\downarrow_i e$, and $time_i((T, \theta)), e)$ to be $\theta(last_i((T, \theta), e))$. If the set $\downarrow_i e$ is empty, we define $time_i((T, \theta), e) = 0$.

Let $\widetilde{\Gamma}$ be a distributed interval alphabet based on $\widetilde{\Sigma}$. Given a trace $T = (E, \leq, \lambda)$ over the symbolic alphabet $\widetilde{\Gamma}$ we can associate a set of timed traces $tt(T)$ with T as follows: Let (T', θ') be a timed trace over $\widetilde{\Sigma}$, with $T' = (E', \leq', \lambda')$. Then $(T', \theta') \in tt(T)$ iff

- $(E', \leq') = (E, \leq)$ and for each $e \in E$, $\lambda'(e) = a$ iff $\lambda(e) = (a, J)$ for some J. That is, λ' is the projection of λ to the Σ-component of the alphabet Γ.
- Let $e \in E$ with $\lambda(e) = (a, J)$. Then for each $i \in loc(a)$ $(\theta'(e) - time_i((T', \theta'), e)) \in J(i)$.

Next, we observe that the uniqueness property of symbolic traces over a proper distributed interval alphabet holds for traces as well.

Lemma 6.15 *Let $\widetilde{\Gamma}$ be a proper distributed interval alphabet based on $\widetilde{\Sigma}$. Let (T', θ') be a timed trace over $\widetilde{\Sigma}$. Then there exists a unique trace $T \in \mathrm{Tr}(\widetilde{\Gamma})$ such that $(T', \theta') \in tt(T)$.*

Proof Let $T' = (E', \leq', \lambda')$. Define $T = (E, \leq, \lambda)$ with $E = E'$, $\leq = \leq'$, and λ given as follows. Let $e \in E$ and let $\lambda'(e) = a$. Then we define $\lambda(e) = (a, J)$ where for each $i \in loc(a)$, $J(i)$ is the unique interval in $\Gamma_{a,i}$ such that $\theta'(e) - time_i((T', \theta'), e) \in J(i)$. T is clearly a trace over $\widetilde{\Gamma}$ and, by construction, $(T', \theta') \in tt(T)$.

The uniqueness of T (up to isomorphism) follows easily using the fact that $\widetilde{\Gamma}$ is proper. $\square$

We say a subset M of $\mathrm{TTr}(\widetilde{\Sigma})$ is an ω-regular timed trace language over $\widetilde{\Sigma}$ iff there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$ and an ω-regular trace language $\widehat{M}$ over $\widetilde{\Gamma}$, with $M = tt(\widehat{M})$. Let $\mathrm{RTTrL}(\widetilde{\Sigma})$ denote the class of ω-regular timed trace languages over $\widetilde{\Sigma}$.

We now define maps *tt-tw* ("timed traces to timed words") and *tw-tt* ("timed words to timed traces") which take us back and forth between distributed interval languages and timed traces languages.

Definition 6.16

1. *The map tw-tt* : $\mathrm{DIL}(\widetilde{\Sigma}) \to \mathrm{RTTrL}(\widetilde{\Sigma})$ *is given as follows. Let* $L \in \mathrm{DIL}(\widetilde{\Sigma})$. *Then there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$ and $\widehat{L} \in \mathrm{RCL}(\widetilde{\Gamma})$ such that $L = tw(\widehat{L})$. We set $tw\text{-}tt(L) = tt(w\text{-}t(\widehat{L}))$.*

2. *The map tt-tw* : $\mathrm{RTTrL}(\widetilde{\Sigma}) \to \mathrm{DIL}(\widetilde{\Sigma})$ *is given as follows. Let* $M \in \mathrm{RTTrL}(\widetilde{\Sigma})$. *Then there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$ and $\widehat{M} \in \mathrm{RTrL}(\widetilde{\Gamma})$ such that $M = tw(\widehat{M})$. We set $tt\text{-}tw(M) = tw(t\text{-}w(\widehat{M}))$.*

We need to show that the maps *tw-tt* and *tt-tw* are indeed well-defined.

Proposition 6.17 *tw-tt and tt-tw are well-defined maps from* $\mathrm{DIL}(\widetilde{\Sigma})$ *to* $\mathrm{RTTrL}(\widetilde{\Sigma})$ *and* $\mathrm{RTTrL}(\widetilde{\Sigma})$ *to* $\mathrm{DIL}(\widetilde{\Sigma})$ *respectively.*

Proof We prove that *tw-tt* is well-defined. The proof that *tt-tw* is well-defined proceeds in a similar manner.

Let $\widetilde{\Gamma}_1$ and $\widetilde{\Gamma}_2$ be distributed interval alphabets based on $\widetilde{\Sigma}$, and let $\widehat{L}_1 \in \mathrm{RCL}(\widetilde{\Gamma}_1)$ and $\widehat{L}_2 \in \mathrm{RCL}(\widetilde{\Gamma}_2)$ such that $tw(\widehat{L}_1) = tw(\widehat{L}_2) = L$. Let $\widehat{M}_1 = w\text{-}t(\widehat{L}_1)$ and let $\widehat{M}_2 = w\text{-}t(\widehat{L}_2)$. Then we show that $tt(\widehat{M}_1) = tt(\widehat{M}_2)$.

We first show that $tt(\widehat{M}_1) \subseteq tt(\widehat{M}_2)$.

Let $(T, \theta) \in tt(\widehat{M_1})$ with $T = (E, \leq, \lambda)$. So there exists $\widehat{T_1} \in \widehat{M_1}$ with $(T, \theta) \in tt(\widehat{T_1})$. Now T has a linearization δ such that $\theta(\delta)$ is progressive. Let $\widehat{\sigma}_1$ be the corresponding linearization of $\widehat{T}_1$. (Since $\widehat{T}_1$ and T are isomorphic, the isomorphism induces a linearization of $\widehat{T}_1$ corresponding to δ.) By Proposition 6.4, it follows that $\widehat{\sigma}_1 \in \widehat{L}_1$. We also note here that the underlying poset of T is isomorphic to the underlying poset of $w\text{-}t(\widehat{\sigma}_1)$ and hence we can assume that E is the set of non-empty prefixes of $\widehat{\sigma}_1$.

Consider now the timed word $\sigma = (\alpha, \eta)$ where α is the projection of $\widehat{\sigma}_1$ to the Σ component—i.e. for each $i \in \mathbb{N}$, $\alpha(i) = a$ iff $\widehat{\sigma}_1(i) = (a, J)$ for some J; for each non-empty prefix w of α we have $\eta(w) = \theta(\widehat{\tau})$ where $\widehat{\tau}$ is the prefix of $\widehat{\sigma}_1$ such that $|\widehat{\tau}| = |w|$. It is not difficult to verify that σ is progressive and hence is a timed word over $\widetilde{\Sigma}$. We now claim that $\sigma \in tw(\widehat{\sigma}_1)$. This is easy to infer once we observe that for any prefix τ of σ, and $\widehat{\tau}$ of $\widehat{\sigma}_1$ such that $|\tau| = |\widehat{\tau}|$, we have $last_i(\tau)$ exists iff $last_i((T, \theta), \widehat{\tau})$ exists and $|last_i(\tau)| = |last_i((T, \theta), \widehat{\tau})|$. This is easy to verify using the fact that $\widehat{\sigma}_1$ is a linearization of T. Thus we conclude that $\sigma \in tw(\widehat{\sigma}_1)$.

Now since $tw(\widehat{L}_1) = tw(\widehat{L}_2)$, there exists $\widehat{\sigma}_2 \in \widehat{L}_2$ such that $\sigma \in tw(\widehat{\sigma}_2)$. Since $\sigma \in tw(\widehat{\sigma}_1)$ and $\sigma \in tw(\widehat{\sigma}_2)$, the sequence of Σ-actions in $\widehat{\sigma}_1$ and $\widehat{\sigma}_2$ are the same. Hence, if $\widehat{T}_2 = w\text{-}t(\widehat{\sigma}_2)$ then $\widehat{T}_1$ and $\widehat{T}_2$ are isomorphic as posets, where the isomorphism matches equal length prefixes of $\widehat{\sigma}_1$ and $\widehat{\sigma}_2$. Thus to show that $(T, \theta) \in tt(\widehat{T}_2)$ we need only verify the following: Let $\widehat{\tau}_1$ be a prefix of $\widehat{\sigma}_1$, and $\widehat{\tau}_2$ a prefix of $\widehat{\sigma}_2$, such that $|\widehat{\tau}_1| = |\widehat{\tau}_2|$. Further, let $\lambda(\widehat{\tau}_1) = a$ and let (a, J) be the label of $\widehat{\tau}_2$ in $\widehat{T}_2$. Then we have $\theta(\widehat{\tau}_1) - time_i((T, \theta), \widehat{\tau}_1) \in J(i)$ for each $i \in loc(a)$. This is again easy to verify using the fact that $\sigma \in tw(\widehat{\sigma}_2)$. This completes the proof that $tt(\widehat{M_1}) \subseteq tt(\widehat{M_2})$.

