Partial Order Reduction for Verification of Timed Systems

Marius Minea

December 1999
CMU-CS-00-102

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee
Edmund M. Clarke, Chair
Randal E. Bryant
Jeannette M. Wing
Doron Peled, Bell Laboratories

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This research is sponsored by the Semiconductor Research Corporation (SRC) under agreements through Contract No. 99-TJ-684, the National Science Foundation (NSF) under Grant Nos. CCR-9505472 and CCR-9803774, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0071. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of SRC, NSF, DARPA, or the United States Government.

Keywords: formal verification, model checking, partial order reduction, timed systems

Abstract

This dissertation presents solutions for the application of partial order methods to the verification of timed systems, with the purpose of reducing the size of the explored state space.

Timed systems, which rely on timing information to operate correctly, pose special difficulties for automatic verification. Not only does the size of their state space grow exponentially with the number of components, as in any concurrent system, but some of the history of past transitions becomes part of the timed state. This hinders the use of partial order reduction, a technique which is applicable if different transition interleavings lead to the same state.

We have given a partial order reduction algorithm for systems described as networks of timed automata, which preserves formulas in a timed extension of linear temporal logic. The algorithm is based on a modified local-time semantics, which allows individual automata to execute independently except for synchronization transitions.

More generally, we have investigated the application of partial order reduction in a continuous-time model whose semantics is defined in terms of timed traces. We show how to separate the causal dependence of transitions from their time ordering due to concurrency and how this leads to the application of partial order reduction. As particular instances of this framework we obtain improved algorithms for timed event/level structures and time Petri nets, as well as our algorithm for timed automata.

We have evaluated the performance of our partial order reduction approach on several timed automata benchmarks. The resulting reduction in state space stems from two sources: the local-time model reduces the number of generated time regions, while the partial order techniques applied from the domain of untimed systems reduce the explored control state space.

Acknowledgements

I was fortunate and privileged to have Ed Clarke as my advisor. He shaped my path to research by guiding me with his extensive knowledge, and at the same time opened up doors so I could gain from the expertise of others. He has shown enthusiasm for my progress and support when I was struggling, and it was Ed who suggested this thesis topic, at a time when I had long been searching for one.

I owe a lot of gratitude to my thesis committee for their advice and patient reading of my thesis. Doron Peled provided me with a lot of detailed research expertise, and friendly but sternly brought me back on track when I was losing focus. Jeannette Wing gave me insightful critical advice, both in overview and in detail; meetings with her were always extremely stimulating. Randy Bryant has been a model for solid, successful research, and it was always reassuring to hear from him that I was on the right track.

This work would not have been possible without the environment in CMU's Computer Science department, with stimulating top-notch research, and at the same time an atmosphere that I can hardly imagine friendlier. Thanks to everyone, from Sharon to facilities, for doing everything in their power so we would feel well here and give our best. And thanks to the virtual community of `zephyr++`.

My introduction to partial order methods came at Bell Laboratories where I had a very productive summer working with Bob Kurshan, Vladimir Levin, Doron Peled and Hüsnü Yenigün. I was kindly invited to Uppsala by Wang Yi, where I benefitted from talks with Bengt Jonsson and Johan Bengtsson, among others. At VERIMAG Grenoble I had insightful discussions with Joseph Sifakis, Stavros Tripakis and Sergio Yovine.

Over the years, I've enjoyed working with many members of the model checking group: Somesh, Xudong, Will, Vicky, Sergey and Yuan. I've had some of the best time working with Sérgio Campos: it is with him that I started to work on real-time systems, and both work and friendship have continued after his graduation.

Thanks to all my friends who shared my life away from work during these years. To Dan, forever joyful and supportive, for everything. To Mihai and Raluca, for the wonderful musical evenings at their place, and also, together with Ciprian, Ion and Cristi, for making me feel closer to home. To my longtime officemates, Bruce and Ralph, and later C.K., for making life between the concrete windowless walls of Wean Hall not only endurable, but enjoyable. To Andrei and Dave, for the joy of climbing, and to the volleyball group for a welcome weekly respite and for friends like Darrell, Dushyanth and Edwin.

Finally, and most importantly, thanks to my parents, to whom I owe what I am, who encouraged and supported me all these years, and who accepted the hardships of being far away so that I could see this through.

Contents

Chapter 1

Introduction

1.1 Motivation

A significant part of today's computer-related systems are time-critical. They may rely on timing information to operate correctly, or their specifications may require that certain actions be executed within given time limits. Examples of such systems are timed asynchronous circuits, network or communication protocols, and industrial controllers. Failure to satisfy these properties may result in malfunction, system shutdown, significant financial costs, or even risk to humans. It is therefore imperative that correctness of these applications is guaranteed under all possible circumstances.

Moreover, the development of techniques that assist this correctness goal can stimulate the use of designs that offer increased efficiency, but whose behavior is more difficult to analyze. Such is the case, for example, with asynchronous circuits which can achieve significant performance gains while dispensing with some of the limitations of designs based on a synchronous clock.

At the same time, timed systems often lack in robustness (a small change in timing can result in a significant change in behavior), and their analysis can be very complex, since they have a large family of possible executions. The latter reason makes traditional methods such as testing and simulation even less likely to deliver exhaustive correctness results than in the case of untimed systems. In fact, with a dense view of time, the family of behaviors for a timed system can be infinite.

Formal verification techniques approach the correctness problem by proving using mathematical formalisms that a system model satisfies its specification under all possible circumstances. Within this category, model checking [CE81] has emerged as a very successful technique, with the benefit that it is completely automatic. However, its application to even larger and more complex systems is limited by the so-called *state space explosion* problem: for many types of systems, the number of possible states grows exponentially with the number of component parts. This quickly leads to models whose size exceeds the current capabilities of verification tools. For real-time systems, the space explosion problem is even more limiting, and is caused by two different factors: complexity in the control space and complexity due to the timing associated with the system.

To illustrate more precisely the causes of state space explosion in the case of timed systems, consider a typical state exploration algorithm. A complete state space search has to consider all transitions which could be executed first from a given state, in order to generate all possible interleavings. In an untimed concurrent system, this leads to a number of interleavings (and explored states) which is exponential in the number of concurrent components. In a timed system, the firing times of transitions become part of the state space, since the future behavior of the system typically depends on the relationship between them. This has two consequences. First, more information is usually needed to describe a timed state, resulting in a higher amount of memory used. Second, two transitions leading to the same state in the underlying untimed control structure will generally lead to different timed states, since the ordering of transitions is incorporated in the timed space.

Partial order reduction (e.g., [God90, Pel93, Val90]) is a well-established method to reduce the complexity of state space exploration in systems consisting of several parallel components. It explores a restricted number of interleavings for independent concurrent transitions, while preserving the verified property in the reduced model. This aspect makes it a very good candidate for containing the state space explosion in timed systems. However, partial order reduction considers two transition interleavings to be equivalent only if they lead to the same state. Thus, for timed systems, the encoding of transition ordering as part of the state, besides being one of the main causes of complexity also prohibits the direct application of partial order reduction. It is precisely this issue that we propose to address.

1.2 Thesis Approach and Contributions

This dissertation presents solutions for the application of partial order methods to the verification of various models of timed systems, with the purpose of reducing the size of the explored state space.

The general approach followed is to define an alternate semantics for the timed model under investigation, in which causal dependence of transitions is separated from time ordering due to concurrency. This relaxed timing semantics is characterized by a richer set of behaviors. Whereas the standard semantics requires successive transitions to occur in a sequence of monotonically increasing timepoints, in the new semantics some transitions can be explored in an order which may be different from the original ordering of their execution times. However, the behaviors of the new model are restricted in such a way as to preserve the truth value of the specification. We chiefly work in the context of specifications expressed in timed extensions of next-time free linear temporal logic.

Performing the state space search using the modified semantics instead of the original one can be advantageous because the relaxed time ordering condition on transitions leads to the generation of fewer timed states. A timed state no longer needs to encode the total order of transitions leading to it, but merely a partial order representing causality. At the same time, the commutativity of transitions which are independent in the underlying untimed system is restored. This allows partial order reduction to be applied, potentially leading to a yet smaller system model, this time due to a decrease in the number of control states.

The main contributions presented in this dissertation are:

- A method for the application of partial order reduction to networks of timed automata, based on a local-time model. In particular, we show how to effectively search the state space of the system using a local-time model, how to perform verification for a timed extension of linear temporal logic, and we give conditions for the selection of a reduced set of transitions during exploration. Our experimental results show that using a local-time model for exploration leads to a significant reduction in the number of timed regions, while partial order reduction results in a further reduction of the control state space.

- A general formalism for the application of partial order reduction for a class of continuous-time systems. We define a general timed model, and a semantics based on execution traces which separates the issues of transition causality from the ordering of their timestamps. We then show how our semantics naturally allows the application of partial order reduction and present an algorithm that performs a reduced state space search on a model based on timed regions.

- An algorithm that applies partial order reduction to the exploration of timed event/level structures, used in the modeling of asynchronous circuits. Compared to the original algorithm which focuses on exploring fewer timing regions, the new algorithm also reduces the number of control states.

- A technique to apply partial order reduction statically at the time of model construction. This represents joint work, presented in [KLM$^+$98]. The method permits reduction to be separated from the model checking algorithm and combined with other verification techniques, in particular with symbolic model checking.

- A proof for the correctness of partial order reduction with ample sets using a weaker condition for independence between transitions.

1.3 Related Work

We present a brief selective overview of relevant related research in the verification of timed systems, first discussing various models, and then previous work on applying partial order reduction to this domain.

1.3.1 Continuous and Discrete Time

To formalize the notion of time, two main directions have been pursued in the literature. One of them considers a dense (continuous) model of time, equating time with the set of real numbers $\mathbb{R}$. In this model, an event (or a transition) can occur at an arbitrary time point on the real scale. On the other hand, the discrete model of time allows transitions to occur only at discrete time quantums, modeling time using the set of integer numbers $\mathbb{Z}$. Throughout the history of verification for timed systems, the relative merits of the two approaches have been compared and debated [AH91, HMP92].

A comparison of the two models can be made in terms of both expressivity and efficiency. The continuous time model is strictly more expressive than the one employing discrete time. Intuitively, continuous time can model delays that are arbitrarily small. When modeling a system using the dense time paradigm, one does not have to assume that the granularity of the clock is appropriate for modeling all system behaviors. Furthermore, when composing two discrete-time systems, one has to match the granularity of the two clocks, an issue which does not occur with continuous time.

However, for some classes of timed systems, certain properties are preserved by discretization. Henzinger, Manna and Pnueli [HMP92] discuss timed transition systems, i.e., state-transition graphs augmented with upper and lower integer time bounds on transitions. They show that all qualitative (or time-independent) properties, and some common quantitative properties such as time-bounded invariance and time-bounded response are preserved by a discrete-time semantics. Furthermore, if a property expressed in a certain timed logic holds in the continuous-time semantics, a weaker, derived property is guaranteed to hold in discrete time.

On the other hand, there exist systems and properties which are not preserved if a discrete-time model is used instead of continuous time. An analysis for combinational circuits is given in [AMP98]. Again, the timing constraints are expressed as bounded delays which are imposed on the output of each gate. It is shown that for acyclic circuits, a discretization quantum can be found such that qualitative behavior (i.e., event ordering) is preserved. In these cases, a time quantum of $1/n$, where n is the number of signals in the circuit, is sufficient. However, there exist cyclic circuits whose continuous-time qualitative behavior is not preserved by any discretization.

From an efficiency point of view, both discrete- and continuous-time models have their individual advantages and disadvantages, although in general, practical results for discrete-time models have been better, as reported for instance in [BMT99]. Discrete-time techniques allow efficient representation techniques from the untimed domain to be used, such as binary decision diagrams [Bry86]. However, discrete time does not constitute an unconditional improvement. Modeling a system in discrete time can already result in a more complex model than by using a continuous-time semantics. Moreover, discrete-time techniques tend to be more sensitive to the size of the constants appearing in the model descriptions, and large constants can result in state space explosion.

1.3.2 Other Partial Order Approaches

We discuss three of the most common models that have been used for the description and verification of timed systems: timed automata, time Petri nets and timed event/level structures, and the related work that has been carried out to apply partial order reduction to these models.

The first partial order reduction procedure for a timed model seems to have been presented in the context of *time Petri nets* by Yoneda, Schlingloff et al. [YSSC93, YS97]. Their model is an extension of Petri nets in which upper and lower time bounds may be placed on transitions [MF76]. Because of their restricted timing conditions, time Petri nets are less expressive than timed automata. On the other hand, converting a Petri net into a timed automaton can potentially involve an exponential increase in the size of the model. Hence, verification algorithms for time Petri nets are not subsumed directly by those for timed automata. Yoneda and Schlingloff prove a partial order reduction algorithm that preserves properties in a timed extension of next-time free LTL. The fundamental idea of their approach is that only transitions from the reduced set chosen for exploration need to be interleaved in all possible time orderings. In Chapter 5 we show how this idea can be generalized, and the required condition can be weakened. Sloan and Buy [SB96, SB97] give a procedure similar to [YS97] for a more restrictive model of simple time Petri nets, in which each transition has a static delay. Lilius [Lil98] suggests an improvement that does not store the firing sequence of transitions as part of a timed state, but can only applied to analyzing reachability of place markings.

Timed automata [AD90, ACD90] are finite-state automata augmented with a set of real-valued clocks that evolve at the same rate. Their transitions are guarded by constraints on clocks or their differences. Combining a natural description formalism with high expressive power, they have been extensively studied in the literature (see [AD94] for a comprehensive survey).

The model checking problem for timed automata has been investigated for powerful timed logics such as timed computation tree logic (TCTL) [ACD90] and timed modal μ-calculus [HNSY92]. The worst-case complexity of model checking is exponential in the number of clocks and the size of the maximal time constant in the model. However, model checking tools such as Kronos [NSY92] and Uppaal [LPW95] have implemented efficient search and representation techniques together with various optimizations that have enabled the verification of a number of real-world examples.

The first approach to the application of partial order reduction for systems composed of communicating timed automata is due to Pagani [Pag96, Pag97]. Her analysis shows however that the dependencies between the passage of time and transitions that cause a state change reduce the independence of transitions significantly compared to the untimed case and thus make the application of partial order reduction difficult. An improvement which identifies additional cases where reduction can be applied is presented in [DGKK98].

Bengtsson et al. [BJLW98] were the first to suggest a modified semantics that allows the component automata of a network to execute individually, synchronizing their local time scales only on synchronization transitions. Our results for timed automata are based on their work. However, the only preservation result proved for the new semantics was for local reachability. Moreover, they did not present a concrete verification algorithm, since the new model lacked an effective condition to decide the equality of two timed regions (i.e., a stopping condition in the state space search). As our main result, we show in Chapter 3 how to use this local-time model to perform model checking for a timed extension of linear temporal logic.

Timed event/level structures [BM97] are a specification formalism tailored to the description of asynchronous circuits, derived from the timed event/rule structures of [Mye95]. A rule describes a causal relation between two events, together with a separation interval (integer upper and lower time bounds) between them. They are in essence similar to Petri nets but in addition allow rules to depend on the value of signals. Belluomini and Myers [BM98] present an algorithm that stores only partial ordering relations between events and thus reduces the number of timed states generated during system exploration. However, the term "partial order" here does not imply the exploration of a reduced set of event or rule interleavings. In Chapter 4 we present how partial order reduction (in the sense of exploring a restricted set of events) can be added to their algorithm to also reduce the set of explored control states.

1.3.3 Other Approaches to State Space Explosion

Partial order reduction attempts to alleviate the state explosion problem for timed systems by addressing one specific cause, the redundant exploration of multiple transition interleavings. A wide variety of other methods have been used to contain state space explosion by addressing orthogonal issues. We mention some of the most relevant techniques, since many of them can be used in a model checker together with partial order reduction.

For timed automata, one of the reasons for the large size of the state space is the fact that during state space exploration, all pairs of clocks are related to each other by clock constraints. However, not all clocks are used at every point during the execution of the system. If a clock is not used in any constraint prior to the next point when it is reset, its relation to other clocks is irrelevant, and it can be removed from the representation of the current state. This method, called clock activity reduction, was introduced first by Daws and Yovine [DY96] and can significantly reduce the amount of memory that is necessary to store a timed state.