By a symmetric argument it follows that $tt(\widehat{M_2}) \subseteq tt(\widehat{M_1})$. Thus $tt(\widehat{M_1}) = tt(\widehat{M_2})$. This proves that $tw\text{-}tt$ is a well-defined map. $\square$

Theorem 6.18 *The classes* $\mathrm{DIL}(\widetilde{\Sigma})$ *and* $\mathrm{RTTrL}(\widetilde{\Sigma})$ *represent each other via the maps tw-tt and tt-tw in the following sense:*

1. *Let* $L \in \mathrm{DIL}(\widetilde{\Sigma})$*. Then* $tt\text{-}tw(tw\text{-}tt(L)) = L$.
2. *Let* $M \in \mathrm{RTTrL}(\widetilde{\Sigma})$*. Then* $tw\text{-}tt(tt\text{-}tw(M)) = M$.

Proof

1. Let $tw\text{-}tt(L) = M$. So there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$, and $\widehat{L} \in \mathrm{RCL}(\widetilde{\Gamma})$ such that $L = tw(\widehat{L})$, and $M = tt(w\text{-}t(\widehat{L}))$. Let $\widehat{M} = w\text{-}t(\widehat{L})$. Now, since $\widehat{M} \in \mathrm{RTrL}(\widetilde{\Gamma})$ and $M =$

$tt(\widehat{M})$, we have $tt\text{-}tw(M) = tw(t\text{-}w(\widehat{M}))$. By Proposition 6.4, $t\text{-}w(\widehat{M}) = \widehat{L}$. Thus $tt\text{-}tw(M) = tw(\widehat{L})$, and hence $tt\text{-}tw(M) = L$.

2. Let $tt\text{-}tw(M) = L$. Then there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$, and $\widehat{M} \in \mathrm{RTrL}(\widetilde{\Gamma})$ such that $M = tt(\widehat{M})$, and $L = tw(t\text{-}w(\widehat{M}))$. Let $\widehat{L} = t\text{-}w(\widehat{M})$. Now, since $\widehat{L} \in \mathrm{RCL}(\widetilde{\Gamma})$ with $L = tw(\widehat{L})$, we have $tw\text{-}tt(L) = tt(w\text{-}t(\widehat{L}))$ which equals $tt(\widehat{M})$ by Proposition 6.4. Thus $tw\text{-}tt(L) = M$.

$\square$

6.6 A logical characterisation

In this section we provide a logical characterisation for the class of ω-regular timed trace languages over $\widetilde{\Sigma}$. The logic, called $\mathrm{TMSOTr}(\widetilde{\Sigma})$, will have the flavour of MSOTr but in a timed setting.

As per the convention used in the earlier logics, we assume a supply of individual variables $x, y, \ldots$ and a supply of set variables $X, Y, \ldots$ which will be interpreted as events and sets of events, respectively, in the partial order of the timed trace under consideration.

Definition 6.19 *The formulas of* $\mathrm{TMSOTr}(\widetilde{\Sigma})$ *are given by the following syntax:*

$$\varphi ::= (x \leq y) \mid Q_a(x) \mid \Delta_i(x, I) \mid (x \in X) \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x\varphi \mid \exists X\varphi.$$

In the atomic formula $Q_a(x)$ we require $a \in \Sigma$, and in $\Delta_i(x, I)$ we require $i \in \mathcal{P}$ and $I \in \mathcal{IR}$.

The formulas of $\mathrm{TMSOTr}(\widetilde{\Sigma})$ are interpreted over timed traces over $\widetilde{\Sigma}$ as described below. Let (T, θ) be timed trace over $\widetilde{\Sigma}$. Let $T = (E, \leq, \lambda)$. Let $\mathbb{I}$ be an interpretation of variables which maps each individual variable (respectively set variable) to an element in E (respectively, a subset of E). Then the atomic formulas are interpreted as follows:

$$\begin{array}{lll}
(T,\theta) \models_{\mathbb{I}} (x \leq y) & \text{iff} & \mathbb{I}(x) \leq \mathbb{I}(y) \text{ in the partial order } (E, \leq) \\
(T,\theta) \models_{\mathbb{I}} Q_a(x) & \text{iff} & \lambda(\mathbb{I}(x)) = a \\
(T,\theta) \models_{\mathbb{I}} (x \in X) & \text{iff} & \mathbb{I}(x) \in \mathbb{I}(X) \\
(T,\theta) \models_{\mathbb{I}} \Delta_i(x, I) & \text{iff} & \lambda(\mathbb{I}(x)) \in \Sigma_i \text{ and } \theta(\mathbb{I}(x)) - time_i((T,\theta), \mathbb{I}(x)) \in I.
\end{array}$$

The boolean operators $\vee$ and $\neg$, and the existential quantifiers $\exists x$ and $\exists X$ are interpreted in the usual way.

For a sentence $\varphi \in \mathrm{TMSOTr}(\widetilde{\Sigma})$, let $L(\varphi)$ denote the set

$$L(\varphi) = \{(T, \theta) \in \mathrm{TTr}(\widetilde{\Sigma}) \mid (T, \theta) \models \varphi\}.$$

We will show that the class of timed trace languages definable by $\mathrm{TMSOTr}(\widetilde{\Sigma})$ sentences is precisely the class of ω-regular timed trace languages over $\widetilde{\Sigma}$. We will do this by translating back and forth between formulas in $\mathrm{TMSOTr}(\widetilde{\Sigma})$ and $\mathrm{MSOTr}(\widetilde{\Gamma})$, for a suitable $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$, and then appealing to Theorem 6.5.

Definition 6.20 *Let $\varphi \in \mathrm{TMSOTr}(\widetilde{\Sigma})$. Let $\widetilde{\Gamma}$ be a* proper *distributed interval alphabet based on $\widetilde{\Sigma}$ such that for each $i \in \mathcal{P}$ and $a \in \Sigma_i$, $\Gamma_{a,i}$ covers the set*

$$voc_i(\varphi) = \{I \mid \Delta_i(x, I) \text{ is a subformula of } \varphi\}.$$

Then the formula $t\text{-}s(\varphi)$ (w.r.t. $\widetilde{\Gamma}$) in $\mathrm{MSOTr}(\widetilde{\Gamma})$ is obtained by replacing in φ atomic formulas of the form $Q_a(x)$ by

$$\bigvee_{(b,J)\in\Gamma,\, b=a} Q_{(b,J)}(x)$$

and atomic formulas of the form $\Delta_i(x, I)$ by the formula

$$\bigvee_{(a,J)\in\Gamma,\, a\in\Sigma_i,\, J(i)\subseteq I} Q_{(a,J)}(x).$$

Lemma 6.21 *Let φ and $\widetilde{\Gamma}$ be as above. Let $\mathbb{I}$ be an interpretation for variables. Then we can say the following:*

1. *Let $T \in \mathrm{Tr}(\widetilde{\Gamma})$ and let $(T', \theta') \in tt(T)$. Then*

$$(T', \theta') \models_{\mathbb{I}} \varphi \text{ iff } T \models_{\mathbb{I}} t\text{-}s(\varphi).$$

2. *If φ is a sentence, then $L(\varphi) = tt(L(t\text{-}s(\varphi)))$.*

Proof

1. We proceed by induction on the structure of φ. Let $T = (E, \leq, \lambda)$ and $T' = (E, \leq, \lambda')$. We consider the base cases when φ is of the form $Q_a x$ and $\Delta_i(x, I)$. The remaining base cases and induction steps are routine.
   (a) Case $\varphi = Q_a(x)$: $(T', \theta') \models_{\mathbb{I}} Q_a(x)$ iff $\lambda'(\mathbb{I}(x)) = a$ which holds iff $\lambda(\mathbb{I}(x)) = (b, J)$ for some $(b, J) \in \Gamma$ with $b = a$. This in turn holds iff $T \models_{\mathbb{I}} \bigvee_{(b,J)\in\Gamma,\, b=a} Q_{(b,J)}(x)$.

(b) Case $\varphi = \Delta_i(x, I)$: Let $(T', \theta') \models_{\mathbb{I}} \Delta_i(x, I)$. Then we know $\lambda'(\mathbb{I}(x)) \in \Sigma_i$ and $\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in I$. Let $\lambda(\mathbb{I}(x)) = (a, J)$. So $a = \lambda'(\mathbb{I}(x)) \in \Sigma_i$. Further, since $(T', \theta') \in tt(T)$, we have $\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in J(i)$. It follows that $I \cap J(i) \neq \emptyset$. Since $I \in voc_i(\varphi)$ and $\Gamma_{a,i}$ covers $voc_i(\varphi)$, we must have $J(i) \subseteq I$. Since we know that $T \models_{\mathbb{I}} Q_{(a,J)}(x)$ with $a \in \Sigma_i$ and $J(i) \subseteq I$, we have:

$$T \models_{\mathbb{I}} \bigvee_{(a,J) \in \Gamma,\, a \in \Sigma_i,\, J(i) \subseteq I} Q_{(a,J)}(x).$$

Conversely, let $T \models_{\mathbb{I}} t\text{-}s(\Delta_i(x, I))$. i.e.

$$T \models_{\mathbb{I}} \bigvee_{(a,J) \in \Gamma\, a \in \Sigma_i,\, J(i) \subseteq I} Q_{(a,J)}(x).$$

Thus, $T \models_{\mathbb{I}} Q_{(a,J)}(x)$ for some $a \in \Sigma_i$ and $(a, J) \in \Gamma$ such that $J(i) \subseteq I$. So, $\lambda(\mathbb{I}(x)) = (a, J)$. Further, since $(T', \theta') \in tt(T)$, we have $\lambda'(\mathbb{I}(x)) = a \in \Sigma_i$ and

$$\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in J(i).$$

Since $J(i) \subseteq I$, we have

$$\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in I.$$

Thus, $(T', \theta') \models_{\mathbb{I}} \Delta_i(x, I)$.

2. Let $(T', \theta') \models \varphi$. Since $\widetilde{\Gamma}$ is a proper distributed interval alphabet, by Lemma 6.15 there is a trace T in $\mathrm{Tr}(\widetilde{\Gamma})$ such that $(T', \theta') \in tt(T)$. By part 1 above, we know $T \models t\text{-}s(\varphi)$. Thus $(T', \theta') \in tt(L(t\text{-}s(\varphi)))$.

   Conversely, let $(T', \theta') \in tt(L(t\text{-}s(\varphi)))$. So there exists T in $\mathrm{Tr}(\widetilde{\Gamma})$ such that $T \models t\text{-}s(\varphi))$ and $(T', \theta') \in tt(T)$. Again, by part 1 of the lemma, $(T', \theta') \models \varphi$, and $(T', \theta') \in L(\varphi)$.

□

In a similar manner, we can go from a formula φ in the "symbolic" logic $\mathrm{MSOTr}(\widetilde{\Gamma})$ to a formula $s\text{-}t(\varphi)$ in $\mathrm{TMSOTr}(\widetilde{\Sigma})$ by replacing subformulas of the form $Q_{(a,J)}(x)$ in φ by the formula

$$Q_a(x) \wedge \bigwedge_{i \in loc(a)} \Delta_i(x, J(i)).$$

Lemma 6.22 *Let $\widetilde{\Gamma}$ be a proper distributed interval alphabet based on $\widetilde{\Sigma}$. Let φ be a formula in* $\mathrm{MSOTr}(\widetilde{\Gamma})$. *Let $\mathbb{I}$ be an interpretation for variables.*

1. *Let $T \in \mathrm{Tr}(\widetilde{\Gamma})$ and let $(T', \theta') \in tt(T)$. Then*

$$T \models_{\mathbb{I}} \varphi \text{ iff } (T', \theta') \models_{\mathbb{I}} s\text{-}t(\varphi).$$

2. *If φ is a sentence then $tt(L(\varphi)) = L(s\text{-}t(\varphi))$.*

Proof

1. The proof once again proceeds by induction on the structure of φ. We give the argument for the case when φ is the atomic formula $Q_{(a,J)}(x)$.

   Let $T \models_{\mathbb{I}} Q_{(a,J)}(x)$. Then $\lambda(\mathbb{I}(x)) = (a, J)$. Since $(T', \theta') \in tt(T)$, we know $\lambda'(\mathbb{I})(x) = a$ and $\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in J(i)$ for all $i \in loc(a)$. Hence

$$(T', \theta') \models_{\mathbb{I}} Q_a(x) \wedge \bigwedge_{i \in loc(a)} \Delta_i(x, J(i)).$$

   Conversely, let

$$(T', \theta') \models_{\mathbb{I}} Q_a(x) \wedge \bigwedge_{i \in loc(a)} \Delta_i(x, J(i)).$$

   (having fixed an element (a, J) of Γ). Let $\lambda(\mathbb{I}(x)) = (b, J')$. Since $(T', \theta') \models_{\mathbb{I}} Q_a(x)$ we have $\lambda'(\mathbb{I}(x)) = a$, and hence $b = a$. Further, since

$$(T', \theta') \models_{\mathbb{I}} \bigwedge_{i \in loc(a)} \Delta_i(x, J(i))$$

   we have, for each $i \in loc(a)$, $\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in J(i)$. Further, using the fact that $(T', \theta') \in tt(T)$ we have for all $i \in loc(a)$ $\theta'(\mathbb{I}(x)) - time_i((T', \theta'), \mathbb{I}(x)) \in J'(i)$. Thus, for all $i \in loc(a)$ we have $J'(i) \cap J(i) \neq \emptyset$. But $\widetilde{\Gamma}$ is proper alphabet. Hence $J(i) = J'(i)$, and we have

$$T \models_{\mathbb{I}} Q_{(a,J)}(x).$$

2. Let $(T', \theta') \in tt(L(\varphi))$. So there exists T in $\mathrm{Tr}(\widetilde{\Gamma})$ such that $T \models \varphi$ and $(T', \theta') \in tt(T)$. By part 1 of the lemma, $(T', \theta') \models s\text{-}t(\varphi)$, and we have $(T', \theta') \in L(s\text{-}t(\varphi))$.

   Conversely, let $(T', \theta') \models s\text{-}t(\varphi)$. Since $\widetilde{\Gamma}$ is a proper distributed interval alphabet, by Lemma 6.15 there is a trace T in $\mathrm{Tr}(\widetilde{\Gamma})$ such that $(T', \theta') \in tt(T)$. By part 1 above, we know $T \models \varphi$. Thus $(T', \theta') \in tt(L(\varphi))$.

$\square$

We can now state and prove the main result of this section.

Theorem 6.23 *$L \subseteq \mathrm{TTr}(\widetilde{\Sigma})$ is a ω-regular timed trace language over $\widetilde{\Sigma}$ iff it is definable in $\mathrm{TMSOTr}(\widetilde{\Sigma})$. That is:*

$$L \in \mathrm{RTTrL}(\widetilde{\Sigma}) \text{ iff } L = L(\varphi) \text{ for some sentence } \varphi \in \mathrm{TMSOTr}(\widetilde{\Sigma}).$$

Proof Let $M \in \mathrm{RTTrL}(\widetilde{\Sigma})$. So there exists a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$, and a language $\widehat{M} \in \mathrm{RTrL}(\widetilde{\Gamma})$ such that $M = tt(\widehat{M})$. By the Theorem 6.5, there exists a sentence φ in $\mathrm{MSOTr}(\widetilde{\Gamma})$ such that $\widehat{M} = L(\varphi)$. By Lemma 6.22 we know that

$$L(s\text{-}t(\varphi)) = tt(\widehat{M}) = M.$$

Conversely, let $M = L(\varphi)$ for some sentence φ in $\mathrm{TMSOTr}(\widetilde{\Sigma})$. Let $\widetilde{\Gamma}$ be a proper distributed interval alphabet based on $\widetilde{\Sigma}$ such that for each $i \in \mathcal{P}$ and $a \in \Sigma_i$, $\Gamma_{a,i}$ covers $voc_i(\varphi)$. Then by Lemma 6.21 we know there exists a sentence $\varphi' = t\text{-}s(\varphi)$ in $\mathrm{MSOTr}(\widetilde{\Gamma})$ such that

$$tt(L(\varphi')) = L(\varphi) = M.$$

Again, appealing to Theorem 6.5, we know that $L(\varphi')$ belongs to $\mathrm{RTrL}(\widetilde{\Gamma})$. Since $M = tt(L(\varphi'))$ we have that $M \in \mathrm{RTTrL}(\widetilde{\Sigma})$. $\square$

Chapter 7

Cellular interval automata

In classical trace theory *asynchronous cellular automata* are known to provide an alternate automata-theoretic characterisation of regular trace languages. Cellular asynchronous automata are again a network of automata based on a distributed alphabet. They have a separate component for each *action* in the alphabet, in contrast to distributed automata which have a component for each location in the distributed alphabet. Each component in the cellular automaton can read the states of the components of actions dependent on it, but must make a move by changing its own local state only.

In this chapter we consider a timed version of these automata called *cellular interval automata*. Here we have a component for each action, and a clock for each component. A component can not only read the states of the components of actions dependent on it, but can also make use of timing constraints based on the the values of the clocks of these components. In addition, the clock of the component corresponding to each action a is reset with every occurrence of a.

In contrast to the classical untimed setting, cellular interval automata are more powerful than distributed interval automata. This is due to the fact we now have a clock for each action, rather than one for each component. In fact, as we show in Section 7.4, they have the same expressive power as the event recording automata of Alur, Fix and Henzinger [AFH94].

The class of timed languages accepted by cellular interval automata are closed under boolean operations. In addition they admit a logical characterisation along the lines of that for distributed interval languages.

Finally, making use of the techniques introduced so far, we give a full monadic second order logic characterisation of event recording languages.

7.1 Preliminaries

The underlying symbolic structure of cellular interval automata correspond to *quasi* asynchronous cellular automata (qACA's). In this section we introduce these automata, and show that, like distributed automata, they too recognise exactly the class of ω-regular consistent languages. We will use this fact to establish several results about the languages accepted by cellular interval automata.

We will need to extend the notation of the last chapter regarding $\mathcal{P}$-indexed families of sets to other families of sets as well.

Let $\{S_x\}_{x \in X}$ be a family of X-indexed sets of states. Let $Y \subseteq X$. Then a Y-state (with respect to the family $\{S_x\}_{x \in X}$) is a map $s : Y \to \bigcup_{x \in Y} S_x$ such that $s(x) \in S_x$ for each $x \in Y$. Let S_Y denote the set of Y-states. For a Y-state s, and $Z \subseteq Y$, we use s_Z to denote the Z-state s' given by $s'(x) = s(x)$ for each $x \in Z$.