Another approach that reduces the complexity related to timing is based on the observation that not all the timing information in the description of a timed system is usually needed to guarantee the satisfaction of a given property. An approximation scheme which uses upper and lower bounds on the set of reachable states is described in the Ph.D. thesis of Wong-Toi [Won94]. Approximations have also been studied by Balarin [Bal96], and are incorporated in the model checked RT-Cospan [AK95]. In the latter situation, the underlying untimed description of the system is composed with an automaton representing the time bounds. Only the bounds that are necessary to verify the given property are successively introduced in the composition.

Time-abstracting bisimulations, which hide the quantitative aspects of time, are discussed in the Ph.D. thesis of Tripakis [Tri98]. If a system's quotient is computed with respect to a time-abstracting bisimulation, efficient methods from the untimed domain, such as minimization of the resulting transition system, can be applied for verification. Methods for abstraction of timed systems are also discussed in the thesis of Taşıran [Taş97].

Symbolic techniques based on BDDs have been investigated with great interest in the domain of timed systems, due to their success in the untimed and discrete-time case. Wong-Toi [Won94] reports successful use of BDDs to encode control states that share the same timing information, especially when used together with approximations. Balarin [Bal96] takes a different approach and uses BDDs to encode the difference bound matrices which represent time zones. Bozga, Maler et al. [BM97, BMT99] show that in several cases, BDDs together with discretization enable the verification of systems with more components than using a standard difference bound matrix (DBM) representation and continuous-time semantics. Belluomini [Bel99] uses BDDs for the storage of the reached state sets, but converts to an explicit DBM representation for the exploration algorithm. This modification makes the exploration slower, but enables the verification of larger models.

In previous joint work [CCM$^+$94, CCM97], later extended in the Ph.D. thesis of Campos [Cam96], we have taken a different approach to the verification of timed systems, by focusing on a discrete-time model with unit transitions. Although very simple, this model is applicable in many situations, and has proved especially useful for systems whose components are naturally scheduled to execute in discrete time intervals. Since the model only needs to handle unit-time transitions, symbolic representation and analysis techniques based on BDDs from the domain of untimed systems are directly applicable, and show the same efficiency in practice. As a significant advantage, the approach allows not only the verification of specifications in temporal logics with or without explicit timing, but also the computation of quantitative properties about the system behavior. These include precise lower and upper bounds on execution times or on times spent in states that satisfy certain conditions, and can be used for detailed assessment of system properties.

The fundamental difference between the above approach and the work presented in this thesis lies in the application domain, and has consequences for modeling and efficiency. Most of the examples analyzed with the approach of [CCM$^+$94] are composed of interacting processes executing on a single processor, or represent hardware and embedded systems where signals are discretely sampled. For these, the unit-time model is very appealing, and provides an efficiency that can likely not be matched for a continuous-time model with multiple clocks. Our thesis presents a general approach to reduction that is targeted mostly at asynchronous timed systems in which discretization may not preserve the system behavior, or lead to state space explosion.

More recently, two data structures have been defined that are specifically tailored to the representation of difference constraints that appear in time zones. In both cases, one of the goals is to efficiently represent unions of time zones in the reached state space, rather than having to represent each time zone separately. Clock difference diagrams [BLP$^+$99] are multiway decision diagrams, in which levels are indexed by clock pairs (i.e., clock differences), and each lower-level node corresponds to an interval on the real time scale for the corresponding clock difference. In difference decision diagrams [MLAH99], the decision is binary and is given by the truth value of an atomic clock constraint. In addition, DDDs are the first data structure that makes possible model checking of timed automata in a fully symbolic fashion.

1.4 Outline

Chapter 2 starts by presenting the basic principles underlying partial order reduction. We give a proof for the correctness of partial order reduction using a weaker notion of independence. Next, we present a static approach to reduction, in which the reduced model is generated at compile-time. The next three chapters present our results concerning the application of reduction to timed systems. In Chapter 3, after introducing the local-time model for networks of timed automata, we show how to apply partial order reduction to the model checking of a timed extension of LTL. Chapter 4 presents a different reduction method, also for timed automata, but this time based on the region graph construction. Then, we show how partial order reduction can be incorporated into an exploration algorithm for timed event/level structures.

Chapter 5 presents our most general result. We identify the principles underlying the reduction techniques presented so far and apply them to a model of timed systems that can be particularized to either timed automata, time Petri nets or TEL structures. Chapter 6 presents a performance evaluation of the reduction method from Chapter 3 on systems modeled as timed automata, the most expressive of the timed models analyzed so far with partial order reduction. Finally, our conclusions and some directions for future work can be found in Chapter 7.

Chapter 2

Partial Order Reduction

2.1 Introduction

The main obstacle for automatic verification methods based on state space exploration is the fact that the systems to be verified often have prohibitively many states for an exhaustive traversal. The state space of a system made up of several components is the product of the state spaces of the individual parts, and its size is therefore exponential in the number of components. Thus, the size of the global system quickly becomes unmanageable, even if each individual component is of relatively small size. This has been called the *state space explosion problem.*

A wide array of techniques has been developed to alleviate this problem. Methods based on compositional reasoning verify the system behavior based on properties of the individual components, without having to construct the global state space. Other methods are relatively independent of the modular system structure. Abstraction techniques create smaller, high-level models that approximate the original one, by removing irrelevant detail. On-the-fly and local model checking techniques restrict exploration to only those parts of the system state space which are relevant for the verified property. Symbolic techniques use an implicit representation of the state space, which does not bear a direct relationship to the number of states and can be significantly smaller.

Partial order reduction is a technique that constructs a smaller state space by addressing a specific reason behind the state space explosion, namely the existence of many potentially equivalent execution traces. This method is typically applied to asynchronous systems, which are described using an interleaving model of computation. Concurrent events are modeled by allowing their execution in all possible orders relative to each other. This serialization creates a large number of possible states and paths. However, not all different interleavings can be generally distinguished by a specification. Partial order reduction techniques take advantage of this by generating and exploring a model with only a reduced set of interleavings, and thus fewer states. At the same time, the reduced model is guaranteed to contains at least one representative from each class of equivalent behaviors, thus preserving the truth value of the specification.

In this chapter, we first present the basic principles behind the partial order reduction method. Next, we prove that a relaxed independence relation between transitions is sufficient to ensure the correctness of the ample set method for partial order reduction. Finally, we present a variant called *static partial order reduction*, which incorporates reduction into the model in a preprocessing step and is thus independent of the model checking algorithm. In particular, this method can be combined with symbolic state space exploration techniques.

Several approaches that use the commutativity between selected transitions to reduce the state space of a system have been suggested in the literature. The first such method seems to have been suggested by Overman [Ove81] in his Ph.D. thesis. However, it was restricted to models whose state space did not contain loops. Later on, Katz and Peled [KP88] described a proof system for concurrent systems that took advantage of commutativity between transitions. This deduction system used as its core a set of proof rules that asserted properties of sequences that are generated by exploring certain subsets of successors from each state.

Over the last decade, several methods have been developed that apply partial order reduction to model checking of finite-state systems. The common characteristic of all these methods is that they explore only a certain subset of transitions from each state. They differ in the conditions imposed on the reduced transition set in order to guarantee correctness. Such techniques are the *stubborn set* method of Valmari [Val90], the *persistent set* method of Godefroid and Wolper [GW91, God96], and the *ample set* method of Peled [Pel93]. We will focus here on the ample set method, occasionally borrowing ideas from the stubborn set technique.

The name *partial order reduction* reflects the fact that the initial versions of this method used an explicit partial order semantics. Generally speaking, a *partially ordered execution* is represented by a set of events and a causality relation between them. The causality relation indicates which events have to precede others in any system execution, whereas the remainder of the events that are not restricted by this relation are independent and can occur in any order. This view of the system is in contrast to a total ordering on events, in which all events are serialized, i.e., any event either precedes or follows any other event. There are versions of partial order reduction that are explicitly based on the fact that the generated reduced state space includes at least one completion into a total order for each partially ordered execution. However, most current methods are no longer based on explicitly maintaining this relation.

2.2 Basic Notions

We analyze systems that are modeled as *state transition graphs.* Let S be the set of system states. A *transition* is identified with a particular action that the system can execute and is given by a relation $\alpha \subseteq S \times S$, which defines the pairs of states between which the action can be executed. A state transition graph is a tuple $M = (S, S_0, T, L)$, where $S_0 \subseteq S$ is a set of initial states, T is a set of transitions $\alpha \subseteq S \times S$, and $L : S \to \mathcal{P}(AP)$ is a labeling function that assigns to each state a subset of the set AP of atomic propositions.

A transition $\alpha \in T$ is *enabled* in a state s if there exists a state s' such that $(s, s') \in \alpha$. Otherwise, α is said to be *disabled* at s. A transition is *deterministic* if for any state $s \in S$ there exists at most one state $s' \in S$ such that $(s, s') \in \alpha$. In this case, α is in fact a partial function on S, and we will use the notation $s' = \alpha(s)$ instead of $(s, s') \in \alpha$. In the following, we will restrict ourselves to systems with deterministic transitions. It is still possible to model nondeterminism in such systems, since in general there can be more than one transition enabled at a given state.

An *execution sequence* σ of a state transition graph is an infinite sequence $\sigma = s_0 \stackrel{\alpha_0}{\to} s_1 \stackrel{\alpha_1}{\to} \ldots$ such that for all i, $s_{i+1} = \alpha_i(s_i)$. We denote by σ_i the suffix of σ that starts at state s_i, i.e., $\sigma_i = s_i \stackrel{\alpha_i}{\to} s_{i+1} \stackrel{\alpha_{i+1}}{\to} s_{i+2} \stackrel{\alpha_{i+2}}{\to} \ldots$. An execution sequence σ is an *initial execution sequence* if s_0, the first state in the sequence, belongs to the set of initial states S_0 of M.

In an asynchronous system, an execution trace serializes transitions regardless whether they occur sequentially in the same component or concurrently in different components. Therefore, the number of transitions separating two events has no direct relationship to the time delay between them. Moreover, a transition which does not change the state labeling (also called a *stuttering step*) and is concurrent with an observable event will be necessarily serialized either before or after it. However, given the concurrent semantics of the system, the serialization order should not affect the specification. These observations argue (cf. [Lam83]) for a specification which cannot distinguish between sequences of identically labelled states on an execution path of the system.

Two infinite execution sequences are *stuttering equivalent* (Figure 2.1) if they reduce to identical sequences of state labelings after in each of them, any finite sequence of identically labeled states is collapsed to a single state. In other words, two infinite paths $\sigma = s_0 \stackrel{\alpha_0}{\rightarrow} s_1 \stackrel{\alpha_1}{\rightarrow} \ldots$ and $\rho = r_0 \stackrel{\beta_0}{\rightarrow} r_1 \stackrel{\beta_1}{\rightarrow} \ldots$ are stuttering equivalent if one can define two infinite sequences of integers $0 = i_0 < i_1 < \ldots$ and $0 = j_0 < j_1 < \ldots$ such that $\forall k \geq 0$, $L(s_{i_k}) = L(s_{i_k+1}) = \ldots = L(s_{i_{k+1}-1}) = L(r_{j_k}) = L(r_{j_k+1}) = \ldots = L(r_{j_{k+1}-1})$. The indices i_k and j_k are the starting points of identically labeled subsequences of states in the two paths, respectively. The stuttering equivalence relation between σ and ρ is denoted by $\sigma \sim_{\mathrm{st}} \rho$.

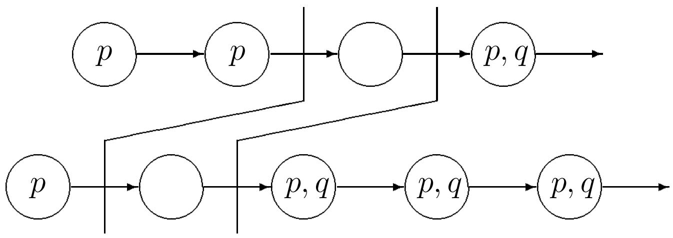

Figure 2.1: Stuttering equivalent paths

For assertions about the behavior of a program, we use the temporal logic LTL [GPSS80]. Given a finite set of propositions AP, the formulas of LTL are defined inductively as follows:

- p is a formula, for every $p \in AP$
- if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\mathbf{X}\,\varphi$ and $\varphi\,\mathbf{U}\,\psi$.

An execution sequence $\sigma = s_0 \stackrel{\alpha_0}{\rightarrow} s_1 \stackrel{\alpha_1}{\rightarrow} \ldots$ is said to satisfy an LTL formula ϕ (denoted by $\sigma \models \phi$) under the following conditions:

- $\sigma \models p$ iff $p \in L(s_0)$, for $p \in AP$,
- $\sigma \models \neg\varphi$ iff not $\sigma \models \varphi$,
- $\sigma \models \varphi \wedge \psi$ iff $\sigma \models \varphi$ and $\sigma \models \psi$,
- $\sigma \models \mathbf{X}\,\varphi$ iff $\sigma_1 \models \varphi$,
- $\sigma \models \varphi\,\mathbf{U}\,\psi$ iff $\exists i \geq 0$ such that $\sigma_i \models \psi$ and $\forall j\,.\,0 \leq j < i \Rightarrow \sigma_j \models \varphi$.

Let *false* stand as an abbreviation for $p \wedge \neg p$, and *true* be an abbreviation for $\neg$*false*. We also use the following abbreviations: $\varphi \vee \psi = \neg((\neg\varphi) \wedge (\neg\psi))$, $\mathbf{F}\,\varphi = \mathit{true}\,\mathbf{U}\,\varphi$, $\mathbf{G}\,\varphi = \neg\mathbf{F}\,\neg\varphi$.

For a given state transition graph M and LTL formula φ, the *model checking problem* for M and φ is to verify that for every initial state $s_0 \in S_0$ and every path σ starting in s_0, it is true that $\sigma \models \varphi$. We write $M \models \varphi$ to denote that the formula φ is true in model M.

An LTL formula φ is *invariant under stuttering* if for any two paths σ and σ' such that $\sigma \sim_{st} \sigma'$, we have $\sigma \models \varphi$ iff $\sigma' \models \varphi$.

Recall that we have argued for the use of specifications that cannot distinguish between stuttering equivalent sequences. In general, an LTL formula is sensitive to stuttering if it contains the next-time operator $\mathbf{X}$. Denote by LTL$_{-\mathrm{X}}$ the subset of logic LTL that does not make use of the next-time operator. It has been shown by Peled and Wilke [PW97] that an LTL property is invariant under stuttering precisely if it can be expressed in LTL$_{-\mathrm{X}}$.

The notion of stuttering equivalence can be naturally extended from paths to state transition graphs. Two state transition graphs M and M' labeled with the same set of atomic propositions are stuttering equivalent if a correspondence relation can be established covering all their execution sequences, such that corresponding execution sequences are stuttering equivalent. Specifically,

- for each initial execution sequence σ of M there exists an initial execution sequence σ' of M' such that $\sigma \sim_{st} \sigma'$.
- for each initial execution sequence σ' of M' there exists an initial execution sequence σ of M such that $\sigma' \sim_{st} \sigma$.

The fact that LTL$_{-\mathrm{X}}$ formulas are stuttering invariant together with the definition of stuttering equivalence of state transition graphs imply the following result:

If M and M' are stuttering equivalent state transition graphs, then for any $\mathrm{LTL}_{-\mathrm{X}}$ formula φ, $M \models \varphi$ iff $M' \models \varphi$.

This is the main result which justifies the use of partial order reduction, since this method generates a reduced model that is stuttering equivalent to the original one. In the next section, we describe the general principles that stand behind the reduced state space generation.

2.3 Principles of Partial Order Reduction

The results about stuttering presented in the previous section imply that when model checking a concurrent asynchronous system with respect to a stuttering-invariant specification, one does not need to explore all behaviors of the model. If the execution sequences are divided into equivalence classes with respect to stuttering equivalence, it is sufficient to select just one behavior from each class as part of the reduced model in order to guarantee correctness.

Consider an example that illustrates the importance of reduction. Assume that the system to be verified is composed of n concurrent processes, $P_1, P_2, \cdots, P_n$. Each process P_i has a transition α_i enabled in some local state s_i, such that $\alpha_i(s_i) = s'_i$. The concurrent transitions α_i can be ordered in $n!$ possible ways, resulting in the exploration of 2^n different states. However, it is possible that the specification is a property that relates the initial global state $(s_1, \ldots, s_n)$ with the resulting global state $(s'_1, \ldots, s'_n)$, without depending on intermediate states, and the path taken between these. Thus, it is much more efficient to consider only one particular ordering and the corresponding $n + 1$ states.