Till further notice, let us fix a $\mathcal{P}$-distributed alphabet $\widetilde{\Delta}$, and a partition of Δ, $\mathcal{C} = \{C_x\}_{x \in X}$. We will need some notation relating locations of $\mathcal{P}$ and blocks of the partition $\mathcal{C}$. Let $a \in \Delta$. Then there exists a unique $x \in X$ such that $a \in C_x$; we set $comp(a) = x$. For $i \in \mathcal{P}$ we will use $comp(i)$ to mean the set $\{comp(a) \mid a \in \Delta_i\}$.

For elements of X we can define an analogous notion of "locations" and "dependent" blocks (with respect to $\widetilde{\Delta}$ and $\mathcal{C}$). For $x \in X$, we define

$$loc(x) = \bigcup_{a \in C_x} loc(a).$$

and the set of blocks dependent on x to be

$$D(x) = \{y \in X \mid loc(x) \cap loc(y) \neq \emptyset\}.$$

Finally, we say $\mathcal{C}$ is a *valid* partition (with respect to $\widetilde{\Delta}$) iff for each $x \in X$ and for each $a, b \in C_x$, we have $loc(a) = loc(b)$. Note that if $\mathcal{C}$ is a valid partition with respect to $\widetilde{\Delta}$, then for any $a \in \Delta$ and $x \in X$ such that $a \in C_x$, we have $loc(a) = loc(x)$.

Example 7.1 *Let $\widetilde{\Delta}$ be the distributed alphabet $(\{a, b, e\}, \{c, d, e\})$. Then the partition $\mathcal{C} = (\{a, b\}, \{c\}, \{d\}, \{e\})$ of Δ is a valid with respect to $\widetilde{\Delta}$.*

We are now ready to define a quasi asynchronous cellular automaton (qACA).

Definition 7.2 *Let $\widetilde{\Delta}$ be $\mathcal{P}$-distributed alphabet. Let $\mathcal{C} = \{C_x\}_{x \in X}$ be a partition of Δ which is valid with respect to $\widetilde{\Delta}$. Then a* qACA *$\mathcal{A}$ over $(\widetilde{\Delta}, \mathcal{C})$ is a structure*

$$\mathcal{A} = (\{Q_x\}_{x \in X}, \{\longrightarrow_a\}_{a \in \Delta}, Q_{in}, \{F_x\}_{x \in X}, \{G_x\}_{x \in X}).$$

where

- *For each $x \in X$, Q_x is a finite set of x-local states.*
- *$\longrightarrow_a \subseteq Q_{D(y)} \times Q_y$ where $y = comp(a)$.*
- *$Q_{in} \subseteq Q_X$.*
- *$F_x, G_x \subseteq Q_x$ are the finitary and infinitary accepting states for component x.*

Let $\mathcal{A}$ be a qACA as above. Then we can define a global transition relation $\longrightarrow_{\mathcal{A}} \subseteq Q_X \times \Delta \times Q_X$ given by $q \xrightarrow{a}_{\mathcal{A}} q'$ iff

- $q_{D(y)} \longrightarrow_a q'(y)$ where $y = comp(a)$.
- for all $x \neq comp(a)$, $q'(x) = q(x)$.

A run of $\mathcal{A}$ on a word $\sigma \in \Delta^\omega$ is a map $\rho : prf(\sigma) \rightarrow Q_X$ such that

- $\rho(\epsilon) \in Q_{in}$ and
- for each prefix τa of σ we have $\rho(\tau) \xrightarrow{a}_{\mathcal{A}} \rho(\tau a)$.

Let $\sigma \restriction x$ denote the projection of the word σ to the subset C_x of Δ. Then we say a run ρ on σ is accepting iff for each $x \in X$, either $\sigma \restriction x$ is finite and $\rho(\tau)(x) \in F_x$ for some prefix τ of σ such that $\tau \restriction x = \sigma \restriction x$; or $\sigma \restriction x$ is infinite and $\rho(\tau)(x) \in G_x$ for infinitely many prefixes τ of σ.

We set $L_{sym}(\mathcal{A})$ to be the set of words in Δ^ω on which $\mathcal{A}$ has an accepting run.

Lemma 7.3 *Let $\widetilde{\Delta}$ be a $\mathcal{P}$-distributed alphabet, and let $\mathcal{C} = \{C_x\}_{x \in X}$ be a valid partition of Δ with respect to $\widetilde{\Delta}$. Let $L \subseteq \Delta^\omega$. Then $L \in$ RCL$(\widetilde{\Delta})$ iff $L = L_{sym}(\mathcal{B})$ for some* qACA *$\mathcal{B}$ over $(\widetilde{\Delta}, \mathcal{C})$.*

Proof Let $L \in$ RCL$(\widetilde{\Delta})$. Then we know there exists a distributed automaton $\mathcal{A}$ over $\widetilde{\Delta}$ such that $L = L_{sym}(\mathcal{A})$. Let

$$\mathcal{A} = (\{S_i\}_{i \in \mathcal{P}}, \{\longrightarrow_a\}_{a \in \Delta}, S_{in}, \{F_i\}_{i \in \mathcal{P}}, \{G_i\}_{i \in \mathcal{P}}).$$

We will construct a qACA $\mathcal{B}$ over $(\widetilde{\Delta}, \mathcal{C})$ which will simulate the behaviour of $\mathcal{A}$.

The idea is that each component x stores the states of each agent in $loc(x)$, as last seen by it. In addition, for each $i \in loc(x)$, it also stores information to keep track of the component (among those in $comp(i)$) with the latest information about agent i. This construction is adapted from a standard construction to show that cellular asynchronous automata can simulate asynchronous automata (see [M94]).

Rather than spell out the construction as a whole, it will be more convenient for the reader if we sketch the construction in a layered fashion. We begin with the underlying structure of $\mathcal{B}$.

The states of $\mathcal{B}$ will be given by $\{Q_x\}_{x \in X}$ where for each $x \in X$, $Q_x = S_{loc(x)} \times \{0, \ldots, |X|-1\}^{loc(x)}$. Thus each Q_x state is of the form (s, v) where s is a vector of states comprising an S_i state for each $i \in loc(x)$. For each $i \in loc(x)$, v contains a value between 0 and $|X|-1$. This value when added to the values in other $comp(i)$ components, will give us the component with the latest information of agent i's state. (We assume, without loss of generality, that $X = \{0, 1, \ldots, |X|-1\}$.) This is formalised in the transition relation described below.

Let $a \in \Delta$ and let $x = comp(a)$. Let $q \in Q_{D(x)}$, and let $q' \in Q_x$. For each $y \in D(x)$ let $q(y) = (r_y, v_y)$. Further, let $q' = (r, v)$. Then we have $q \longrightarrow'_a q'$ iff there exists a transition $s \longrightarrow_a s'$ in $\mathcal{A}$ such that for each $i \in loc(a)$

- $q(z)(i) = s(i)$, where z is the component in $comp(i)$ with the most recent information about agent i. That is, $z = (\sum_{y \in comp(i)} v_y(i)) \; mod \; |X|$.
- $r(i) = s'(i)$.
- $v(i)$ is such that $((\sum_{y \in (comp(i) - \{x\})} v_y(i)) + v(i)) \; mod \; |X| = x$.

The set of initial states Q_{in} of $\mathcal{B}$ will be given as follows. Let $q \in Q_X$, with $q(x) = (r_x, v_x)$ for each $x \in X$. Then $q \in Q_{in}$ iff

1. there exists a $s \in S_{in}$ such that for each $i \in \mathcal{P}$ and each $x \in comp(i)$, $r_x(i) = s(i)$.
2. $\sum_{x \in comp(i)} v_x(i) \in comp(i)$.

To give an appropriate acceptance condition for $\mathcal{B}$ we will have to carry additional information in the basic structure defined above.

To begin with, the automaton $\mathcal{B}$ guesses the following:

- A subset *fin* of X which is the set of components of $\mathcal{B}$ participating only finitely often in the word.

- For each $x \in \mathit{fin}$, a subset $last(x)$ of $loc(x)$ which is the set of agents in $loc(x)$ for which the last action performed is a C_x action.

This guessing is implemented using the non-deterministic choice available through multiple initial states.

Now, each component $x \notin \mathit{fin}$ maintains, for each agent $i \in loc(x)$, a pair (f, W) with $f \in \{0, 1\}$ and $W \subseteq comp(i)$. The pair (f, W) is used to indicate whether agent i has seen a G_i state ($f = 1$) and the components W in $comp(i)$ which have "witnessed" this fact. Initially, $f = 0$ and $W = \emptyset$ for all agents. Suppose component x performs an action in Δ_i. Let (f, W) be the pair associated with agent i in the component with the latest information about agent i. Then if $W = comp(i)$, f is set to 0 and W is set to $\emptyset$. Else, if agent i moves into a state in G_i, f is set to 1 and W to $W \cup \{x\}$. Otherwise, f is set to 0 and W remains unchanged.

The pairs (f, W) in which $x = 1$ will be used by each component to signal its infinitary final states. As is done for generalised Büchi conditions, each component x signals an infinitary final state only after it checks that for each agent in $loc(x)$, the associated pair (f, W) is such that $f = 1$.

For the local finitary final states, a component $x \in \mathit{fin}$ non-deterministically decides that it has done its last action in the current word and moves into a state which indicates so. For each $i \in loc(x)$, the corresponding state is marked "good" if either $i \notin last(x)$ or agent i has moved into a state in its local finitary final state set F_i. Otherwise the corresponding state is marked "bad." The finitary final states for component x are now all those states in which the corresponding state for each agent in $loc(x)$ is marked "good."

This completes the description of the qACA $\mathcal{B}$ which accepts the same language as $\mathcal{A}$.

Conversely, let $\mathcal{B}$ be a qACA over $(\widetilde{\Delta}, \mathcal{C})$. We will show that we can simulate $\mathcal{B}$ using a distributed automaton $\mathcal{A}$. This will show that $L_{sym}(\mathcal{B}) \in \mathrm{RCL}(\widetilde{\Delta})$.

Let $\mathcal{B} = (\{Q_x\}_{x\in X}, \{\longrightarrow'_a\}_{a\in\Delta}, Q_{in}, \{F_x\}_{x\in X}, \{G_x\}_{x\in X})$. The distributed automaton $\mathcal{A}$ will simulate $\mathcal{B}$ by keeping track of, for each agent $i \in \mathcal{P}$, the state of each of the components in $comp(i)$. In fact after every move of $\mathcal{A}$, the following invariant will be maintained. Suppose x is the component of $\mathcal{B}$ which has made the last move. Then for each $x \in X$, each agent in $loc(x)$ will contain the current state of component x.

Thus the states of $\mathcal{B}$ will be $\{S_i\}_{i\in\mathcal{P}}$ where for each $i \in \mathcal{P}$, $S_i = Q_{comp(i)}$.

The transition relation $\longrightarrow_a$ for $\mathcal{A}$ will be given as follows. Let $a \in \Delta$ and let $a \in C_x$. Let $y \in D(x)$. Then we know that there exists an $i \in \mathcal{P}$ such that $C_x, C_y \subseteq \Delta_i$, and hence $x, y \in comp(i)$. Let s, s' be S_a states. Then we define $s \longrightarrow_a s'$ iff there exists a transition in $\mathcal{B}$ of the form $q \longrightarrow'_a q'$ such that

1. for each $z \in D(x)$, and $i \in loc(x)$ such that $z \in comp(i)$, we have $s(i)(z) = q(z)$.
2. for each $i \in loc(a)$ $s'(i)(x) = q'$ and $s'(i)(x') = s(i)(x')$ for each $x' \neq x$.

The set of initial states S_{in} is the set of states $s \in S_{\mathcal{P}}$ such that there exists a state $q \in Q_{in}$ with $s(i)(x) = q(x)$ for each $i \in \mathcal{P}$ and each $x \in comp(i)$.

For setting up the final states, we will assume that each component x has been modified so that it uses only "terminating" finitary final states—that is, once in a finitary final state, the component never makes a further transition. The finitary final states for agent i will now comprise the states $s \in S_i$ such that $s(x) \in F_x$ for each $x \in comp(i)$. For the infinitary final states we again use a counter which cycles through each component in $comp(i)$ checking that either it is in a finitary final state, or moves into an infinitary final state in G_x at some point. The states with counter value 0 will then be the infinitary final states for each agent.

It is not difficult to verify that $L_{sym}(\mathcal{A}) = L_{sym}(\mathcal{B})$. This completes the proof of the lemma. $\Box$

7.2 Cellular interval automata

Once again, let us fix a set of agents $\mathcal{P} = \{1, \ldots, k\}$ and a $\mathcal{P}$-distributed alphabet $\widetilde{\Sigma}$. For $a \in \Sigma$ let $D(a)$ denote the set of letters in Σ which are dependent on a. That is, $D(a) = \{b \in \Sigma \mid aD_{\widetilde{\Sigma}}b\}$. Then a *cellular interval alphabet* based on $\widetilde{\Sigma}$ is a finite set of elements of the form (a, K) where $a \in \Sigma$ and $K : D(a) \to \mathcal{I}\mathbb{R}$.

Let Γ be a cellular interval alphabet based on $\widetilde{\Sigma}$. As before we can define a map tw which associates with each $\widehat{\sigma} \in \Gamma^\infty$ a set of timed words $tw(\widehat{\sigma}) \subseteq T\widetilde{\Sigma}^\infty$. This is done as follows. Let $\sigma \in T\widetilde{\Sigma}^\infty$ and let $\widehat{\sigma} \in \Gamma^\infty$. Then $\sigma \in tw(\widehat{\sigma})$ iff $|\sigma| = |\widehat{\sigma}|$ and for each prefix $\tau(a,t)$ of σ and $\widehat{\tau}(b,K)$ of $\widehat{\sigma}$ with $|\tau| = |\widehat{\tau}|$ we have $a = b$ and for each $c \in D(a)$, $t - time_c(\tau) \in K(c)$. Here $time_c(\tau)$ is the time of occurrence of the latest c action in τ: it is defined inductively as follows. $time_c(\epsilon) = 0$ and $time_c(\tau'(a,t)) = t$ if $a = c$ and $time_c(\tau')$ otherwise. We extend the map tw to work on subsets of Γ^∞ in the natural way.

Before we define a cellular interval automaton, we note that a cellular interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$ induces in a natural way a valid partition of Γ. This partition is given by $\mathcal{C}_{\widetilde{\Gamma}} = \{C_a\}_{a \in \Sigma}$ with $C_a = \{(b, K) \in \Gamma \mid b = a\}$.

Definition 7.4 *A cellular interval automaton over* $\widetilde{\Sigma}$ *is a* qACA *over* $(\widetilde{\Gamma}, \mathcal{C}_{\widetilde{\Gamma}})$, *for some cellular interval alphabet* $\widetilde{\Gamma}$ *based on* $\widetilde{\Sigma}$.

A cellular interval automaton $\mathcal{A}$ over $\widetilde{\Sigma}$ accepts a timed language $L(\mathcal{A}) \subseteq T\widetilde{\Sigma}^{\omega}$ given by $L(\mathcal{A}) = tw(L_{sym}(\mathcal{A}))$. We will call such languages *ω-regular cellular interval languages* over $\widetilde{\Sigma}$ and denote the class of such languages by $\mathrm{CIL}(\widetilde{\Sigma})$.

Example 7.5 Let $\widetilde{\Sigma} = (\{a, b\})$. We assume the ordering $a < b$ and represent functions (over elements of Σ) as as tuples $\langle I, I' \rangle$, (s, s') etc. Let $\widetilde{\Gamma}$ be the cellular interval alphabet $(\{(a, \langle I, I \rangle), (b, \langle I, I \rangle), (b, \langle I, I' \rangle)\})$ where $I = (0, \infty)$ and $I' = (1, \infty)$. Here $X = \Sigma = \{a, b\}$ and $\mathcal{C}_{\widetilde{\Gamma}}$ is the partition of Γ given by $C_a = \{(a, \langle I, I \rangle)\}$ and $C_b = \{(b, \langle I, I \rangle), (b, \langle I, I' \rangle)\}$. Define

$$\mathcal{A} = (\{Q_x\}_{x \in \Sigma}, \{\longrightarrow_{\widehat{a}}\}_{\widehat{a} \in \Gamma}, Q_{in}, \{F_x\}_{x \in \Sigma}, \{G_x\}_{x \in \Sigma}) \text{ where}$$

- $Q_a = \{s_1, s_2\}$, and $Q_b = \{s_1, s_2, s_3\}$
- The transition relations are given by:

$$\begin{aligned} (s_1, s_1) &\longrightarrow_{(b,I,I)} s_2, \\ (s_1, s_2) &\longrightarrow_{(a,I,I)} s_2, \\ (s_2, s_2) &\longrightarrow_{(b,I,I')} s_3, \\ (s_2, s_3) &\longrightarrow_{(b,I,I)} s_3. \end{aligned}$$

- $Q_{in} = \{(s_1, s_1)\}$.
- $F_a = \{s_2\}$, $F_b = \emptyset$, $G_a = \emptyset$, and $G_b = \{s_3\}$.

$L(\mathcal{A})$ can be seen to be essentially the language L of Lemma 2.10, extended in a trivial way to give us an ω-language. Thus

$$L(\mathcal{A}) = \{(b, t)(a, t')(b, t'')(b, t_1)(b, t_2) \cdots \in T\Sigma^{\omega} \mid t'' - t > 1\}$$

□

Cellular interval automata can be simulated by timed Büchi automata in much the same way as product and distributed interval automata. Here we will need to have a clock for each letter of Σ.

Let $\mathcal{B} = (\{Q_a\}_{a \in \Sigma}, \{\longrightarrow'_a\}_{a \in \Gamma}, Q_{in}, \{F_a\}_{a \in \Sigma}, \{G_a\}_{a \in \Sigma})$ be a cellular interval automaton over Σ, via the cellular interval alphabet $\widetilde{\Gamma}$. The time complexity for checking emptiness of $L(\mathcal{B})$ by this method can be seen to be

$$O(|S| + |E|) \cdot 2^{O(|\Sigma|)} \cdot |\Sigma|! \cdot \prod_{a \in \Sigma} (c_a + 1)$$

where S and E are the global states and transitions of $\mathcal{B}$ respectively. For each $a \in \Sigma$, c_a is the largest (normalised) interval bound in the set

$$\{K(a) \mid b \in D(a) \text{ and } (b, K) \in \Gamma\}.$$

Cellular interval languages can be seen to be strictly more expressive than distributed interval languages.