In most variants of partial order reduction, the reduced model of the system is built by performing a modified depth-first search on an explicit-state representation of the system. This is followed by a separate model checking phase performed on the reduced state-transition graph. Another option is to construct the reduced model *on the fly*, during model checking. This has the advantage that the state space construction can be guided taking into account the specification, and the size of the constructed model can be reduced further. Another variant, described in detail later in this chapter involves the use of breadth-first search, which has the potential of combining partial order reduction with a symbolic representation. The essential aspect common to all these approaches is that the reduced model is built directly, without first constructing the full state graph of the original system. This is a natural requirement, since the full model is typically too large to be constructed in the first place, and an indirect approach would defeat the purpose of the reduction.

The selection of representative behaviors is made by following from each state only a subset of the enabled transitions, as opposed to an ordinary search which would explore all of them. We denote by $ample(s) \subseteq enabled(s)$ the set of transitions which are explored from state s in the case of partial order reduction.

The key to applying partial order reduction is a procedure that calculates at each state s a suitable set $ample(s)$ of transitions to be explored. On one hand, this set should be small (significantly smaller than the set of all enabled transitions), in order to effectively reduce the searched state space. On the other hand, the correctness of the verification result has to be preserved, by including in the reduced state graph at least one equivalent execution sequence for each execution of the original model. Finally, the overhead for computing an ample set should be sufficiently small such that the verification time is not increased compared to full state space search, offsetting the benefits of the reduction.

In order to obtain such a procedure for selecting transitions, one has to formalize the notion of transitions that can be reordered. Two key concepts play a role in this process: the notion of transition *independence* relates to the interaction between the execution of transitions in the model, whereas transition *visibility* is determined by the properties examined by the specification.

Two transitions $\alpha, \beta \in T$ are *independent* in a state $s \in S$ if they satisfy the following two conditions:

- Enabledness: If $\alpha, \beta \in enabled(s)$, then $\alpha \in enabled(\beta(s))$ and symmetrically $\beta \in enabled(\alpha(s))$.

- Commutativity: If $\alpha, \beta \in enabled(s)$ then $\alpha(\beta(s)) = \beta(\alpha(s))$.

The enabledness condition expresses the fact that two independent transitions that are enabled at a given state cannot disable each other. The commutativity condition states that the execution of two independent transitions in any order (which is guaranteed to be possible by the enabledness condition) leads to the same state. Two transitions that are independent at each state $s \in S$ are called *globally independent.* In the following, "independent" implicitly stands for "globally independent", unless a specific state is mentioned. Two transitions are called *dependent* (at a particular state or globally) if they are not *independent.*

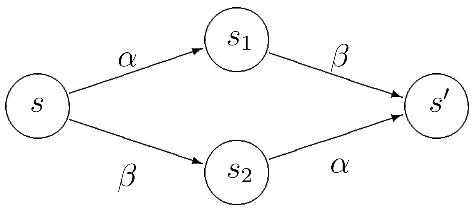

Figure 2.2: Independent transitions

The independence relation can be pictorially represented using a diagram such as the one in Figure 2.2, which depicts a simple fragment of a state transition graph. Transitions α and β are independent in state s. A possible reduction would consider only the execution sequence $s \xrightarrow{\alpha} s_1 \xrightarrow{\beta} s'$, eliminating the path $s \xrightarrow{\beta} s_2 \xrightarrow{\alpha} s'$. However, this reduction is only correct if the checked property cannot distinguish between the intermediate states s_1 and s_2. (Additional conditions for the correctness of the reduction are needed, and they will be described in the next subsection. For instance, eliminating one of these states may prevent the exploration of its successors, which may be significant for verification.)

The definition of independence given here requires independent transitions not to *disable* one another. However, the execution of a transition can *enable* the execution of another one, while maintaining independence. The partial order reduction literature often uses a more restrictive version of the enabledness condition that requires independent transitions to neither disable or enable one another. Specifically, the more restrictive condition requires that if $\alpha \in \mathit{enabled}(s)$, then $\beta \in \mathit{enabled}(s)$ iff $\beta \in \mathit{enabled}(\alpha(s))$, together with the symmetric condition with α and β reversed (the commutativity condition remains the same). In the stubborn set approach of Valmari [Val90] the less restrictive condition is consistently used, whereas the persistent set method of Godefroid et al. [God96] is defined using the more restrictive condition. The papers describing the approach of Peled generally use the more restrictive condition, save for [Pel94] (revised in [Pel96a] to the more restrictive condition) and [HP94]. In both of the latter cases, reference to proofs made using the more restrictive condition is made. In the following we use the less restrictive condition and prove that it is sufficient for handling $\mathrm{LTL}_{-\mathrm{X}}$.

To examine what it means for a specification to distinguish between two states, we introduce a second key notion, that of transition *visibility*. Recall that one of the elements of the state transition graph is the labeling function $L : S \to \mathcal{P}(AP)$ which assigns to each state a set of atomic propositions. The specification may not observe all atomic propositions in AP; let $AP' \subseteq AP$ be the subset of atomic propositions which are actually used in the formula. A transition α is called *invisible* with respect to $AP' \subseteq AP$ if its execution between any two states does not change the labeling with atomic propositions from AP'. Formally, the transition $\alpha \in T$ is invisible with respect to AP' if for any two states $s, s' \in S$ with $s' = \alpha(s)$ we have $L(s) \cap AP' = L(s') \cap AP'$. A transition is called *visible* if it is not invisible. Since the set of atomic propositions with respect to which we consider visibility is typically given by the specification, in the following we will use the terms "visible" and "invisible" without referring specifically to a set of atomic propositions AP'.

2.4 Conditions for Partial Order Reduction

The notions of independence and visibility of transitions are the fundamental properties taken into account when selecting a reduced set of transitions to explore at a given state. The selected subset of transitions should be small, in order to facilitate reduction. However, if at some state a reduced set of transitions cannot be found, the search algorithm is safe in exploring all enabled transitions. In this case, if $\mathit{ample}(s) = \mathit{enabled}(s)$, the state is said to be *fully expanded*.

In order to describe the most general reduction conditions, and at the same time facilitate a natural proof of correctness, the reduced sets of transitions at each state are not described operationally by means of an algorithm to select them. Rather, a set of conditions is given that these transitions must satisfy [Pel93]. Following these conditions, algorithms and heuristics can be devised that actually construct an ample set for each state. Such algorithms are reviewed in a later section.

The first trivial condition has to ensure that at each step some new state can be explored in the reduced model if this is possible in the original model:

C0 (Emptiness) $\mathit{ample}(s) = \emptyset$ iff $\mathit{enabled}(s) = \emptyset$.

The next constraint ensures that any path of the original state graph can be transformed into a path of the reduced model by commuting independent transitions. This is a first step to guarantee that the reduced model will contain a representative for each path in the full state space.

C1 (Faithful decomposition) *For any execution sequence* $s_0 \stackrel{\alpha_0}{\rightarrow} s_1 \stackrel{\alpha_1}{\rightarrow} \ldots$ *of* M*, and for any* $k \in \mathbb{N}$*, if* $\alpha_i \notin ample(s_0)$ *for* $0 \leq i \leq k$*, then* α_i *is independent of any transition* $\beta \in ample(s_0)$ *for* $0 \leq i \leq k$*.*

In other words, on any execution sequence of the original model starting at some state s, no transition which is dependent on a transition from $ample(s)$ can occur before some transition from $ample(s)$ is executed. Since any transition in $enabled(s) \setminus ample(s)$ can be executed from s in the original model, this implies immediately that any transition which is not in $ample(s)$ is independent from any transition in $ample(s)$. This property has been named "faithful decomposition" in [KP88], since the set of enabled transitions at any state s is partitioned into two sets, $ample(s)$ and its complement, and neither of the transitions in one of the two sets can affect the execution of a transition in the other set.

Condition **C1** is used to show that for any execution sequence σ starting at some state s_0 in the original model, some transition in $ample(s_0)$ can be taken without disabling any of the transitions in the given sequence. This in turn, can be used as an inductive argument to construct an execution sequence in the reduced state model from each execution sequence in the original model (a complete proof is given in a subsequent section). We explain informally why this condition holds and give a complete proof in a later section. If the first transition is an ample transition, $\alpha_0 \in ample(s_0)$, the property is trivially true. The following two cases remain:

$$
\begin{tikzcd}
s_0 \arrow[r, "\alpha_0"] \arrow[d, "\beta"'] & s_1 \arrow[r, "\alpha_1"] \arrow[d, "\beta"'] & \cdots \arrow[r, "\alpha_{k-2}"] & s_{k-1} \arrow[r, "\alpha_{k-1}"] \arrow[d, "\beta"'] & s_k \arrow[d, "\beta"'] \\
s'_0 \arrow[r, "\alpha_0"] & s'_1 \arrow[r, "\alpha_1"] & \cdots \arrow[r, "\alpha_{k-2}"] & s'_{k-1} \arrow[r, "\alpha_{k-1}"] & s'_k
\end{tikzcd}
$$

Figure 2.3: Reordering of transitions based on commutativity

(a) σ contains some transition from $ample(s_0)$. Let the first such transition be $\beta = \alpha_k$, with $k \geq 1$. By condition **C1**, $\alpha_k \in ample(s_0)$ is independent of $\alpha_0, \ldots, \alpha_{k-1}$ and commutes with all these transitions.

Thus, the transition sequence $\alpha_k\alpha_0\alpha_1\ldots\alpha_{k-1}$ can be executed in s_0, leads to the same state as the transition sequence $\alpha_0\alpha_1\ldots\alpha_k$, and can be followed from this state by the remaining suffix σ_k of σ.

(b) σ does not contain any transition from $ample(s_0)$. Let $\beta \in ample(s_0)$ be an arbitrary transition. By condition **C1**, β is independent from *all* transitions in σ. Therefore, if $s'_1 = \beta(s_0)$, then $\alpha_0 \in enabled(s'_1)$, and inductively if follows that the entire transition sequence $\alpha_0\alpha_1\ldots$ can be executed from s'_1.

However, the fact that each path in the full state space can be transformed into a path which includes the same transitions and has a prefix which belongs to the reduced model is not sufficient in itself. One has to guarantee that the specification is not affected, by ensuring that the generated path is stuttering equivalent to the original one. This aspect is handled by the following condition:

C2 (Visibility) *If $ample(s)$ contains a visible transition, then the state s is fully expanded, i.e., $ample(s) = enabled(s)$.*

We explain the effect of this condition based on the cases (a) and (b) presented for condition **C1**. In case (a), since $\alpha_0 \notin ample(s_0)$, it follows that state s_0 is not fully expanded and thus all transitions from it must be invisible. If we denote $s'_i = \alpha_k(s_i)$, for $0 \leq i \leq k$ (cf. Fig. 2.3 for $\beta = \alpha_k$), then we have $L(s_i) = L(s'_i)$. Thus the two state sequences $s_0s_1\ldots s_ks'_k$ and $s_0s'_0s'_1\ldots s'_{k-1}s'_k$ are stuttering equivalent, since a one-to-one correspondence of labelings exists after collapsing s_k with s'_k in the first sequence and s_0 with s'_0 in the second. A similar argument holds in case (b). Here too, β must be invisible, and after collapsing s_0 and s'_0, the prefixes $s_0s_1\ldots s_k$ and $s_0s'_0s'_1\ldots s'_k$ are stuttering equivalent for any k.

Note that one of the possible transformation cases described for condition **C1** (specifically, the second) does not consume any transition from σ while generating an alternate execution sequence in the reduced state model. Instead, a supplementary transition from the ample set of the current state is inserted. It is possible for this step to be repeated sufficiently often, so that the inserted ample transitions closes a cycle in the state space of the reduced (and original) model (see Figure 2.4). Then, a stuttering-equivalent path for σ will not be generated, since the transition α_0 will never be explored, despite remaining continually enabled while executing the ample transitions. This can affect the truth value of the specification, since α_0 may be visible or lead to parts of the state space which are not explored otherwise. The following condition guarantees that no transition is ignored and the above case does not occur:

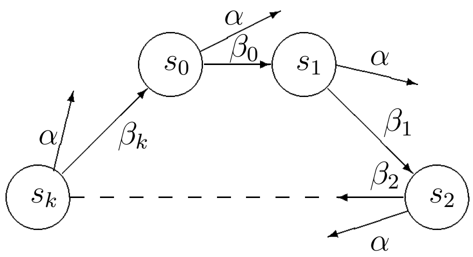

Figure 2.4: Cycle-closing condition

C3 (Cycle closing) *A transition which is enabled in every state of a cycle in the reduced state space belongs to the ample set of some state on the cycle.*

The conditions for partial order reduction can be simplified if model checking is done under fairness assumptions. Typically, the verified system consists of multiple processes, and the usual notion of fairness states that each process has to execute infinitely often. Noting that two transitions enabled at the same local state of a process are dependent, this notion of fairness implies the following condition [Pel94, Pel96a]:

F *If a transition α is enabled in the starting state s of an execution sequence σ, then σ must contain either α or a transition dependent on α.*

The fairness condition **F** ensures precisely that case (b) discussed above cannot happen. Indeed, if **F** is applied to $\alpha \in \mathit{ample}(s)$, then σ must contain either α or some transition β dependent on it. In the latter case, **C1** states that some transition in $\mathit{ample}(s)$ must appear in σ before β. In either case, σ contains a transition from $\mathit{ample}(s)$. In [Pel96a], the visibility condition **C2** is handled by including visible transitions in the dependence relation. Since **C1** implies that an ample transition is independent of all transitions outside the ample set, it follows that an ample set that contains one visible transition has to contain all of them. Thus, condition **C1** subsumes **C2** in this case.

This completes the presentation of the conditions which characterize ample sets. It can be shown that under these conditions, the constructed reduced model is stuttering equivalent to the original one. In the next section, we give a new proof that this result holds even if we use the less restrictive notion for transition independence discussed in Section 2.3.

2.5 A Proof for Partial Order Reduction

The correctness proofs for the ample set given in the literature employ a restricted definition of transition independence, which requires that two independent transitions neither disable or enable one another at any state. However, the stubborn set method of Valmari uses the less restrictive version presented in Section 2.3, which considers two transitions independent even if one of them enables the other. Godefroid [God96] presents his persistent set approach using the more restrictive independence condition, but does not mention in his comparison to ample sets and stubborn sets whether this difference is relevant or not.

A clear statement regarding the two different conditions is important, since the weaker version allows the selection of potentially smaller ample sets. Moreover, the weak version also forms the basis for existing criteria and heuristics for ample set selection such as those used in the SPIN model checker [HP94]. In the following, we prove the the correctness of ample set reduction for $\mathrm{LTL}_{-\mathrm{X}}$ model checking using the weaker independence condition. An alternative, independent proof of this result is given in [CGP99].

We prove that for every transition sequence σ in the original state transition graph we can construct a stuttering equivalent sequence σ' in the reduced model. Let $\sigma = s_0 \stackrel{\alpha_0}{\rightarrow} s_1 \stackrel{\alpha_1}{\rightarrow} \cdots s_n \stackrel{\alpha_n}{\rightarrow} \cdots$ be an arbitrary transition sequence. Given σ and a natural number i, we denote by $\sigma_{\leq i}$ the prefix of σ formed by taking the first i transitions, and by $\sigma_{\geq i}$ the remaining suffix of the transition sequence. We prove by induction on i that for prefixes of σ with length $i \geq 0$ we can construct a sequence $\sigma'_{\leq j}$ of length $j \geq 0$ which is stuttering equivalent to $\sigma_{\leq i}$. Moreover, $\forall k < j \,.\, \alpha'_k \in \mathit{ample}(s'_k)$, i.e., $\sigma'_{\leq j}$ is a finite sequence of transitions which can be taken in the reduced model. In the course of the induction proof, we will refer to i and j as the *current points* in σ and σ', respectively. At each point, $\sigma'_{\leq j}$ will contain all transitions of $\sigma_{\leq i}$ (in some order), with two possible types of transitions added:

(i) Ample transitions after the current point in σ may be executed earlier (before the current point) in σ'. The finite ordered set (sequence) $I \subseteq \mathbb{N}$ contains the indices of transitions beyond the current point i in σ that have been already included in σ'. We call such transitions *marked*.

(ii) Additional ample transitions may be inserted in σ' in order to ensure that it is a legal transition sequence in the reduced model. We denote the sequence of all such inserted transitions by δ.