Proposition 7.6 *The class of ω-regular distributed interval languages over a distributed alphabet $\widetilde{\Sigma}$ is a subclass of the class of ω-regular cellular interval languages over $\widetilde{\Sigma}$.*

Proof Let $\mathcal{A}$ be a distributed interval automaton over $\widetilde{\Sigma}$. Then $\mathcal{A}$ is also a distributed automaton over a distributed interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$. Consider the partition $\mathcal{C}$ of Γ given by $\mathcal{C} = \{C_a\}_{a \in \Sigma}$, where $C_a = \{(b, J) \in \Gamma \mid b = a\}$. Then $\mathcal{C}$ is valid with respect to $\widetilde{\Gamma}$. By Lemma 7.3 we can construct a a qACA $\mathcal{B}$ over $(\widetilde{\Gamma}, \mathcal{C})$ such that $L_{sym}(\mathcal{B}) = L_{sym}(\mathcal{A})$.

We now transform $\mathcal{B}$ to a qACA $\mathcal{B}'$ over a cellular interval alphabet $\widetilde{\Gamma}'$ based on $\widetilde{\Sigma}$. Let $\widetilde{\Gamma}'$ be the set of all (a, K) with $a \in \Sigma$ and $K : D(a) \to \mathcal{I}\mathbb{R}$ such that $K(b)$ is the intersection of a subset of intervals mentioned in Γ. We define the qACA $\mathcal{B}'$ over $(\widetilde{\Gamma}', \mathcal{C}_{\widetilde{\Gamma}'})$ as follows. The automaton $\mathcal{B}'$ is essentially the same as $\mathcal{B}$, except that the transition relation of $\mathcal{B}'$ is obtained by replacing each edge $q \longrightarrow_{(a,J)} q'$ of $\mathcal{A}$ by the edge $q \longrightarrow'_{(a,K)} q'$, provided for each $b \in D(a)$, $K(b) = \bigcap_{i \in P} J(i)$, where $P \subseteq loc(a)$ is the set of agents for which $q(b)$ has the latest information.

We are making use of the fact that $time_i(\tau)$ equals $time_b(\tau)$ provided b was the last Σ_i action to take part in τ. It is not difficult to see that this property ensures that the timed language accepted by $\mathcal{B}'$ is the same as that of $\mathcal{A}$.

To show that (over certain distributed alphabets) cellular interval automata are strictly more powerful than distributed interval automata, consider the timed language L accepted by the cellular interval automaton $\mathcal{A}$ of Example 7.5. $\mathcal{A}$ is defined over the single-component distributed alphabet $\widetilde{\Sigma} = (\{a, b\})$. Suppose that L were accepted by a distributed interval automaton over $\widetilde{\Sigma}$. Since $\widetilde{\Sigma}$ is a single-component distributed alphabet, this would imply that L is also a regular interval language over Σ. But, by essentially the same argument as Lemma 2.10, we know that L is not a regular interval language over Σ, and this is a contradiction. Thus L is not a ω-regular distributed interval language over $\widetilde{\Sigma}$. □

7.3 Closure and logical properties

In the light of results proved in the preceding section, particularly Lemma 7.3, cellular interval languages are simply images under the *tw* map of ω-regular consistent languages over a cellular interval alphabet. Using the notion of a *proper* cellular interval alphabet—defined in the obvious way—and using the fact that ω-regular consistent languages are closed under boolean operations, we can show that the class of cellular interval languages over a distributed alphabet $\widetilde{\Sigma}$ are also closed under boolean operations.

Theorem 7.7 *The class of* $\mathrm{CIL}(\widetilde{\Sigma})$ *of ω-regular cellular interval languages over $\widetilde{\Sigma}$ is closed under the boolean operations of union, intersection and complementation.* $\square$

In Section 6.5 we showed that ω-regular distributed interval languages can be represented by ω-regular timed trace languages. In a similar manner we can show that ω-regular cellular interval languages can be represented by ω-regular *cellular* timed trace languages. These are defined via the map (called *tt* as well) which is given as follows. Let $T = (T, \leq, \lambda)$ be a trace over a cellular interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$. Let (T', θ') be a timed trace over $\widetilde{\Sigma}$, with $T' = (E', \leq', \lambda')$. For an element $e \in E'$ we define $time_a((T', \theta'), e)$ to be the time of the last a-labelled event below e in the timed trace (T', θ'). Then $(T', \theta') \in tt(T)$ iff

- $(E', \leq') = (E, \leq)$ and for each $e \in E$, $\lambda'(e) = a$ iff $\lambda(e) = (a, K)$ for some K. That is, λ' is the projection of λ to the Σ-component of the alphabet Γ.
- Let $e \in E$ with $\lambda(e) = (a, K)$. Then for each $b \in D(a)$ $(\theta'(e) - time_b((T', \theta'), e)) \in K(b)$.

As before, we extend *tt* to act on subsets of $\mathrm{Tr}(\widetilde{\Gamma})$ as well.

The class $\mathrm{RTCTrL}(\widetilde{\widetilde{\Sigma}})$ of ω-regular timed cellular trace languages over $\widetilde{\widetilde{\Sigma}}$ is defined to be the set of all timed trace languages $M \subseteq \mathrm{TTr}(\widetilde{\Sigma})$ such that there exists a cellular interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$ and $\widehat{M} \in \mathrm{RTrL}(\widetilde{\Gamma})$ such that $M = tt(\widehat{M})$.

We now give a logical characterisation of ω-regular timed cellular trace languages over $\widetilde{\Sigma}$ in terms of a logic interpreted over timed traces, as done in Section 6.6.

The syntax of this logic, which we call $\mathrm{TMSOC}(\widetilde{\Sigma})$, is given by:

$$\varphi ::= (x \leq y) \mid Q_a(x) \mid \Delta_a(x, I) \mid (x \in X) \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x \varphi \mid \exists X \varphi.$$

In the formula $\Delta_a(x, I)$ we require $a \in \Sigma$ and $I \in \mathcal{IR}$.

The semantics of the logic are similar to that of TMSOTr$(\widetilde{\Sigma})$ given in Section 6.6. The operator $\Delta_a(x, I)$ is new however, and it is interpreted as follows. Let $\mathbb{I}$ be an interpretation for variables. Then:

$$(T, \theta \models_{\mathbb{I}} \Delta_a(x, I) \text{ iff } \lambda(\mathbb{I}(x)) \in D(a) \text{ and } \theta(\mathbb{I}(x)) - \mathit{time}_a((T, \theta), \mathbb{I}(x)) \in I.$$

Using essentially the same arguments as for TMSOTr$(\widetilde{\Sigma})$ we can show the following result.

Theorem 7.8 *Let* $M \subseteq \mathrm{TTr}(\widetilde{\Sigma})$*. Then* $M \in \mathrm{RTCTrL}(\widetilde{\Sigma})$ *iff* $M = L(\varphi)$ *for some sentence* $\varphi \in \mathrm{TMSOC}(\widetilde{\Sigma})$*.* □

7.4 Event recording automata

We conclude this chapter by showing how cellular interval automata are related to the event recording automata of Alur, Fix and Henzinger. In addition we apply the methods introduced earlier in the thesis to establish a logical characterisation of languages accepted by event recording automata.

Event recording automata were proposed in [AFH94] as a determinizable (and hence complementable) subclass of timed automata. They are timed Büchi automata which have a clock x_a for each action a, with the restriction that along every transition labelled a, the set of clocks reset is exactly $\{x_a\}$.

We can define an event recording automaton in the same spirit as the automata introduced thus far. Let Σ be an alphabet of actions. Then an *event-recording alphabet* based on Σ is a finite subset of $\Sigma \times \mathcal{IR}^{\Sigma}$. Thus elements of the event-recording alphabet are of the form (a, K), where $K : \Sigma \to \mathcal{IR}$.

Let Γ be an event recording alphabet based on Σ, and let $\widehat{\sigma} \in \Gamma^{\infty}$. Then, we can associate a set of timed words $tw(\widehat{\sigma})$ with $\widehat{\sigma}$ as follows: Let $\sigma \in T\Sigma^{\infty}$. Then $\sigma \in tw(\widehat{\sigma})$ iff $|\sigma| = |\widehat{\sigma}|$ and for $\tau(a, t) \in \mathit{prf}(\sigma)$ and $\widehat{\tau}(b, K) \in \mathit{prf}(\widehat{\sigma})$ such that $|\tau| = |\widehat{\tau}|$ we have $a = b$, and $t - \mathit{time}_c(\tau) \in K(c)$ for all $c \in \Sigma$. As before, $\mathit{time}_c(\tau)$ is the time of occurrence of the latest c action in τ. For a subset $\widehat{L}$ of Γ^{∞}, we set $tw(\widehat{L}) = \bigcup_{\widehat{\sigma} \in \widehat{L}} tw(\widehat{\sigma})$.

An event recording automaton $\mathcal{A}$ over Σ is a Büchi automaton over an event recording alphabet based on Σ. The language of timed words accepted by $\mathcal{A}$ is given by $L(\mathcal{A}) = tw(L_{sym}(\mathcal{A}))$. We term such languages as *event recording languages* over Σ.

Event recording automata and cellular interval automata have the same expressive power in the sense of Proposition 7.9 below. As in the case of timed automata, we will need to relax the definition of the timed language accepted by event recording automata to allow them to accept non-decreasing timed words. This is to facilitate a comparison with cellular interval languages in which simultaneous occurrences of independent actions are allowed. As done in Section 3.3 we set $L_{nd}(\mathcal{A})$ to be the language of non-decreasing timed words accepted by an event recording automaton $\mathcal{A}$. Also, for convenience, we will consider only ω-behaviours for these automata.

Proposition 7.9 *1. Let $\mathcal{A}$ be an event recording automaton over an alphabet Σ. Then we can construct a cellular interval automaton $\mathcal{B}$ over the single-component distributed alphabet $\widetilde{\Sigma} = (\Sigma)$ such that $L(\mathcal{B}) = L(\mathcal{A})$.*

2. Let $\mathcal{B}$ be a cellular interval automaton over a distributed alphabet $\widetilde{\Sigma}$. Then we can construct an event recording automaton $\mathcal{A}''$ over the alphabet Σ such that $L_{nd}(\mathcal{A}'') = L(\mathcal{B})$.

Proof

1. We know that $\mathcal{A}$ is a Büchi automaton over a event recording alphabet Γ based on Σ and $L(\mathcal{A}) = tw(L_{sym}(\mathcal{A}))$. Now $L_{sym}(\mathcal{A})$ is an ω-regular consistent language over $\widetilde{\Gamma} = (\Gamma)$. Thus there exists a qACA $\mathcal{B}$ over $(\widetilde{\Gamma}, \mathcal{C}_{\widetilde{\Gamma}})$ such that $L_{sym}(\mathcal{B}) = L_{sym}(\mathcal{A})$ (by Lemma 7.3). Further, $\widetilde{\Gamma}$ is also a cellular interval alphabet over $\widetilde{\Sigma}$. Thus $\mathcal{B}$ is a cellular interval automaton over $\widetilde{\Sigma}$. Finally, we observe that the definitions of the tw map for symbolic words over an event recording alphabet and a cellular interval alphabet coincide for a single-component distributed alphabet. It is now easy to see that $L(\mathcal{B}) = tw(L_{sym}(\mathcal{B})) = tw(L_{sym}(\mathcal{A})) = L(\mathcal{A})$.

2. Let $\mathcal{B}$ be a cellular interval automaton over $\widetilde{\Sigma}$ as in the statement of the proposition. Then $\mathcal{B}$ is a qACA over $(\widetilde{\Gamma}, \mathcal{C}_{\widetilde{\Gamma}})$, for some cellular interval alphabet $\widetilde{\Gamma}$ based on $\widetilde{\Sigma}$.

   Let $\mathcal{B} = (\{Q_a\}_{a\in\Sigma}, \{\longrightarrow_{\widehat{a}}\}_{\widehat{a}\in\Gamma}, Q_{in}, \{F_a\}_{a\in\Sigma}, \{G_a\}_{a\in\Sigma})$. We assume here that each local finitary final state in $\mathcal{B}$ is terminal.

   We define an event recording automaton $\mathcal{A}$ over Σ with the property that $L_{nd}(\mathcal{A}) \cap T\widetilde{\Sigma}^{\infty} = L(\mathcal{B})$. Let Γ' be the event recording alphabet given by

   $$\begin{aligned}\Gamma' \;=\; \{(a, K') \mid \; & \exists (a, K) \in \Gamma : \forall b \in \Sigma,\; K'(b) = K(b) \text{ if } b \in D(a), \\ & \text{and } (0, \infty) \text{ otherwise}\}.\end{aligned}$$

   Consider the event recording automaton $\mathcal{A} = (S, \longrightarrow, S_{in}, F, G)$ over Γ', where:

- The set of states is $S = Q_\Sigma \times \{0, \ldots, |\Sigma|\}$.
- The transition relation is given as follows. We have

$$(q, l) \xrightarrow{(a,K')} (q', m)$$

iff the following conditions hold:

(i) there exists $(a, K) \in \Gamma$ such that $q \xrightarrow{(a,K)}_{\mathcal{B}} q'$, and $K'(b) = K(b)$ for each $b \in D(a)$ and $K'(b) = (0, \infty)$ if $b \notin D(a)$.

(ii) $m = (l+1) \bmod (|\Sigma| + 1)$ if $q'(l) \in F_l \cup G_l$ or $l = 0$; otherwise $m = l$.

- The initial states are given by $S_{in} = Q_{in} \times \{0\}$.
- $F = \emptyset$
- The infinitary final states are given by $G = \{(q, l) \in S \mid l = 0\}$.

It is easy to verify that $L_{nd}(\mathcal{A}) \cap T\widetilde{\Sigma}^\infty = L(\mathcal{B})$.

Next we define an event recording automaton $\mathcal{A}'$ over Σ which accepts the non-decreasing timed language $T\widetilde{\Sigma}^\omega$. This is done in a similar manner to the timed automaton $\mathcal{B}'$ of Section 3.3, except that here we must keep track of the last action and replace the transition label $(a, x, (x \in I))$ by (a, K) where for each $b \in \Sigma$, $K(b) = I$ if b is the last action to have taken place; otherwise $K(b) = (0, \infty)$.

We can now intersect $\mathcal{A}$ with the automaton $\mathcal{A}'$ to get the event recording automaton $\mathcal{A}''$ with the property that $L_{nd}(\mathcal{A}'') = L(\mathcal{B})$.

□

Using the techniques used in the previous chapters we can show the result established in [AFH94] that event recording languages are closed under boolean operations. In addition we can give the following logical characterisation of these languages.

Consider the logic TMSOER(Σ) given by

$$\varphi ::= (x \in X) \mid (x < y) \mid Q_a(x) \mid \Delta_a(x, I) \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x\varphi \mid \exists X\varphi.$$

The logic is interpreted over timed words in $T\Sigma^\omega$ and the semantics is similar to that of TMSO(Σ) introduced in Section 2.5. The operator $\Delta_a(x, I)$ is interpreted as follows:

$$\sigma \models_{\mathbb{I}} \Delta_a(x, I) \text{ iff } time(\mathbb{I}(x)) - time_a(\mathbb{I}(x)) \in I.$$

Thus the predicate $\Delta_a(x, I)$ allows us to check if the time since the last occurrence of a lies in the interval I.

Theorem 7.10 *Let Σ be an alphabet of actions and let $L \subseteq T\Sigma^\omega$. Then L is an event recording language over Σ iff there exists a sentence $\varphi \in$* TMSOER(Σ) *such that $L = L(\varphi)$.*

Proof The proof proceeds in a very similar manner to the proof of Theorem 2.15, and the details are omitted here. $\square$

Chapter 8

Conclusion

In this chapter we summarise the main contents of this thesis. We also point to some areas which merit further study.

8.1 Summary

In this thesis we have proposed some subclasses of Alur-Dill timed automata and developed their theory. A common feature of our automata is that they comprise a network of timed agents in which the use of clocks is determined by the topology of the network.

The simplest of these classes of automata are called product interval automata. They comprise a network of components, each of which is a timed automaton with a single clock. The clock for each component is required to be reset with every action of the component. The classes of distributed interval automata and cellular interval automata are generalisations of product interval automata, based on classical extensions of products of automata. The relative expressive power of these automata is depicted in Fig 8.1.

We show that these classes of automata enjoy several properties which are desirable from both a practical as well as theoretical viewpoint.

Each of these classes is closed under the boolean operations of union, intersection and complementation. In addition they admit logical characterisations via full monadic second order logics.

In terms of the modelling power we show that these automata are powerful enough to model an important class of timed behaviours, namely those of asynchronous digital circuits.

Product interval automata and distributed interval automata have some useful preservation properties in the context of partial order reduction methods for timed systems. Their disciplined use of clocks can be exploited to

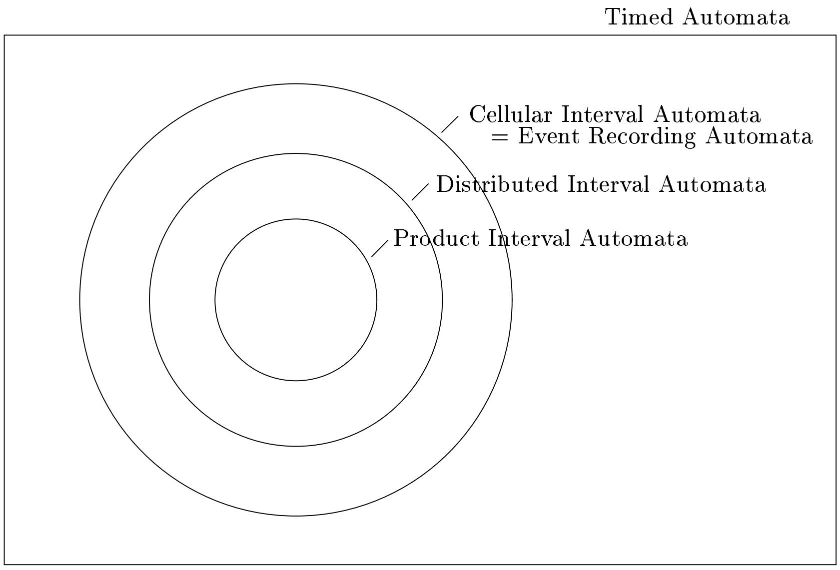

Figure 8.1: Subclasses of timed automata studied in this thesis

efficiently infer the independence of transitions in the quotiented state space [M99]. We also demonstrate a property of the languages accepted by these automata which leads to a saving in the emptiness check for these automata compared to that of general timed automata.