Notation: If $I \subseteq \mathbb{N}$ is a finite increasingly ordered set of indices, we denote by $\sigma|_I$ the transition sequence obtained by selecting from σ the transitions with indexes in I (in the given order). Similarly, we denote by $\sigma|_{\bar{I}}$ the sequence obtained by deleting from σ the transitions whose indices are in I (here the ordering of I is irrelevant).

Our induction invariant relates σ, i, σ', j, I, and δ as follows:

(a) The transition sequence $\sigma'_{\leq j}$ is stuttering equivalent to $\sigma_{\leq i}$. In particular, $L(s_i) = L(s'_j)$.

(b) If $k \in I$, then $k \geq i$ and α_k is invisible and independent of α_l, for all $l \notin I, i \leq l < k$. (A marked transition is invisible and independent of all unmarked transitions past the current point in σ but preceding it.)

(c) The transition sequence $(\alpha_k)|k \geq i, k \notin I$, obtained from the suffix $\sigma_{\geq i}$ by removing marked transitions, is enabled in s'_j in the original model.

(d) Each transition in δ is invisible and independent of all transitions α_k, $\forall k \geq i, k \notin I$ (all unmarked transitions past the current point in σ).

(e) $s_i \overset{\sigma|_I\delta}{\rightarrow} s'_j$. That is, the marked transitions (comprising $\sigma|_I$) together with the inserted transitions (comprising δ) are exactly those that belong to $\sigma'_{\leq j}$ but not to $\sigma_{\leq i}$. Their sequence is enabled in s_i and takes this state to s'_j.

For the base case, choose $j = i = 0$, $s'_0 = s_0$, $I = \emptyset$ and $\delta = \epsilon$ (the empty sequence). All parts of the invariant are trivially satisfied: (a) is true because both transition sequences consist of just the same initial state, (b) is vacuously true, since I is empty, (c) is true since σ is enabled in s_0, (d) is vacuously true since δ is empty, and (e) is true since both $\sigma|_I = \delta = \epsilon$ and $s_0 = s'_0$.

For the induction step, we consider the following cases:

1. $i \in I$. (The next transition in σ is marked.) Let $i' = i + 1$ and $I' = I \setminus \{i\}$. That is, we advance the current point in σ and delete α_i from the set of marked transitions, since it is now before the current point. Since α_i is invisible according to (b), we have $L(s_{i+1}) = L(s_i) = L(s'_j)$, which maintains (a). Part (b) still holds since no transitions are added to I, and α_i is no longer relevant for the independence condition (since

$i' = i+1$). For parts (c) and (d) the unmarked sequence of transitions after the current point in σ remains the same (α_i is no longer marked, but it is now before the current point), and δ does not change either. Finally, $i = \min I$, so α_i is the first transition in $\sigma|_I$, therefore (e) can be written as $s_i \xrightarrow{\alpha_i} s_{i+1} \xrightarrow{\sigma|_{I'}\delta} s'_j$, the last part of which is exactly (e) after this step.

2. $i \notin I$ and $\alpha_i \in \mathit{ample}(s'_j)$, so α_i is a legal transition in the reduced model. We include α_i in σ', advance both counters ($i' = i+1, j' = j+1$) and set $s'_{j+1} = \alpha_i(s'_j)$. By (b) and (d), α_i is independent of all transitions in $\sigma|_I$ and δ and therefore commutes with them. Since $s_i \xrightarrow{\sigma|_I\delta} s'_j \xrightarrow{\alpha_i} s'_{j+1}$, it follows that $s_i \xrightarrow{\alpha_i} s_{i+1} \xrightarrow{\sigma|_I\delta} s'_{j+1}$, which proves (e). Moreover, $L(s_{i+1}) = L(s_{j+1})$, so (a) is preserved. Part (b) is preserved since I is the same ($i' = i+1$ but $\alpha_i \notin I$), and α_i no longer appears in the independence condition. The transition sequence in part (c) is of the form $\alpha_i\beta$ (with β some transition string). If $\alpha_i\beta$ is enabled in s'_j, then β is enabled in $\alpha_i(s'_j) = s'_{j+1}$. Finally, part (d) is weakened since α_i no longer appears.

3. $i \notin I$, $\alpha_i \notin \mathit{ample}(s'_j)$, and $\exists k > i, k \notin I$, such that $\alpha_k \in \mathit{ample}(s'_j)$. That is, α_i is neither marked nor ample, but there is an ample unmarked transition α_k later in the sequence. Let k be the smallest such index. We mark transition α_k and append it to σ', i.e., $I' = I \cup k$, $j' = j+1$ and $s'_{j+1} = \alpha_k(s'_j)$. Because α_k is ample, $\sigma'_{\leq j'}$ is a legal transition sequence in the reduced model. Since $\alpha_i \notin \mathit{ample}(s'_j)$, s'_j is not fully expanded, and α_k is invisible by **C2**. Thus $L(s'_{j+1}) = L(s'_j) = L(s_i)$, and (a) still holds. By condition **C1**, all transitions preceding α_k in $\sigma_{\geq i}|_{\bar{I}}$ (i.e., α_l with $i \leq l < k, l \notin I$) have to be independent of α_k, so (b) is preserved. Independence implies commutativity, so α_k can be executed as first transition of $\sigma_{\geq i}|_{\bar{I}}$ in s'_j, and the remainder of this sequence remains enabled in s'_{j+1}, which proves (c). Part (d) still holds since δ is the same, and there is one less unmarked transition. By substituting α_k for α_l in (b), we obtain that α_k commutes with all marked transitions which occur later in σ, and because of (d) it also commutes with the transitions in δ. Therefore, since $I' = I \cup k$, $s_i \xrightarrow{\sigma|_I\delta} s'_j \xrightarrow{\alpha_k} s'_{j+1}$ implies $s_i \xrightarrow{\sigma|_{I'}\delta} s'_{j+1}$ (with α_k inserted to preserve the increasing ordering of I') and the final part of the invariant is proved.

4. $i \notin I$, and $\forall k \geq i, k \notin I$, $\alpha_k \notin ample(s'_j)$. That is, there is no remaining unmarked transition which belongs to $ample(s'_j)$. We need to insert an ample transition so σ' remains a legal transition sequence in the reduced model. Select an arbitrary transition $\beta \in ample(s'_j)$ and let $j' = j{+}1$, $s'_{j+1} = \beta(s'_j)$. We also append β to the sequence of transitions inserted so far, $\delta' = \delta\beta$. Again, since s'_j is not fully expanded, β has to be invisible, so $L(s'_{j+1}) = L(s'_j) = L(s_i)$, and (a) still holds. None of the variables involved in (b) changes. Since β is independent of all transitions in $\sigma_{\geq i}|_{\bar{I}}$, (c) remains valid as well, and β can also be appended to δ without violating (d). Finally, $s_i \stackrel{\sigma|_I\delta}{\rightarrow} s'_j \stackrel{\beta}{\rightarrow} s'_{j+1}$, therefore $s_i \stackrel{\sigma|_I\delta'}{\rightarrow} s'_{j+1}$, which proves (e).

To conclude the induction proof, we note that only a finite number of steps of type (3) or (4) (for which the current point in σ is not advanced) can be taken without performing either (1) or (2). Otherwise, the transition sequence $\sigma'_{\geq j}$ eventually closes a cycle on which transition α_i is always enabled without ever belonging to an ample set, which contradicts **C3**. Therefore, after a finite number of steps either (1) or (2) must be performed, which advances the current point in σ by 1, $i' = i + 1$. The above four cases therefore guarantee a finite procedure that constructs in the reduced model a stuttering equivalent prefix for $\sigma_{\leq i+1}$ starting from a similar prefix for $\sigma_{\leq i}$. We also note that since every transition in $\sigma_{\leq i}$ is included in $\sigma'{\leq_j}$, we have $i \leq j$, which ensures that j grows unbounded as i does. By induction, a stuttering equivalent sequence σ' exists in the reduced model for the entire transition sequence σ, q.e.d.

2.6 Calculating Ample Sets

The established conditions for ample set reduction do not directly provide an operational procedure that effectively determines an ample set of transitions at each state. To apply partial order reduction in practice, a procedure which computes ample sets has to be devised. On one hand, this procedure must generate ample sets that are small enough so that the resulting state space is significantly smaller than the original one. On the other hand, the algorithm must be sufficiently simple so that it can be implemented easily, without introducing significant overhead and slowing down verification. This section reviews some selection criteria which are typically employed to ensure that each of the given conditions is satisfied.

It is trivial to verify that the ample set is nonempty (condition **C0**). Likewise, the visibility of a transition is immediately determined, and thus for condition **C2** it suffices to examine each transition in turn. In fact, in order to obtain small ample sets, a single invisible transition is the ideal case.

In general, it is much more difficult however to check condition **C1**. First, this condition describes a property of ample sets in terms of the execution sequences of the full state-transition graph, and the principal aim of the reduction technique is to avoid constructing this graph in the first place. Furthermore, the execution sequences on which **C1** would have to be checked can extend arbitrarily far into the future, up to the occurrence of the first ample transition. In general, checking condition **C1** is at least as hard as checking reachability for the full state transition graph, as has been shown in [CGP99].

In practice, using an expensive algorithm that can verify condition **C1** for an arbitrarily chosen set of ample transitions could be quite expensive. Instead, partial order verifiers take advantage of the specific system structure to generate ample sets of transitions for which **C1** can be easily guaranteed to hold. In particular, the ample set selection becomes much easier in the typical case when the system is described as a composition of concurrent processes. We present practical conditions that can be used for concurrent processes with synchronous communication, a model which also forms the underlying control structure for timed automata, and discuss how the introduction of global data variables affects these conditions.

A system consists in this case of a set of *processes*, which are modeled as state-transition graphs. Each process may also have a set of *local variables* that can be changed only by transitions performed by that process. Control states and local variables form the *local state* of the process, and the product of the local states forms the *global state* of the system. A transition that only changes the control state and local variables of a process is called an *internal* transition.

In the synchronous communication model, the sender and the receiver coordinate, and the sending and receiving transitions occur simultaneously. This is the case, for example, in Communicating Sequential Processes [Hoa95] and in the rendezvous model of ADA. The sending and receiving transitions can therefore be considered as a common transition shared by the two processes. We call such a transition a *communication transition*. Simultaneous communication between more than two processes can be handled in the same way. Assume that all transitions in the system are either local or communication transitions.

Two local transitions, each belonging to a distinct process, are clearly independent, since the execution of each depends only on the local state of its process and produces changes only in its local state. Two local transitions enabled from the same state of one process are dependent, since executing one will lead to a different local state, from which the other transition is no longer enabled. Consequently, if at a given state, an ample set contains a local transition, it must contain all other enabled local transitions of the same process.

For communication transitions, the dependence relation is more complex. If a process P_i is at a communication point (from which a send or receive transition can be executed), the corresponding communication transition is said to be *locally enabled* by P_i. More precisely, a communication transition between two processes P_i and P_j is said to be *locally enabled* by P_i at state s if it can be executed from some state s' that has the same local state of P_i as s. The transition is only enabled globally when the communication partner of P_i is also at its corresponding communication point.

Locally enabled transitions must be considered in the computation of ample sets even if they are not globally enabled. Consider a process P_i which has two outgoing transitions at its current state: a local transition α and a locally enabled (but globally disabled) communication transition β with some other process P_j. Including only the enabled local transition in the ample set of the current state is not sufficient, since the communication transition (which is dependent on α, originating at the same state) may potentially become enabled if P_j reaches its communication point after executing some local transitions, which are not part of the ample set. This would contradict condition **C1**.

Consequently, if the transitions enabled at the current state s in some process P_i are included in the ample set for that state, so must be all enabled transitions of processes P_j whose communication transitions with P_i are locally enabled in P_i at state s. Taking the transitive closure of this operation, the following condition is obtained (cf. [Pel96b]):

Let *ample*(s) be the set of all transitions enabled at s in some set of processes $\mathcal{P}$ with the following property: No process $P_i \in \mathcal{P}$ has a communication transition locally enabled in P_i with a process outside of $\mathcal{P}$.

In practice, the rule is applied by first selecting a single process as a member of $\mathcal{P}$ and then repeatedly adding the processes that communicate with processes in $\mathcal{P}$. If $\mathcal{P}$ grows to include all processes, the state is fully expanded and no reduction is obtained at that state.

Consider now the case where the model is augmented to include global variables, which can be tested by boolean guards associated to transitions, and assigned as an effect of executing a transition. The dependence relation now has to take global variables into account. If $readv(\alpha)$ and $writev(\alpha)$ denote the sets of global variables written and read by a transition α, then two transitions α and β in separate processes are still independent if there is no read-write or write-write conflict between them, i.e., $readv(\alpha) \cap writev(\beta) = readv(\beta) \cap writev(alpha) = writev(\alpha) \cap writev(\beta) = \emptyset$ (cf. [God96]).

An ample set can be determined similarly as above by taking into account that shared variables are a form of communication. Thus, a transition disabled at the current state in some process P_i because its guard is not satisfied may become enabled if a global variable in its guard is modified by a transition in some other process P_j. Hence, one can choose as $ample(s)$ the set of all transitions enabled at s in some set of processes $\mathcal{P}$ with the following property: No process $P_i \in \mathcal{P}$ has a communication transition locally enabled in P_i with a process outside of P or has a transition whose guard reads a global variable written to by a process outside of $\mathcal{P}$.

The above conditions take a conservative approach to enforcing condition **C1**, specifically in identifying when a transition may become enabled by a transition from some other process. However, a more detailed analysis can be used to produce smaller ample sets. For example, it is possible to weaken the condition given above which selects a set of processes $\mathcal{P}$. It is safe for a process from $\mathcal{P}$ to have a locally enabled communication transition with a process outside $\mathcal{P}$ if it can be determined that this communication cannot actually take place in any state reachable from the current state.

However, checking that a transition is disabled in the future on any path starting from a given state is again as hard as the model checking problem itself. To avoid this problem, one can use an analysis procedure which is able to identify some of the transitions that can no longer become enabled starting from the current state, rather than all of them. To achieve this, one can perform a separate reachability analysis for each process, relying on the fact that a single process has a much smaller state space than the global system. For the synchronous communication case discussed above, one would check whether the matching communication transition can be reached in the other process starting from its local state, assuming conservatively that all communication transitions with other processes are enabled by those processes. In the case of systems containing data variables, it is possible to selectively or completely abstract away their values, and in the simplest case perform only a static analysis of the control flow graph of the process.

This analysis can be performed in a preliminary stage of the reduction algorithm, and the state transition graph may be annotated with information that allows both the selection of smaller ample sets, and their faster computation at run-time, thus increasing the performance of partial order reduction both in terms of memory requirements and execution time.

2.7 Other Partial Order Reduction Methods

The ample set approach to reduction, as well as the related methods that use stubborn or persistent sets generate a reduced model based on exploiting information about the structure of the system, about enabled, disabled and independent transitions. A different technique, the *sleep set* method suggested by Godefroid [God90] for detecting deadlocks exploits instead information about the past of the search.

One potential limitation of ample sets is that they have to be transitively closed with respect to dependency. This can lead to ample sets that contain pairs of independent transitions, simply because each of them is dependent on some other transition in the ample set. Since all transitions from an ample set are explored, the algorithm will be forced to consider both interleavings for two independent transitions, which is unnecessary and contrary to the initial purpose of reduction.

The sleep set approach addresses this problem by maintaining for each state s expanded by the algorithm, a set of transitions $\mathit{sleep}(s)$. This set contains the transitions which do *not* have to be explored from s. As a state is expanded, all transitions explored from it are added one by one to its sleep set. Part of these transitions are inherited by the successors of s as follows: If transition α is explored from state s, leading to state $s' = \alpha(s)$, all transitions $\beta \in \mathit{sleep}(s)$ which are independent of α are added to $\mathit{sleep}(s')$. This can be done because exploring α and then β has the same effect as exploring β followed by α, and β can be in $\mathit{sleep}(s)$ for two reasons. First, β did not need to be explored from s, and thus $\beta\alpha$ (and hence $\alpha\beta$) is also not needed. Second, β has been already explored from s, and in the process also the sequence $\beta\alpha$. Therefore, the equivalent sequence $\alpha\beta$ is no longer needed.

During the state space search, sleep sets for each state are stored. If a state is reached again during expansion, a new sleep set is calculated for it, and is compared with the previous value. If the old sleep set contains transitions that do not belong to the new sleep set, the node is expanded again with a sleep set which is the intersection of the new and the old sleep set. This ensures that if a state is reached from several states, enough successors are explored in all cases.

It has been shown [GW91] that the sleep set and persistent set reduction methods are orthogonal and hence their benefits can be combined.

A conceptually different approach to partial order reduction is the unfolding technique of McMillan [McM92, McM95]. This method is directly based on the partial order model of execution and has been originally defined in the context of Petri nets. In this approach, a structure of partially ordered local states is constructed, with the order between events representing the *causal order* of their execution. The unfolding algorithm generates a representation of the checked system which is sometimes called an *event structure*. It thus avoids generating the global states of the system altogether. The original unfolding algorithm was designed for deadlock detection. Subsequently, extensions of this algorithm were developed for checking different properties, e.g., by Esparza [Esp94]. The unfolding technique also stands at the basis of some partial order reduction approaches for time Petri nets, for instance in [Lil98, BF99].

Since partial order reduction is based on the assumption that a significant number of the execution traces of the system differ only in the ordering of transitions, it is not a technique which universally leads to good reduction results for all types of systems. Even for systems where partial order reduction is efficient, supplementary benefits can still be obtained by employing additional reduction techniques. However, since partial order reduction implies significant changes in the model checking algorithm, its applicability jointly with other methods does not always follow in a straightforward fashion.

Partial order can be combined with *on-the-fly* model checking [Kur94], a method in which the reduced state space is generated at the same time as the search for counterexamples that falsify the checked property. Employing this approach can result in significant space savings, since a counterexample may be found before the entire (reduced) state graph is generated [Pel94, Val90].

Often, concurrent systems contain several identical components. In this case, symmetry can provide stronger, supplementary reduction conditions. Partial order reduction and symmetry have been combined in [EJP97].

Symbolic model checking [BCM+90, McM93], which uses BDDs to efficiently store and manipulate sets of states, has had a significant impact on verification. Though mainly used for synchronous hardware systems, is performs well for asynchronous systems as well and is thus a natural candidate to combine with partial order reduction. One method for the joint use of these techniques was suggested in [ABH+97], using a reduction based on breadth first search [CP96]. A different approach to combining these methods, based on a static generation of the reduced model is suggested in [KLM+97, KLM+98]. This approach is presented in detail in the next section.

2.8 Static Partial Order Reduction

In the verification literature, partial order reduction has been used as a method for verifying mainly asynchronous concurrent systems, and has been traditionally implemented using explicit-state depth-first search. On the other hand, significant advances in alleviating the state-space explosion problem have been obtained using symbolic model checking [BCM+90, McM93], which uses an implicit representation of the state space. This method has shown significant benefits especially in verification of synchronous hardware systems. It appears natural to investigate whether the two methods can be combined, since this approach offers several potential benefits.

First, there is the potential of combining the efficiency gain of both methods: partial order reduction decreases the size of the search space, whereas symbolic techniques can further offer a compact representation with smaller amounts of memory and efficient space traversal algorithms. Second, the combined approach could exploit the advantages of the individual methods for systems that comprise components from the application domains of both methods: asynchronous hardware or mixed hardware-software systems. Furthermore, the approach could be extended to address different application domains.

The approach to partial order reduction presented here has been developed in a team working on a hardware-software co-verification project at Bell Laboratories. The targeted models were embedded systems, in which the software was written in SDL [SDL93]. The hardware was described in the automata-based language S/R of the model checker COSPAN [HK90], which can verify synchronous models and supports as an option BDD-based symbolic search. With this purpose in mind, a partial order reduction procedure was needed that satisfied the following goals:

- The reduction method should work efficiently for systems that are composed of both hardware and software. In particular, it should be usable in conjunction with BDDs and symbolic model checking.

- The reduction algorithm should be independent of the type of search (e.g., depth-first or breadth-first search).

- The reduction should be performed as much as possible during the generation of the system model.

- The reduction procedure should be adaptable to existing model checking tools without requiring changes to their search engines.

The last requirement in particular was important not only in the initial setting of this work, where an existing model checker had to be used as backend, but also in general. Since both partial order reduction algorithms and model checking engines are quite complex, an approach which completely separates reduction from model checking greatly increases the ease of applying reduction, as well as the possibility of combining it with other optimizations brought to the model checking engine.

The partial order reduction approach of selecting a subset of the enabled actions from each state to generate a smaller model is not inherently incompatible with a symbolic BDD-based search. However, model checkers that incorporate partial order reduction have so far used mostly an explicit state depth-first search, since this approach is suggested by the cycle-closing condition **C3**. Recall that a transition which is enabled in every state of a cycle has to belong to the ample set of some state of that cycle in order for the reduction to be correct. The stack maintained by a depth-first search provides an easy means to check this condition.

Alternate means to ensure this reduction condition have been suggested first by Holzmann and Peled [HP94]. They describe a static implementation of the reduction conditions in the model checker SPIN [Hol92]. However, despite being static, this approach required significant changes to the code of the SPIN model checker in order to control its backtracking mechanism. Subsequently, Chou and Peled [CP96], as a by-product to giving a mechanized proof of the partial order reduction conditions, showed that the cycle-closing condition could also be used with breadth-first search.

A first method for combining partial order reduction and symbolic model checking using BDDs was given in [ABH+97]. This solution uses the set of reached states as history and is based on a conservative approximation of when a cycle may be closed during a breadth-first search. Essentially, when an edge connects a node to another node that is at the same or a lower level in the breadth-first search, it is assumed to close a cycle.

The *static partial order reduction* approach presented here is different and more general in that all the information needed for performing the partial order reduction is obtained during a compilation of the system model. The partial order reduction step is effectively a preprocessing phase that takes a system description and modifies it such that only a reduced number of transitions are enabled at each state, corresponding precisely to an ample set. The resulting model is still described in the same input language as the original model and can be used as an input for the model checker without requiring any changes to it. This is in contrast to usual partial order reduction algorithms, which are applied on the internal representation of the system used by the model checker and interact with its search algorithms. It is precisely this separation of the reduction step that allows the combination of partial order reduction with BDD-based algorithms and, in general, with any optimization technique applied by the model checker.

2.8.1 A Modified Cycle Closing Condition

The cycle closing condition **C3** guarantees that a transition which is enabled in all states of a cycle is eventually chosen as part of an ample set. In practice, a slightly stronger condition is used, which states that at least one state on every cycle in the reduced state graph is fully expanded. This formulation clearly implies **C3**, since a transition which is continuously enabled along a cycle will be explored at the state that is fully expanded.

Typically, **C3** is ensured during depth-first search by examining the successors of all transitions that make up a candidate for an ample set. If any of these states has not been completely explored yet (i.e., is still on the search stack), the chosen set of transitions cannot be an ample set. Another candidate set has to be found at that state, or the state has to be fully expanded.

In devising a new means to enforce the cycle closing condition, we first observe that both **C2** and **C3** limit the extent of the reduction: they define cases where a state has to be fully expanded. Moreover, condition **C2** can help to ensure **C3**: on a cycle which contains a visible transition, **C2** guarantees that the originating state of that transition is fully expanded, and hence the cycle also satisfies **C3**. This observation suggests that **C2** and **C3** can be combined into a single condition **C2'**:

C2' *There exists a set of transitions T_{s}, which includes all visible transitions, such that any cycle in the reduced state space contains a transition from T_{s}. If ample$(s) \cap T_{\mathrm{s}} \neq \emptyset$, then ample$(s) =$ enabled(s).*

In other words, any cycle in the state graph of M' must execute at least one transition from T_{s}, and the originating states of any transitions in T_{s} are fully expanded. The transitions in T_{s} have been called *sticky transitions*, since they "stick" to all other transitions which are enabled at the same state and force their exploration.

We have seen in Section 2.6 how to ensure condition **C1** statically for systems composed of communicating processes. It remains to devise a procedure that determines a suitable set T_{s} of transitions which breaks all cycles in the reduced state space. This cannot be done directly, since the reduced state space itself depends on which transitions are chosen for exploration, and implicitly on T_{s}. However, the problem can be reduced to a simpler one, by observing that in a system composed of multiple processes, each cycle in the global state space projects to a cycle in each of the component processes. Conversely, a set of sticky transitions that breaks each local cycle is also guaranteed to break each global cycle. Thus, it is sufficient that the removal of all sticky transitions leave all component processes acyclic, a condition which can be ensured statically and locally, without constructing the state space of either the full or the reduced model.

Thus, condition **C2'** can be strengthened to the following formulation:

C2" *There exists a set of sticky transitions T_{s} which includes all visible transitions, such that each local cycle of a component process contains at least one sticky transition. If ample(s) includes a sticky transition, then s is fully expanded.*

Since a sticky transition forces a state to be fully expanded, it follows that fewer sticky transitions will result in a better reduction. It is important therefore to generate a small set T_{s} of sticky transitions. However, as with the selection of ample sets, the actual efficiency of the reduction depends on which transition are chosen to be sticky, and not solely on the number of transitions. Likewise, it is important that the set of sticky transitions be generated with a small overhead.

Assuming that a set of sticky transitions has been selected, a useful reduction strategy is to attempt at each state to find an ample set without including a sticky transition, in order to postpone the full expansion of a state as much as possible. Eventually, a sticky transition has to be selected, since otherwise no cycle can be closed, and the current state is expanded completely. However, it is likely that by giving priority to non-sticky transitions, the set of enabled transitions at that state contains many sticky transitions which have been delayed so far. Thus, rather than having each of them forcing the full expansion of a *different* state, only one state needs to be expanded, and the effect of having many sticky transitions is compensated to some extent.

Even with this heuristic, it is beneficial to generate a small set of sticky transitions in the first place. The next section describes how this can be done by analyzing the effects of transitions and their dependencies to determine potential cycles in the state space. The procedure given in this section marks transitions in order to satisfy the weaker condition **C2'**, resorting to the stronger condition **C2"** only when no optimizations can be made.

2.8.2 Determining Sticky Transitions

We examine the common case of a system with a control structure defined by parallel processes, and a set of data variables. Consider a finite state system composed of processes $P_1, P_2, \cdots, P_n$, each of which is described as a state-transition graph, also called process control graph. In addition, the system may contain a set V of variables which may be either global or local to a process, and whose value may be changed by the transitions in the system. The state of the system is thus composed of the control state of each process, together with the values of the variables.

Let T_{v} be the set of visible transitions. Following condition **C2"**, it suffices to find a set of transitions $T_{\mathrm{s}} \supseteq T_{\mathrm{v}}$, such that all process control graphs become acyclic when the transitions from T_{s} are removed. Here, the removal of a transition which changes the state in several processes means the removal of all edges that are projections of the transition in the process control graphs. After first removing the visible transitions, the remainder of each process control graph can be made acyclic in linear time by removing all back edges in a depth-first search. Yet, in the worst case, up to half of the edges need to be removed, whereas our goal is to keep this set small. For the general case, a somewhat better bound on the number of edges is presented in [BS97], and [ELS93] gives a simple heuristic algorithm to find a small set of such edges in linear time. In the following, we exploit the semantic structure of the system in order to obtain a small set of sticky transitions.

As a preliminary observation, the projection of any global cycle onto a process control graph has to belong to some strongly connected component of the control graph. Thus, any transitions whose projections do not belong to a strongly connected component can be ignored in the following analysis.

The key observation is that a cycle in the state space of the system has not only to restore the control point of each process, but also the value of each data variable. Assume that an ordering relation $\prec_v$ is defined on the domain of a variable v. Then, if a cycle contains a transition that increases the value of v (with respect to $\prec_v$), this must be compensated by a transition which decreases the value of v.

Consequently, we examine the effect of a transition α on a variable v. If it can be established statically that all executions of α increment (or, respectively, decrement) v, we denote $\mathit{effect}(\alpha, v) = +$, and, respectively $\mathit{effect}(\alpha, v) = -$. If α always leaves v unmodified, we denote $\mathit{effect}(\alpha, v) = 0$. Finally, if the effect of α varies from one execution to another, or cannot be determined statically, we denote $\mathit{effect}(\alpha, v) = \star$.

If $\mathit{effect}(\alpha, v) = +$, define $\mathit{Compensate}(\alpha, v) = \{\beta \mid \mathit{effect}(\beta, v) \in \{-, \star\}\}$ as the set of all transitions whose effect on v is potentially opposite to that of α, and likewise $\mathit{Compensate}(\alpha, v) = \{\beta \mid \mathit{effect}(\beta, v) \in \{+, \star\}\}$ for $\mathit{effect}(\alpha, v) = -$. It follows that if a cycle contains a transition α, it also has to contain at least one transition from $\mathit{Compensate}(\alpha, v)$ in order to restore the value of v.

The information about the effects of transitions on variables can be used to remove additional edges from a process control graph, without having to mark them as sticky, thus reducing the number of sticky transitions needed to break all its cycles. First, let T_u be the set of *uncompensated* transitions α, for which there exists a variable v such that $\mathit{Compensate}(\alpha, v) = \emptyset$. Then α cannot belong to any global cycle in the system state space, since no other transition can restore the value of v. Thus, the transitions in T_u can be removed from all process control graphs.

Second, suppose that *Handled* is the set of transitions for which we already know that any global cycle containing a transition from this set will also contain a sticky transition. For instance, this is trivially true if *Handled* is the set of sticky transitions selected so far; ultimately, we want all transitions to be in *Handled*, which would guarantee **C2'**. For a transition α, we define the predicate $\mathit{Covered}(\alpha, \mathit{Handled}) = \exists v \in V \,.\, \mathit{Compensate}(\alpha, v) \subseteq \mathit{Handled}$. Then, any cycle which contains a transition α covered by *Handled* will also contain a transition from *Handled* and thus a sticky transition. Again, α can be removed from all process control graphs since all potential cycles containing α already contain sticky transitions.

The notions and properties established so far lead to the following algorithm for computing a set T_{s} of sticky transitions, given in Figure 2.5:

remove T_{u} and T_{v} from all process control graphs
$\mathit{Handled} = T_{\mathrm{s}} = T_{\mathrm{v}}$
for all strongly connected components C of process control graphs
  $C' = C$
  for all transitions α in C
    if $\mathit{Covered}(\alpha, \mathit{Handled})$ then $C' = C' \setminus \{\alpha\}$
  $T_{\mathrm{s}} = T_{\mathrm{s}} \cup \mathit{BackEdges}(C')$
  $\mathit{Handled} = \mathit{Handled} \cup \mathit{transitions}(C)$
end

Figure 2.5: Algorithm for computing sticky sets

Initially, the visible and uncompensated transitions are removed from all process control graphs, and the sets of handled and sticky transitions are initialized with all visible transitions. Next, we consider in turn the strongly connected components of all process control graphs. First, we compute the set of transitions which always belong to cycles containing transitions handled so far, and remove them from the strongly connected component C. Next, all back edges of the resulting graph C' are found and marked as sticky. At these point, any cycle with a transition from C contains a sticky transition, and the transitions from C can thus be included in *Handled*. The algorithm completes when all strongly connected components are analyzed.

2.8.3 Experimental Evaluation

Conceptually, static reduction cannot benefit from all the information which is available to a dynamic reduction algorithm. In our framework, determining when a cycle can be closed in the global state space is based on a conservative analysis of the local cycles in all component processes. It is possible that transitions are marked as sticky in order to break global cycles that never actually occur during the execution of the system. In comparison, traditional dynamic reduction techniques can use complete state information as well as the history of the search in order to guide the selection of ample transitions.

A first evaluation of static partial order reduction is reported in [KLM+98] after implementing static partial order reduction in a compiler from SDL to S/R. Several typical benchmarks exhibiting concurrency have been analyzed, including a concurrent sorting algorithm, a leader election protocol and an asynchronous tree arbiter, all parameterized with various numbers of processes. The results reported in [KLM+98] are presented here in Table 2.1. A comparison of static partial order reduction with the traditional dynamic algorithm, using explicit state search in both cases, has shown similar performance in terms of the resulting state space. Thus, in practice the limited information available to a static technique does not lead to performance drawbacks if a good algorithm for selecting sticky transitions is employed.

Experiments	Number of states (no reduction)	(static reduction)
Concurrent sort, $N = 2$	191	66
Concurrent sort, $N = 3$	4903	553
Concurrent sort, $N = 4$	135329	4163
Concurrent sort, $N = 5$	3940720	29541
Leader election, $N = 2$	383	107
Leader election, $N = 3$	11068	490
Leader election, $N = 4$	537897	3021
Leader election, $N = 5$	26523000	21856
Tree arbiter, $N = 2$	73	48
Tree arbiter, $N = 4$	18247	4916
Tree arbiter, $N = 6$	3272700	358352

Table 2.1: Experimental Results

The same examples show, as expected, that for small examples, a symbolic search on the statically reduced models is more expensive than a traditional explicit search with partial order reduction, and also more expensive than a symbolic search with no reduction at all. This is not an intrinsic property of static partial order reduction. It simply relates to the fact that for systems which are trivial to analyze, using both reduction and symbolic representation are optimizations whose overhead does not pay off. The characteristic profile of static reduction emerges as systems with a larger number of processes are analyzed. As the state space increases, the symbolic search with partial order reduction performs better than the symbolic search without reduction. The combined use of both techniques enables the verification of systems which are too large to be handled by either method alone.