In the case of product interval languages we study a timed temporal logic called $\mathrm{TLTL}^{\otimes}$. This logic is expressively equivalent to the first order fragment of $\mathrm{TMSO}^{\otimes}$, the monadic second order logic corresponding to product interval languages. We give automata theoretic solutions to the satisfiability and model checking problems for this logic.

The closure properties of our automata can be attributed to the fact that all clocks are *implicit*—the action labels on the transitions determine the set of clocks to be reset. In conjunction with this property the use of proper interval alphabets constitutes the main technical idea. Proper alphabets help us to represent timed languages symbolically in a way in which operations on timed languages can factor through corresponding operations at the symbolic level. This allows us to make use of powerful results in classical automata theory and logic.

8.2 Future Work

An issue which needs further examination is whether product interval automata are amenable to a more efficient emptiness check. At the moment our method of checking emptiness is to treat these automata as general timed automata and then apply the region (or zone) construction. As shown in Section 6.3 product interval automata do have a minor advantage over general timed automata in that progressive run conditions can be done away with, leading to a minor saving in the size of the region automaton. However, one needs to explore the possibility that product interval automata may admit a more efficient emptiness check which takes advantage of the the structure of these automata.

Another area which we aim to examine further is the use of our automata in partial order reduction methods for timed systems.

In the untimed setting, the unfolding method of McMillan [M93] constitutes an alternate approach to the partial order based reduction methods of [V92, G96]. Here one constructs a finite prefix of the unfolding of a transition system (given by say a Petri net or product system) from which it is possible to retrieve information about reachability and deadlock in the corresponding global transition system. This often leads to a saving in the work done as retrieving this information from the the finite prefix could be more efficient than working with the global transition system.

We would like to explore the applicability of this method in a timed setting. Using the local time semantics of product interval automata it may be possible to define an appropriate notion of a timed unfolding. We expect such a method to help in the practical verification of asynchronous circuits.

Bibliography

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

[AFH94] R. Alur, L. Fix, T. A. Henzinger: Event-clock automata: a determinizable class of timed automata, *Proc. 6th International Conference on Computer-aided Verification, LNCS 818,* 1–13, Springer-Verlag (1994).

[AFH96] R. Alur, T. Feder, T. A. Henzinger: The benefits of relaxing punctuality, *Journal of the ACM, 43,* 116–146 (1996).

[AH92] R. Alur, T. A. Henzinger: Logics and Models of Real Time: A Survey, in *Real-Time: Theory in Practice,* J. W. de Bakker, H. Huizing, W. -P. de Roever, G. Rozenberg (Eds.), LNCS 600, 74–106, (1992).

[AH93] R. Alur, T. A. Henzinger: Real-time logics: complexity and expressiveness, *Information and Computation 104,* 35–77 (1993).

[B60] J. R. Büchi: Weak second-order arithmetic and finite automata, *Zeitschrift für Math. Logik und Grundlagen der Mathematik, 6,* 66–92 (1960).

[BJ+98] J. Bengtsson, B. Jonsson, J. Lilius, W Yi: Partial Order Reductions for Timed Systems, *Proc. CONCUR '98, LNCS 1466* (1998).

[BM83] B. Berthomieu, M. Menasche: An Enumerative Approach for Analyzing Time Petri Nets, *Information Processing 83* North Holland (1983).

[BS94] J. A. Brzozowski, C-J. H. Seger: *Asynchronous Circuits,* Springer-Verlag (1994).

[DH99] D. D'Souza, T. Hune: An on-the-fly Construction for an Event Clock Logic, Manuscript (to appear in *BRICS Technical Report Series*) (2000).

[DM94] V. Diekert, A. Muscholl: Deterministic asynchronous automata for infinite traces, *Acta Informatica* 31, 379–397 (1994).

[DR95] V. Diekert, G. Rozenberg: *The Book of Traces*, World Scientific, Singapore (1995).

[DT98] D. D'Souza, P. S. Thiagarajan: Distributed Interval Automata, Internal Report TCS-98-3, Chennai Mathematical Institute (1998).

[DT99] D. D'Souza, P. S. Thiagarajan: Product Interval Automata: A Subclass of Timed Automata, *Proc. 19th Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS 1732* (1999).

[EM96] W. Ebinger, A. Muscholl: Logical definability on infinite traces, *Theoretical Computer Science* 154: 67–84 (1996).

[G96] P. Godefroid: Partial-Order Methods for the Verification of Concurrent Systems, *LNCS 1032* (1996).

[GP92] P. Gastin, A. Petit: Asynchronous cellular automaton for infinite traces, *Proceedings of ICALP '92, LNCS 623*, 583–594 (1992).

[GP+80] D. Gabbay, A. Pnueli, S. Shelah, J. Stavi: The Temporal Analysis of Fairness, *Seventh ACM Symposium on Principles of Programming Languages,* 163–173, (1980).

[HRS98] T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens: The regular real-time languages, *Proc. 25th International Colloquium on Automata, Languages, and Programming 1998, LNCS 1443,* 580–591 (1998).

[K68] K. Kamp: Tense Logic and the Theory of Linear Order, PhD Thesis, University of California (1968).

[KMS95] N. Klarlund, M. Mukund, M. Sohoni: Determinizing Büchi Asynchronous Automata, *Proceedings of FSTTCS 15, LNCS 1026,* 456–470 (1995).

[KP92] S. Katz, D. Peled: Interleaving Set Temporal Logic, *Theoretical Computer Science, Vol. 75 (3),* 21–43 (1992).

[M93] K. McMillan: *Symbolic model checking*, Kluwer Academic Publishers (1996).

[M94] A. Muscholl: On the complementation of Büchi asynchronous cellular automata, *Theoretical Computer Science, 169,* 123–145 (1996).

[M99] M. Minea: Partial Order Reduction for Model Checking of Timed Automata. To appear in the Proceedings of CONCUR'99.

[MF76] P. Merlin, D. J. Farber: Recoverability of Communication Protocols—Implications of a Theoretical Study, *IEEE Trans. Commun. Vol 24, No. 9* (1976).

[MP95] O. Maler, A. Pnueli: Timing Analysis of Asynchronous Circuits using Timed Automata, in *Proc. CHARME '95, LNCS 987,* 189–205 (1995).

[MT96] M. Mukund, P. S. Thiagarajan: Linear Time Temporal Logics over Mazurkiewicz Traces, *Proc. MFCS 96, LNCS 1113,* 62–92 (1996).

[P77] A. Pnueli: The Temporal Logic of programs, *Proceedings of 18th IEEE Symposium on Foundations of Computer Science,* 46–57 (1977).

[RS97] J. -F. Raskin, P. -Y. Schobbens: State-clock Logic: A Decidable Real-Time Logic, *Proc. HART '97: Hybrid and Real-Time Systems, LNCS 1201,* 33–47 (1997).

[SC85] A. P. Sistla, E. M. Clarke: The complexity of propositional linear temporal logic, *Journal of the ACM,* 32: 733–749 (1985).

[Thi95] P. S. Thiagarajan: A Trace Consistent Subset of PTL, *Proc. CONCUR '95,* LNCS 962, 438–452 (1995).

[Tho89] W. Thomas: On the logical definability of trace languages, in ASMICS Workshop on "Free Partially Commutative Monoids," Kochel (1989).

[Tho90] W. Thomas: Automata on Infinite Objects, in J. V. Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B, 133–191, Elsevier Science Publ., Amsterdam (1990).

[V92] A. Valmari: A Stubborn Attack on State Explosion, *Formal Methods in System Design*, 1: 297-322 (1992).

[VW86] M. Y. Vardi, P. Wolper: An automata-theoretic approach to automatic program verification, Proc. IEEE Symposium on Logic in Computer Science (LICS), 332–334 (1986).

[VWS83] M. Y. Vardi, P. Wolper, A. P. Sistla: Reasoning about infinite computation paths, *Proc. 24th IEEE Symposium on Foundation of Computer Science*, Tuscan, 1983, pages 185–194.

[W94] Th. Wilke: Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata, in *Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863,* 694–715 (1994).

[WN94] G. Winskel, M. Nielsen: Models for Concurrency, in S. Abramsky, D. Gabbay (Eds.) *Handbook of Logic in Computer Science, Vol. 3,* Oxford University Press (1994).

[YJ94] W. Yi, B. Jonsson: Decidability of Timed Language-Inclusion for Networks of Real-Time Communicating Sequential Processes, *Proc. FST&TCS 94, LNCS 880* (1994).

[Z87] W. Zielonka: Notes on finite asynchronous automata, *R.A.I.R.O.— Inform. Theor. Appl., Vol 21*, 99–135 (1987).

List of Publications

1. D. D'Souza and P. S. Thiagarajan:
   Product Interval Automata: A Subclass of Timed Automata, in *Proceedings of 19th Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS 1732* (1999).

2. D. D'Souza and P. S. Thiagarajan:
   Distributed Interval Automata, Internal Report TCS-98-3, Chennai Mathematical Institute (1998).
   (Available electronically at `www.smi.ernet.in/techreps`.)

3. D. D'Souza and P. Madhusudan:
   On-the-fly verification for Product-LTL, in *Proceedings of Seventh National Seminar in Theoretical Computer Science* (1997).