Concluding, the main advantage of static partial order reduction is that it can be performed as a preprocessing phase prior to verification and hence is completely separate from the model checking engine. This enables a more modular construction of a model checking environment, and specifically, the use of partial order reduction with any existing model checker. As a resulting benefit, the performance advantages obtained by reducing the state space can be combined with optimizations specific to the target model checker. In particular, results show that partial order reduction and symbolic model checking, both techniques which have been long used independently, can be combined, extending the limits of automatic verification.

Chapter 3

Partial Order Reduction for Timed Automata

3.1 Introduction

Timed automata, originally defined by Alur and Dill [AD90] are a widespread model for continuous-time systems. They are extensions of finite state automata with constraints on timing behavior. The underlying state-transition graph of a timed automaton is augmented with a set of continuous-time clocks. Transitions (and in some variants of the model, states) are labeled with clock constraints that restrict the executions of the system in time.

The introduction of continuous time significantly increases the complexity of the verification problem. The state space of timed automata is inherently uncountable, but can be reduced to a finite model. The first such method is the *region graph construction* of [ACD90], however, its complexity is exponential in the number of clocks and of the largest constant in the model. A different construction, the so-called *zone automaton* model is based on performing computations on clock constraints [Dil89]. Though its theoretical worst-case complexity is not lower, it has proved efficient in practice, and has been used by a number of real-time verifiers [NSY92, Won94, LPW95]. Among the systems that have been successfully modeled and verified using timed automata are asynchronous circuits, communication protocols, automotive and manufacturing systems.

In this section, we show how to improve the efficiency of model checking for a system composed of timed automata using partial order reduction.

First, we describe our model and related approaches to partial order reduction, including a local-time semantics [BJLW98]. In the remainder of the section, we extend this approach. We show that the local-time semantics can be modified to preserve the truth value of specifications in a timed extension of next-time free LTL. We give a constructive proof that the resulting model accepts a finite quotient, by presenting a condition for the equivalence of two local-time zones, which forms the basis for a state-space search algorithm. We discuss how the representation of time zones in the local-time semantics can be improved, and how to select ample sets of transitions for partial order reduction. The method leads to efficiency improvements on two counts: the local-time model has as effect the generation of fewer time zones, whereas partial order reduction leads to the exploration of fewer control states.

3.2 Timed Automata

3.2.1 Definition

Timed automata use a global and continuous notion of time. The clocks used to describe a timed automaton are real-time variables that evolve at the same rate:

Definition 1 *(Clock; clock assignment) A* clock *is a variable over the set $\mathbb{R}^+$ of nonnegative reals. Given a set of clocks $C = \{x_1, x_2, \ldots, x_n\}$, a* clock assignment *is a function $v : C \to \mathbb{R}^+$ which assigns each clock a nonnegative real value. The set of clock assignments over C is denoted by $\mathcal{V}(C)$.*

Definition 2 *(Clock constraint) Let C be a finite set of clocks. A* clock constraint *over C is a formula defined by the following grammar:*

$$\psi ::= \text{true} \mid c \prec x \mid x \prec c \mid x - y \prec c \mid \psi \wedge \psi$$

where $x, y \in C$ are clocks, $c \in \mathbb{Z}$ is an integer, and $\prec \in \{<, \leq\}$. The first four terms on the right hand side are atomic clock constraints*. The set of clock constraints over C is denoted by $\mathcal{B}(C)$.*

Since a constraint is a conjunction of elementary inequalities, it always represents a convex region in the space of clocks.

Definition 3 *(Timed Automaton) A timed automaton is represented by a tuple $A = (S, S^0, C, E, I, \mu)$, where*

- *S is a finite set of nodes (also called control states or locations)*
- *$S^0 \subseteq S$ is the set of initial nodes*
- *C is a finite set of real-valued non-negative clocks*
- *$E \subseteq S \times \mathcal{B}(C) \times 2^C \times S$ is a finite set of edges. Each edge $e = \langle s, \psi, R, s' \rangle$ has a clock constraint ψ called* enabling condition *and a set $R \subseteq C$ of clocks that are reset on traversing the edge*
- *$I : S \rightarrow \mathcal{B}(C)$ associates each node with a clock constraint called the* invariant condition
- *$\mu : S \rightarrow \mathcal{P}(AP)$ is a function labeling each node with atomic propositions from a set AP.*

The clocks of a timed automaton allow the expression of timing properties. A clock that is reset by a transition can be subsequently used in a timing constraint, allowing a reference to the timepoint when that transition was taken. An enabling condition constrains the execution of a transition, without forcing it to be taken. An invariant condition, on the other hand, allows an automaton to stay at a certain state only as long as the constraint is satisfied.

We reason about systems composed of several timed automata. We define a general parallel composition parameterized by a synchronization function:

Definition 4 *(Network of timed automata) Consider the timed automata $A_i = (S_i, S_i^0, C_i, E_i, I_i, \mu_i)$ for $1 \leq i \leq n$ and a synchronization function $f : \prod_{i=1}^{n}(E_i \cup \{\epsilon\}) \rightarrow \{0, 1\}$, where ϵ is a symbol denoting a null edge. The network of timed automata $A_1 \parallel A_2 \parallel \ldots \parallel A_n$ is a timed automaton $A = (S, S^0, C, E, I, \mu)$, where:*

- *$S = S_1 \times S_2 \times \ldots \times S_n$*
- *$S^0 = S_1^0 \times S_2^0 \times \ldots \times S_n^0$*
- *$C = C_1 \cup C_2 \cup \ldots \cup C_n$ (it is assumed that $C_i \cap C_j = \emptyset$, for $i \neq j$)*
- *E contains a family of edges (called a* transition*) for each tuple of edges with $f(e_1, \ldots, e_n) = 1$. The edges of transition a have $\psi = \bigwedge_{i \in active(a)} \psi_i$ and $R = \bigcup_{i \in active(a)} R_i$, where $active(a) = \{i \mid e_i \neq \epsilon\}$, and $e_i =$*

$\langle s_i, \psi_i, R_i, s'_i \rangle$. The components s_i and s'_i of the edge endpoints are given by e_i for $i \in active(a)$ and are arbitrary, but pairwise equal ($s_j = s'_j \in S_j$) for $j \notin active(a)$.

- $I(s) = \bigwedge_{i=1}^{n} I_i(s_i)$
- $\mu(s) = \bigcup_{i=1}^{n} \mu(s_i)$ *(it is assumed that the sets of atomic propositions AP_i are pairwise disjoint).*

In other words, a transition in the network of automata corresponds to the synchronous traversal of edges in several of the component automata. The synchronization function determines which automata execute (the *active set* for the given transition) and which ones remain at their local state (those for which $e_i = \epsilon$). A transition whose active set contains more than one automaton is called a *synchronization transition*, otherwise it is called *local*. The set of transitions is denoted by $\mathcal{T}$.

The above definition allows the modeling of many common synchronization paradigms. For instance, to model CCS-type communication one can assume that the edges of the individual automata are labeled by action symbols, and choose a synchronization function which has value 1 for tuples which contain a pair of matching communication transitions, and ϵ-transitions otherwise. Multi-way synchronization can be modeled in a similar fashion.

3.2.2 Semantics

Two basic operations are defined on clocks: incrementing and resetting. If $v \in \mathcal{V}(C)$ is a clock assignment and $d \in \mathbb{R}^+$ a nonnegative real number, then $v + d$ is the assignment given by $(v + d)(x) = v(x) + d$, for each clock $x \in C$.

Given a set of clocks $R \in C$, $v[R \mapsto 0]$ denotes the clock assignment that agrees with v for all clocks in $C \setminus R$, and is zero for all clocks $x \in R$. For a clock constraint $\psi \in \mathcal{B}(C)$ and a clock assignment $v \in \mathcal{V}(C)$, denote by $\psi(v)$ the truth value of ψ for the clock assignment v.

The semantics of a timed automaton can be defined as follows:

Definition 5 *(Semantics) A model of a timed automaton is a state-transition graph $\mathcal{S}(A) = (\Sigma, \Sigma^0, \rightarrow)$, where*

- $\Sigma = \{(s, v) \mid I(s)(v)\}$ *is the set of states whose clock assignment satisfies the node invariant*

- $\Sigma^0 = \{(s^0, 0_C) \mid s^0 \in S^0\}$ *is the set of initial states, where* 0_C *is the clock assignment with* $0_C(x) = 0$*, for all* $x \in C$

- $\rightarrow$ *is the transition relation defined as the union of* delay (or time) transitions $\overset{d}{\rightsquigarrow}$ *and* action (or event) transitions $\xrightarrow{a}$*, as follows:*

  $(s, v) \overset{d}{\rightsquigarrow} (s, v+d)$ *if* $d \in \mathbb{R}^+$ *and* $I(s)(v+d')$ *holds for all* $d' \in [0, d]$

  $(s, v) \xrightarrow{a} (s', v[R \mapsto 0])$ *for* $a \in \mathcal{T}$ *if there exists an edge* $e = (s, \psi, R, s') \in a$*, such that* $\psi(v)$ *is true and* $I(s')(v[R \mapsto 0])$ *holds.*

In other words, a timed state in the model is a pair consisting of a node and a clock assignment that satisfies the location invariant. Transitions are of two types: a *delay transition* is caused by the elapsing of time in the same control state, if the invariant condition remains satisfied throughout. An *action transition* can be executed if the clock assignment satisfies the enabling condition. The clocks in the set R associated with the edge are reset, whereas the other clocks maintain their value (the transition is instantaneous). The location invariant has to be satisfied in the resulting state.

Any timed automaton can be transformed into an equivalent automaton whose state invariants only impose upper bounds on clocks, i.e., are composed of constraints of the form $x_i \prec c$. This is true because constraints of the type $x_i - x_j \prec c$ or $c \prec x_i$ cannot be falsified by the passage of time and will remain true in a control state if they were true upon entering it:

- $x_i - x_j \prec c \Leftrightarrow (x_i + d) - (x_j + d) \prec c$, for all $d \in \mathbb{R}$

- $c \prec x_i \Rightarrow c \prec x_i + d$, for all $d \in \mathbb{R}^+$

Therefore, it suffices to have these two types of constraints guaranteed at the execution time of any transition entering the given location. Specifically, consider the edge $e = \langle s, a, \psi, R, s' \rangle$. The conjuncts discussed above can be incorporated into the enabling condition of the edge by splitting the invariant into $I(s') = I_{edge} \wedge I_{node}$ and rewriting the enabling condition for an arbitrary clock valuation v as follows (we write $E[R \mapsto 0]$ for the clock expression E in which the clocks belonging to R are replaced with 0):

$$\begin{aligned} & \psi(v) \wedge (I_{edge} \wedge I_{node})(v[R \mapsto 0]) \\ = \quad & \psi(v) \wedge I_{edge}(v[R \mapsto 0]) \wedge I_{node}(v[R \mapsto 0]) \\ = \quad & \psi(v) \wedge (I_{edge}[R \mapsto 0])(v) \wedge I_{node}(v[R \mapsto 0]) \\ = \quad & (\psi \wedge I_{edge}[R \mapsto 0])(v) \wedge I_{node}(v[R \mapsto 0]) \end{aligned}$$

The new enabling condition is therefore $\psi \wedge I_{edge}[R \mapsto 0]$, and the new invariant is I_{node}. We assume in the following that all timed automata have been transformed to observe this property.

In addition, since all clock constraints are conjunctions and therefore convex, the invariant holds in all intermediate states of a delay transition if it holds at the endpoints. Assuming that the automaton has been transformed as above (with invariants enforcing only upper bounds on clocks), the invariant only needs to be checked in the resulting state:

$$(s, v) \overset{d}{\rightsquigarrow} (s, v + d) \text{ if } d \in \mathbb{R}^+, \text{ and } I(s)(v + d) \text{ holds}$$

In our analysis of the system, we will observe its *execution traces*, defined as follows:

Definition 6 *(Execution trace) An* execution trace *of a timed automaton is a finite or infinite sequence $\sigma = (s^0, 0_C) \rightarrow (s^1, v^1) \rightarrow \ldots \rightarrow (s^k, v^k) \ldots$ starting from an initial location $s^0 \in S^0$.*

We denote by $\sigma(k) = (s^k, v^k)$ the k^{th} state on the trace σ, by σ_k the finite prefix of σ ending at (s^k, v^k) and by σ^k the suffix of σ starting at the same state.

3.3 The model checking problem

Verification of timed automata models has been studied in several contexts. The Kronos model checker [NSY92] is built for timed versions of CTL and of the modal μ-calculus [HNSY92]. Uppaal [LPW95] accepts a logic for safety and bounded liveness properties which can reference values of clocks. However, partial order approaches have been so far restricted to less expressive properties: Pagani [Pag96, Pag97] addresses the problem of deadlock detection, whereas Bengtsson et al. [BJLW98] check local reachability within one process.

We propose to use an extension of LTL that also allows atomic time constraints to be used in place of atomic propositions. The logic is inspired from the timed temporal logic for nets (TNL) of Yoneda et al. [YSSC93], which was defined and used in the context of Petri nets. The inclusion of atomic time constraints in the logic allows the real-time aspect of the system to be captured: comparing the difference of two clocks that are reset by two transitions permits reasoning about the time separation of the corresponding two events.

The formulas of our logic, which we will call LTL$_\Delta$, are defined as follows:

- an atomic formula φ_a is an atomic proposition $p \in AP$ or an atomic clock constraint $x - y \prec c$, where $x, y \in C$, $c \in Z$ and $\prec \in \{<, \leq\}$
- if φ_1 and φ_2 are formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \, \mathbf{U} \, \varphi_2$ are formulas.

To maintain the correspondence with untimed systems, we will define the semantics of LTL$_\Delta$ only for infinite execution paths on which time diverges, i.e., for which the sum of delays is infinite. This means disallowing *Zeno* paths, on which an infinite number of transitions is taken in a finite amount of time.

Definition 7 *(Semantics of logic) For an infinite, time-divergent execution trace $\sigma = (s^0, v^0) \to (s^1, v^1) \to \ldots \to (s^k, v^k) \to \ldots$, the semantics of an LTL$_\Delta$ formula is defined as follows:*

- $(s, v) \models p$ *iff* $p \in \mu(s)$
- $(s, v) \models x - y \prec c$ *iff* $v(x) - v(y) \prec c$.
- $\sigma \models \varphi_a$ *iff* φ_a *is an atomic formula and* $(s^0, v^0) \models \varphi_a$
- $\sigma \models \neg\varphi$ *iff not* $\sigma \models \varphi$
- $\sigma \models \varphi_1 \wedge \varphi_2$ *iff* $\sigma \models \varphi_1$ *and* $\sigma \models \varphi_2$
- $\sigma \models \varphi_1 \, \mathbf{U} \, \varphi_2$ *iff there* $\exists k \geq 0$ *such that* $\sigma^k \models \varphi_2$ *and* $\sigma^j \models \varphi_1$ *for all* $0 \leq j < k$

For a delay transition $(s, v) \stackrel{d}{\rightsquigarrow} (s, v + d)$, the automaton passes through the continuous sequence of intermediate states $(s, v + d')$ with $0 \leq d' \leq d$. Since both control state and clock differences are preserved in each of these intermediate states (for any two clocks $x, y \in C$ we have $v(x) - v(y) \prec c \Leftrightarrow (v+d')(x) - (v+d')(y) \prec c$, for all $d' \in [0, d]$), they have the same truth value for all atomic subformulas of a formula in LTL$_\Delta$. Thus, the given semantics of LTL$_\Delta$ (considering truth values at transition endpoints) corresponds to the intuitive meaning of continuous execution.

3.3.1 Effect of transition interleavings

The traditional reachability analysis algorithm for networks of timed automata explores all possible transition interleavings among the individual components. A partial order method would select a representative from each set of equivalent interleavings, exploring only a reduced portion of the state space. However, in the given model, clocks advance simultaneously in all automata, causing dependencies between transitions in individual components. Different interleavings may therefore produce different assignments to clock values. The following simple example illustrates this problem.

Consider the system of two automata in Figure 3.1. The initial state is given by $\langle (r_1, s_1), x = y \rangle$. From there, if transition a is executed first, the system reaches the state $\langle (r_2, s_1), x \leq y \rangle$ (clock x is reset on executing the transition, so its value will not exceed that of y). Subsequently, on executing b, clock y is reset, resulting in the state $\langle (r_2, s_2), x \geq y \rangle$. On the other hand, if b is executed first, the system reaches the state $\langle (r_1, s_2), x \geq y \rangle$, and then, after executing a, the state $\langle (r_2, s_2), x \leq y \rangle$.

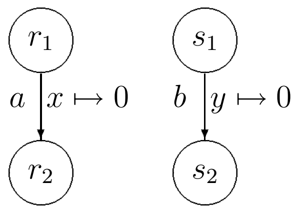

Figure 3.1: Effect of transition interleavings

The two interleavings lead to the same control state, but to different clock zones and therefore different states in the zone automaton. Thus, the two transitions are not independent and the partial order reduction techniques developed for untimed systems cannot be applied in this case.

Note that after executing both transitions in either interleaving, the system state belongs to the union of the two zones, $\langle (r_2, s_2), x \geq y \vee x \leq y \rangle = \langle (r_2, s_2), \mathit{true} \rangle$. If the verified property is insensitive to the relative ordering of x and y, the two interleavings are still equivalent. In such cases, a partial order reduction procedure should produce a zone containing the timed states reachable by all transition interleavings, while exploring only one interleaving, and thus fewer states.

3.4 Related Work

Partial order reduction techniques have been first investigated in the context of timed Petri nets by Yoneda, Schlingloff et al. [YSSC93, YS97]. Their model has earliest and latest firing times associated with transitions and is thus less expressive than timed automata. For specifications, they defined an expressive timed extension of next-time free LTL, which is the source for our logic LTL_Δ. Their reduction method is based on stubborn sets, and they show that only the transitions in the reduced set chosen for exploration need to be ordered in time. Lilius [Lil98] suggests an improvement where the transition firing order need not be stored in the timed state. This reduces the complexity of the timing constraints and the branching in the generated graph. He also shows how the unfolding prefix of McMillan [McM95] can be used to select reduced sets of transitions for exploration.

For timed automata, the first approach to partial order reduction appears to be the work of Pagani [Pag96, Pag97]. Her thesis analyzes the dependence relation between transitions in a network of timed automata and the cases when partial order reduction can be applied. We briefly present the main results, in order to illustrate the issues that restrict the application of partial order reduction for this timed model. Pagani observes that there are the following elementary dependence cases among action and delay transitions:

- $\stackrel{t}{\leadsto}$ disables $\xrightarrow{a}$ if there exists a state $\langle s, v\rangle$ and $t \in \mathbb{R}^+$ such that $\xrightarrow{a}$ is enabled in $\langle s, v\rangle$ but not in $\langle s, v+t\rangle$.

- $\stackrel{t}{\leadsto}$ enables $\xrightarrow{a}$ if there exists a state $\langle s, v\rangle$ and $t \in \mathbb{R}^+$ such that $\xrightarrow{a}$ is not enabled in $\langle s, v\rangle$ but is enabled in $\langle s, v+t\rangle$.

- $\xrightarrow{a}$ disables $\stackrel{t}{\leadsto}$ if there exists a state $\langle s, v\rangle$ and $t \in \mathbb{R}^+$ such that time t can pass in $\langle s, v\rangle$ but not in $\langle s', v\rangle$ (where $s \xrightarrow{a} s'$).

- $\xrightarrow{a}$ enables $\stackrel{t}{\leadsto}$ if there exists a state $\langle s, v\rangle$ and $t \in \mathbb{R}^+$ such that time t cannot pass in $\langle s, v\rangle$ but can pass in $\langle s', v\rangle$ (where $s \xrightarrow{a} s'$).

The above elementary relations induce dependence relations between the transitions of the zone automaton, which model the joint effect of action and delay transitions. Pagani identifies seven dependence cases for transitions occurring in different component automata. One of these cases can be dismissed by using the weaker notion of independence which allows transitions to enable one another. The remaining dependences cases are as follows:

1. $\overset{t}{\leadsto}$ enables $\overset{a}{\rightarrow}$ and $\overset{t}{\leadsto}$ disables $\overset{b}{\rightarrow}$.

2. $\overset{t}{\leadsto}$ enables $\overset{a}{\rightarrow}$ and $\overset{b}{\rightarrow}$ disables $\overset{t}{\leadsto}$.

3. $R_a \neq \emptyset$ and $R_b \neq \emptyset$.

4. $R_a \neq \emptyset$ and $\overset{t}{\leadsto}$ enables $\overset{b}{\rightarrow}$.

5. $R_a \neq \emptyset$ and $\overset{t}{\leadsto}$ disables $\overset{b}{\rightarrow}$.

6. $R_a \neq \emptyset$ and $\overset{b}{\rightarrow}$ disables $\overset{t}{\leadsto}$.

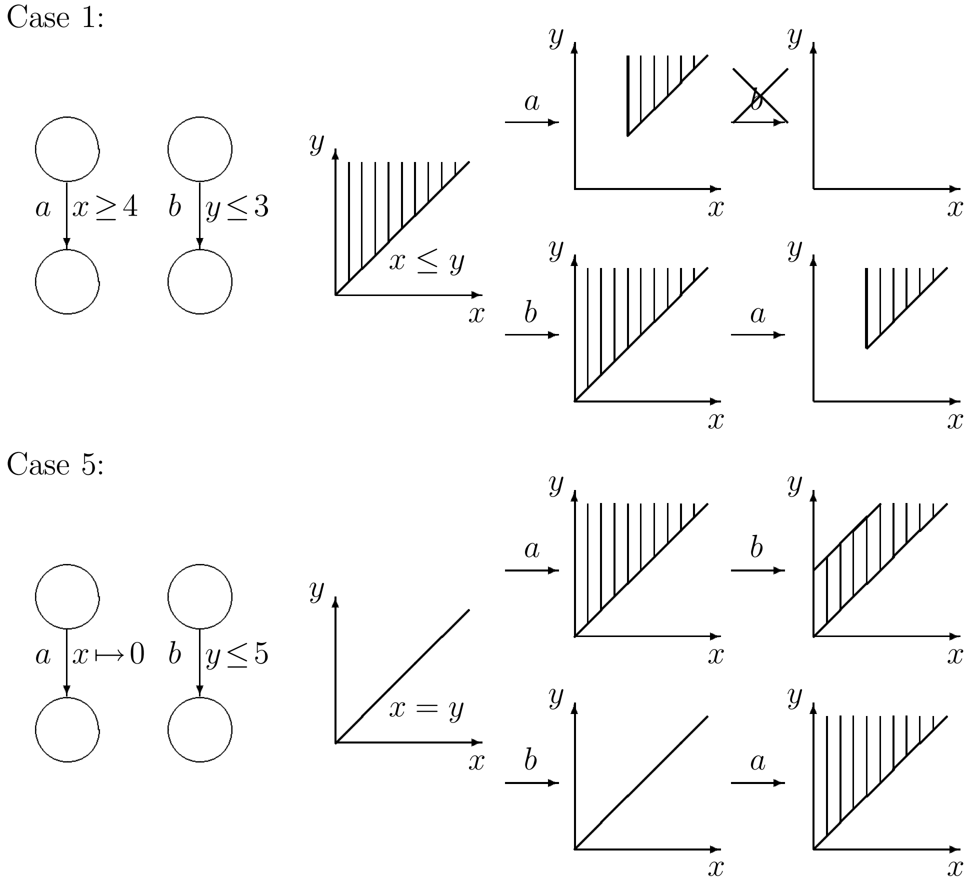

Figure 3.2: Dependent transitions in the zone automaton

The first two cases in the definition above violate the enabledness condition of independence, whereas the last four cases violate the commutativity requirement. For instance, in case (1) it may be possible to execute $\xrightarrow{b}$ and after some time $\xrightarrow{a}$, but if $\xrightarrow{a}$ is executed first, the amount of time that needs to pass may disable $\xrightarrow{b}$. In case (3) (discussed in the previous section), the relative value of the clocks in R_a and R_b will depend on the order in which they were reset, i.e., on the execution order of $\xrightarrow{a}$ and $\xrightarrow{b}$. In case (5), if $\xrightarrow{b}$ is executed after $\xrightarrow{a}$, the relative difference between x and y is limited as a consequence. Thus, the set of timed states reached after one interleaving includes all states reached when the other interleaving is executed. Overall, the analysis of Pagani shows that the number of independent transitions is significantly reduced by timing.

The approach of Dams et al. [DGKK98] eliminates some of these dependencies by using the asymmetric notion of *covering* instead of independence. A transition b *covers* a transition a if all timed states reached by executing $\xrightarrow{a}$ followed by $\xrightarrow{b}$ can be reached by executing $\xrightarrow{b}$ and then $\xrightarrow{a}$, such as in case (5) depicted above. Thus, to account for all the states, it is still sufficient to choose just one interleaving, but while for independent transitions the choice is irrelevant, here the covering transition needs to be explored first. However, the approach is limited to the situation where the covering transition does not reset any clocks.

Belluomini and Myers [BM98] also use a model with lower and upper time bounds associated to transitions. They use partially ordered sets of events, thus generating typically a single timed state for each control state. However, the full set of control states is still explored.

Bošnački and Dams [BD98] describe an extension of the Spin model checker with discrete time, which is compatible with the original partial order reduction algorithms. The result follows from the restrictions imposed on the timing extensions: clocks cannot be used in specifications, and passage of time is possible only if no other transitions are enabled.

The approach on which we draw most is that of Bengtsson, Jonsson, Lilius and Wang [BJLW98]. They define a local-time semantics based on desynchronized execution of the component automata and local time delays, with additional reference clocks to model synchronization. In this model the same independence conditions as in the untimed case apply, and an algorithm is given to decide the reachability of a local control state.

3.5 A local time model

In this section we revisit the local time model of Bengtsson et al. [BJLW98] using a somewhat different notation and prove several results which underlie its use for model checking.

To analyze the causes of dependence among transitions in a network of timed automata, consider the effects of action and delay transitions in each component automaton. From the definition of parallel composition one can see that the enabling of an action transition and the resulting state change only depend on the state of the automata in its active set. Moreover, the states in the other automata are not changed by the transition. Consequently, two action transitions involving two disjoint sets of automata are independent, just as for composition of untimed systems.

On the other hand, a delay transition changes the state in *all* automata by incrementing the values of all clocks (and is henceforth called a *global delay transition*). It becomes therefore dependent on any action transition that also changes clock values (for which $R \neq \emptyset$). However, one can view a global delay transition as a set of simultaneous transitions with equal delay in all component automata. This suggests that time-induced dependencies can be removed by separating a global delay transition into individual transitions for each component automaton, without requiring their simultaneity. To this effect, local passage of time is introduced as follows:

Let $v \in \mathcal{V}(C)$ be a clock valuation. For $d \in \mathbb{R}$ and $i \in \overline{1,n}$, define the clock valuation $v +_i d$ by: $(v +_i d)(x) = v(x) + d$ for $x \in C_i$ and $(v +_i d)(x) = v(x)$ otherwise.

A *local delay transition* $\overset{d}{\leadsto}_i$ increments only the clocks in automaton A_i. We associate such a transition with a pair $(d, i) \in \mathcal{T}_\Delta = \mathbb{R}^+ \times \overline{1,n}$, define $active(\overset{d}{\leadsto}_i) = \{i\}$ and denote by $\mathcal{T}_l = \mathcal{T} \cup \mathcal{T}_\Delta$ the set of action and local delay transitions of A. For $i \in \overline{1,n}$, define the functions $delay_i : \mathcal{T}_l \mapsto \mathbb{R}^+$ as follows: $delay_i(\overset{d}{\leadsto}_i) = d$, $delay_i(\overset{d}{\leadsto}_j) = 0$ for $i \neq j$, and $delay_i(\overset{a}{\rightarrow}) = 0$ for $a \in \mathcal{T}$. They indicate the delay caused by a transition in a component automaton.

Definition 8 *(Local time model) The local time model $\mathcal{L}(A)$ for a network of timed automata $A = A_1 \parallel A_2 \parallel \ldots \parallel A_n$ is a state-transition graph with state set Σ, initial state set Σ^0 and execution traces $\sigma = (s^0, v^0) \xrightarrow{\tau_1} (s^1, v^1) \ldots \xrightarrow{\tau_k} (s^k, v^k) \ldots$ starting from a state $(s^0, v^0) \in \Sigma^0$ and satisfying one of the following conditions for any $k \geq 1$:*

- $\tau_k = (d,i) \in \mathcal{T}_\Delta$, $s^k = s^{k-1}$, $v^k = v^{k-1} +_i d$, *and* $I_i(s_i^k)(v^{k-1} + d')$ *holds, for all* $d' \in [0,d]$, *or*

- $\tau_k \in \mathcal{T}$, $(s^{k-1}, v^{k-1}) \xrightarrow{\tau_k} (s^k, v^k)$, *and* $\sum_{l=1}^{k-1} \mathit{delay}_i(\tau_l) = \sum_{l=1}^{k-1} \mathit{delay}_j(\tau_l)$ *for all* $i, j \in \mathit{active}(\tau_k)$

In the first case, automaton A_i takes a local delay transition, denoted by $(s^{k-1}, v^{k-1}) \overset{d}{\rightsquigarrow}_i (s^k, v^k)$. The second case corresponds to an action transition $(s^{k-1}, v^{k-1}) \xrightarrow{\tau_k} (s^k, v^k)$, with the additional constraint that the elapsed time (the sum of delays) is identical for all automata in the active set. (For a local action transition, with only one active automaton, this additional constraint is void). In both cases, a transition τ_k that satisfies the given conditions is said to be *enabled* after the execution of σ_{k-1}. Denote by $\mathit{enabled}(\sigma)$ and $\mathit{enabled}^*(\sigma)$ the set of transitions and transition sequences, respectively, that can follow a finite trace σ.

For a finite execution trace $\sigma = (s^0, v^0) \xrightarrow{\tau_1} (s^1, v^1) \ldots \xrightarrow{\tau_k} (s, v)$, let $\mathit{time}_i(\sigma) = t_0 + \sum_{l=1}^{k} \mathit{delay}_i(\tau_l)$ where $t_0 \in \mathbb{R}^+$ is an arbitrary value denoting the timepoint at which the execution of σ starts. Then, $\mathit{time}_i(\sigma)$ (or simply time_i, when σ is understood from the context) denotes the timepoint reached in A_i after executing the transitions in σ. The *local configuration* of A_i reached by σ is the tuple $\mathit{cfg}_i(\sigma) = (s_i, v_i, \mathit{time}_i)$, where v_i is the restriction of v to the clocks of A_i. The *global configuration* of A is the tuple $\mathit{cfg}(\sigma) = (\mathit{cfg}_1(\sigma), \mathit{cfg}_2(\sigma), \ldots, \mathit{cfg}_n(\sigma))$, also written as $\mathit{cfg}(\sigma) = (s, v, \mathit{time})$ with $\mathit{time} = (\mathit{time}_1, \mathit{time}_2, \ldots, \mathit{time}_n)$. The set of all configurations is then $\Sigma_C = \Sigma \times (\mathbb{R}^+)^n$.

The definition of the local time model expresses the enabling of an action transition in terms of the trace executed so far. The following proposition shows that a configuration contains sufficient information to completely determine the subsequently enabled transitions.

Proposition 1 *The following properties hold in the local time model* $\mathcal{L}(A)$ *for finite execution traces* σ *and* σ' *and transition* $\tau \in \mathit{enabled}(\sigma)$*:*

- *if* $\mathit{cfg}_i(\sigma) = \mathit{cfg}_i(\sigma')$ *for all* $i \in \mathit{active}(\tau)$*, then* $\tau \in \mathit{enabled}(\sigma')$ *and* $\mathit{cfg}_i(\sigma\tau) = \mathit{cfg}_i(\sigma'\tau)$ *for all* $i \in \mathit{active}(\tau)$

- $\mathit{cfg}_j(\sigma\tau) = \mathit{cfg}_j(\sigma)$ *for all* $j \notin \mathit{active}(\tau)$*, where* $\sigma\tau$ *denotes the trace obtained by extending* σ *with the transition* τ*.*

Proof: For the first part of the proposition it suffices to show that the enabledness of a transition and its effect depend only on the local configurations of the automata in its active set. For a local delay transition $\overset{d}{\rightsquigarrow}_i$ in automaton A_i, its enabledness is a function only of the local invariant in state s_i and the clock valuation v_i. The only state change is the increment of valuation v_i by d, which is again independent of other components.

For an action transition $(s, v) \xrightarrow{a} (s', v')$, the definition of parallel composition implies that its enabledness in $\mathcal{S}(A)$ depends on the local states (s_i, v_i) and the invariants of s'_i for $i \in \mathit{active}(a)$. For $\mathcal{L}(A)$, the additional constraint is written as $\mathit{time}_i(\sigma) = \mathit{time}_j(\sigma)$ for $i, j \in \mathit{active}(a)$, which also depends only on $\mathit{cfg}_i(\sigma)$ for $i \in \mathit{active}(a)$. The state change is a function of the local state only: for $i \in \mathit{active}(a)$, s'_i is given by the edge $\langle s_i, \psi_i, R_i, s'_i\rangle$ in automaton A_i, and $v'_i = v_i[R_i \mapsto 0]$. For $j \notin \mathit{active}(a)$, we have $s'_j = s_j$ by definition and $v'_j = v_j$ since no clocks in A_j are reset.

Finally, for the time component, we have $\mathit{time}_i(\sigma\tau) = \mathit{time}_i(\sigma) + \mathit{delay}_i(\tau)$ for all $i \in \overline{1,n}$. Therefore $\mathit{time}_i(\sigma) = \mathit{time}_i(\sigma') \Rightarrow \mathit{time}_i(\sigma\tau) = \mathit{time}_i(\sigma'\tau)$. Since the definition of *delay* ensures that $\mathit{delay}_j(\tau) = 0$ for all $j \notin \mathit{active}(\tau)$, this implies $\mathit{time}_j(\sigma\tau) = \mathit{time}_j(\sigma)$ for $j \notin \mathit{active}(\tau)$. $\Box$

As a consequence, two finite execution traces leading to the same configuration have the same set of enabled transitions. For a configuration $\gamma \in \Sigma_C$ one can thus define $\mathit{enabled}(\gamma) = \mathit{enabled}(\sigma)$, where σ is an arbitrary execution trace with $\mathit{cfg}(\sigma) = \gamma$. Likewise, the successor configuration of γ by a transition $\tau \in \mathit{enabled}(\sigma)$ is defined as the configuration reached when extending the trace σ by transition τ: $\mathit{succ}_\tau(\gamma) = \mathit{cfg}(\sigma\tau)$. This is again independent of σ and we write $\gamma \xrightarrow{\tau} \mathit{succ}_\tau(\gamma)$.

We are now ready to prove the desired independence properties for transitions in $\mathcal{L}(A)$. In general, two transitions are called independent if neither disables the execution of the other, and the same state is reached by executing them in either order. This notion is formalized as follows:

Definition 9 *(Independence) Two transitions τ_1 and τ_2 are independent iff for any finite execution trace σ such that $\tau_1, \tau_2 \in \mathit{enabled}(\sigma)$ the following two conditions hold:*
Enabledness: $\tau_2 \in \mathit{enabled}(\sigma\tau_1) \wedge \tau_1 \in \mathit{enabled}(\sigma\tau_2)$
Commutativity: $\mathit{fin}(\sigma\tau_1\tau_2) = \mathit{fin}(\sigma\tau_2\tau_1) \wedge \mathit{enabled}^*(\sigma\tau_1\tau_2) = \mathit{enabled}^*(\sigma\tau_2\tau_1)$
where $\mathit{fin}(\sigma)$ denotes the last state on the trace σ.

The following theorem then holds (cf. [BJLW98]):

Theorem 1 *Two (action or local delay) transitions $\tau_1, \tau_2 \in \mathcal{T}_l$ that involve disjoint sets of automata ($active(\tau_1) \cap active(\tau_2) = \emptyset$) are independent.*

Proof: If $j \in active(\tau_2)$, then $j \notin active(\tau_1)$ and $cfg_j(\sigma\tau_1) = cfg_j(\sigma)$ for all $j \in active(\tau_2)$. Thus, $\tau_2 \in enabled(\sigma) \Rightarrow \tau_2 \in enabled(\sigma\tau_1)$, and symmetrically for the second conjunct.

For commutativity, since $active(\tau_1) \cap active(\tau_2) = \emptyset$, each of the local configurations is changed at most once, either by τ_1 or by τ_2, independently of their ordering. Therefore, $cfg(\sigma\tau_1\tau_2) = cfg(\sigma\tau_2\tau_1)$. In particular, this means $fin(\sigma\tau_1\tau_2) = fin(\sigma\tau_2\tau_1)$, and furthermore, since the enabledness of transitions depends only on the reached configuration, $enabled^*(\sigma\tau_1\tau_2) = enabled^*(\sigma\tau_2\tau_1)$. $\square$

A finite trace σ in $\mathcal{L}(A)$ is called *synchronized* if $time_i(\sigma) = time_j(\sigma)$ for all $i, j \in \overline{1, n}$, i.e., if all automata have executed for the same amount of time, denoted by $time(\sigma)$. The following theorem relates the reachable state spaces of the standard and local time models (cf. [BJLW98]):

Theorem 2 *Each state (s, v) reachable in $\mathcal{S}(A)$ is also reachable in $\mathcal{L}(A)$. Moreover, each state reached by a synchronized trace σ_l in $\mathcal{L}(A)$ is also reachable in $\mathcal{S}(A)$.*

Proof: For the first part, note that any execution trace in $\mathcal{S}(A)$ yields an execution trace in $\mathcal{L}(A)$ by replacing each global delay transition $\overset{d}{\leadsto}$ with the sequence of local delay transitions $\overset{d}{\leadsto}_1 \ldots \overset{d}{\leadsto}_n$.

The reverse implication follows by induction on the number of action transitions in σ_l. For the base case, if σ_l is synchronized and contains only local delay transitions, they sum up to the same total delay d. Then, $fin(\sigma_l)$ is reachable in $\mathcal{S}(A)$ by executing the global delay transition $\overset{d}{\leadsto}$.

For the induction step, consider the action transition a in σ_l executed at the latest timepoint, $t_a \leq t = time(\sigma_l)$. Then, in every automaton, σ_l ends with local delay transitions totaling at least $t - t_a$. Removing this delay in every automaton yields a synchronized trace σ_l' with $time(\sigma_l') = t_a$. In σ_l', a is the last transition in all participating automata. Its removal results in the synchronized execution trace σ_l'' with fewer action transitions. By the induction hypothesis, the state $fin(\sigma_l'')$ is reachable in $\mathcal{S}(A)$, and $fin(\sigma_l)$ is reachable from it by executing $\xrightarrow{a}$ followed by $\overset{t-t_a}{\leadsto}$. $\square$

3.6 The local-time zone automaton

In the global-time semantics, sets of timed states can be represented using clock constraints, resulting in a quotient structure called the zone automaton. In [BJLW98], this approach is adapted to the local-time model. Using our notations, a *local-time zone* is a convex set of configurations $z \in \Sigma_C$ with the same control state. A transition is enabled in a zone iff it is enabled in some configuration belonging to the zone. We denote this set by $\mathit{enabled}(z) = \{\tau \in \mathcal{T}_l \mid \exists \gamma \in z \,.\, \tau \in \mathit{enabled}(\gamma)\}$. The successor of a zone z by a transition $\tau \in \mathit{enabled}(z)$ is $\mathit{succ}_\tau(z) = \{\mathit{succ}_\tau(\gamma) \mid \gamma \in z \wedge \tau \in \mathit{enabled}(\gamma)\}$.

For the standard zone automaton, an exploration step consists of an action transition followed by a delay transition of arbitrary amount. For the local-time model, we combine an action transition with subsequent delay transitions in all automata belonging to its active set, and prove that any reachable local-time state can be generated in this way. Specifically, we show:

Proposition 2 *For any finite execution trace σ, there exists a trace σ' with the same final configuration, which starts with a local delay transition in each component automaton, after which every subsequent action transition is followed by local delay transitions in all participating automata.*

Proof: A delay transition $\overset{d}{\rightsquigarrow}_i$ commutes with any other delay transition, and with action transitions a such that $i \notin \mathit{active}(a)$. Thus, delay transitions can be moved towards the beginning of the execution trace σ, while merging consecutive delay transitions in the same automaton, until the preceding action transition involves the same automaton, or until it precedes all action transitions. $\Box$

Based on this result, we choose a zone successor operation that first performs an action transition $\xrightarrow{a}$, followed by arbitrary delay transitions in the automata belonging to its active set:

$$\mathit{succ}_l^{\mathrm{Z}}(z, a) = \{\gamma_k \in \Sigma_C \mid \exists \gamma \in z, \exists d_{i_1}, \cdots, d_{i_k} \in \mathbb{R}^+ \,.\, \gamma \xrightarrow{a} \gamma' \overset{d_{i_1}}{\rightsquigarrow}_{i_1} \ldots \overset{d_{i_k}}{\rightsquigarrow}_{i_k} \gamma_k\}$$

where $\mathit{active}(a) = \{i_1, i_2, \ldots, i_k\}$. The independence of local delay transitions ensures the uniqueness of the above definition irrespective of the ordering of indices in $\mathit{active}(a)$. An initial local-time zone is the set of all configurations reachable from an initial state by a sequence of delay transitions:

$$\mathit{init}_l^{\mathrm{Z}}(s^0) = \{\mathit{cfg}(\sigma) \mid \exists d_{i_1}, \cdots, d_{i_n} \in \mathbb{R}^+ \,.\, \sigma = (s^0, 0_C) \overset{d_{i_1}}{\rightsquigarrow} \ldots \overset{d_{i_n}}{\rightsquigarrow} (s^0, v^n)\}$$

If $succ_i^\Delta(z) = \{\gamma' \mid \exists \gamma \in z, \exists d \in \mathbb{R}^+ . \gamma \overset{d}{\rightsquigarrow}_i \gamma'\}$ is the successor by an arbitrary local delay, then $init_l^Z(s^0) = (succ_n^\Delta \circ \ldots \circ succ_1^\Delta)(s^0, 0_C))$ and $succ_l^Z(z, a) = (succ_{i_k}^\Delta \circ \ldots \circ succ_{i_1}^\Delta \circ succ_a)(z)$, where $\circ$ denotes function composition. The local-time zone automaton can now be defined as follows:

Definition 10 *(Local-time zone automaton) The local-time zone automaton $\mathcal{Z}_l(A)$ for a network of automata A is a tuple $(Z_l, Z_l^0, succ_l^Z)$, where $Z_l^0 = \{init_l^Z(s^0) \mid s^0 \in S^0\}$ is the set of initial local-time zones, $succ_l^Z$ is the successor relation defined above, and Z_l is the set of all local-time zones reachable by successive application of $succ_l^Z$ from an initial zone.*

Together with Proposition 2, this definition implies directly the following:

Theorem 3 *A state is reachable in the model $\mathcal{L}(A)$ iff it belongs to a zone z which is reachable in the local-time zone automaton $\mathcal{Z}_l(A)$.*

3.6.1 Representation of local-time zones

In [BJLW98], a representation of local-time zones as difference bound matrices [Dil89] is given which uses one additional variable per automaton. For a class of timed automata, we derive an improved representation which does not need additional space compared to the standard zone automaton.

In the standard zone automaton, zones are represented using difference constraint on the clocks of the automaton. Atomic constraints between two clocks are invariant to global delay transitions. However, in the local-time model, the difference between two clocks in different automata is affected by a local delay transition in either of these automata. A transition $\overset{d}{\rightsquigarrow}_i$ increments both the clocks in C_i and the value of $time_i$. Instead of reasoning about the value $v_i(x)$ of a clock $x \in C_i$, this suggests considering the value $time_i - v_i(x)$. Indeed, this value represents the timepoint at which clock x was last reset, and is consequently invariant to any delay transitions.

Consider the new variables t_i for $i \in \overline{1,n}$, and t_x for all clocks $x \in C$. Denote $T_i = \{t_x \mid x \in C_i\}$ for $i \in \overline{1,n}$, $T_i^+ = T_i \cup \{t_i\}$, $T = \{t_x \mid x \in C\} = \bigcup_{i=1}^n T_i$, and $T^+ = \bigcup_{i=1}^n T_i^+$. Given a configuration $(s, v, time)$, define the valuation $\bar{v} : T^+ \to \mathbb{R}^+$ by $\bar{v}(t_i) = time_i$ for $i \in \overline{1,n}$ (t_i is the reference time in automaton A_i) and $\bar{v}(t_x) = time_i - v(x)$ for $x \in C_i$ (t_x is the last reset time of x). Conversely, $\bar{v}$ uniquely determines v and $time$, and $(s, \bar{v})$ is an alternate representation for a configuration.

Any atomic clock constraint appearing in the description of A can be rewritten as a difference constraint on two variables in T^+. Indeed, in a difference constraint $x - y \prec c$, both clocks belong to the same automaton A_i, and therefore $x - y = (t_i - t_x) - (t_i - t_y) = t_y - t_x$. Likewise, the constraints $x \prec c$ or $c \prec x$ can be rewritten as $t_i - t_x \prec c$ or $c \prec t_i - t_x$, respectively.

A *local-time clock zone* is the set of valuations belonging to a local-time zone. A zone is then written as $\langle s, \psi_l \rangle$ where s is the control state and ψ_l is the clock zone. We prove:

Proposition 3 *A local-time clock zone can be written as a difference constraint on the variables in T^+: $\psi_l = \bigwedge_{t_u,t_w \in T^+} t_u - t_w \prec c_{uw}$, with $c_{uw} \in \mathbb{Z}$.*

Proof: In an initial configuration, $t_x = t_i = t_0$, $\forall x \in C_i$, $i \in \overline{1,n}$. Thus, $\psi_l = \bigwedge_{t_u,t_w \in T^+}(t_u = t_w)$.

For an action transition $(s, \bar{v}) \xrightarrow{a} (s', \bar{v}')$, we have $\bar{v}'(t_u) = \bar{v}(t_u)$ for $u \notin R_a$ and $\bar{v}'(t_x) = t_i$ for $x \in R_a$ (where i_x identifies the automaton A_{i_x} containing x). We denote this by $\bar{v}' = \bar{v}[t_x \mapsto t_{i_x}]_{x \in R_a}$ and extend the notation to clock zones. Also, the enabling condition ψ_a holds for $\bar{v}$ and the reference times in $T_a = \{t_i \mid i \in \mathit{active}(a)\}$ have equal values. Thus, we have $\mathit{succ}_a(\psi_l) = \{\bar{v}' \mid (s, \bar{v}) \xrightarrow{a} (s', \bar{v}')\} = (\psi_l \wedge \psi_a \wedge \bigwedge_{t_i,t_j \in T_a} t_i = t_j)[t_x \mapsto t_{i_x}]_{x \in R_a}$, or equivalently $\mathit{succ}_a(\psi_l) = [\exists_{X_a} . \psi_l \wedge \psi_a \wedge \bigwedge_{t_i,t_j \in T_a} t_i = t_j] \wedge \bigwedge_{x \in R_a} t_x = t_{i_x}$, with $X_a = \{t_x \mid x \in R_a\}$ and $\exists_{X_a}$ denoting quantification over all variables in X_a. Difference constraints are closed under conjunction and quantification, therefore $\mathit{suc}_a(\psi_l)$ is a difference constraint.

For a local delay transition $(s, \bar{v}) \overset{d}{\rightsquigarrow}_i (s, \bar{v}')$, we have $\bar{v}'(t_i) = \bar{v}(t_i) + d$ and $\bar{v}'(t_u) = \bar{v}(t_u)$ for all $t_u \in T^+ \setminus \{t_i\}$. Denote this by $\bar{v}' = \bar{v} +_i d$ and let $\psi_l \Uparrow^i = \{\bar{v}' \mid \exists \bar{v} \in \psi_l, \exists d \in \mathbb{R}^+ . \bar{v}' = \bar{v} +_i d\}$ be the zone obtained from ψ_l after an arbitrary delay $\overset{d}{\rightsquigarrow}_i$. If $e[y/x]$ denotes substitution of y for x in e, then we have $\psi_l \Uparrow^i = [\exists d \in \mathbb{R}^+ . \psi_l][t_i / t_i + d] = [\exists t_i \in \mathbb{R}^+ . \psi_l \wedge t'_i \geq t_i][t_i / t'_i]$. Since $(s, \bar{v}) \overset{d}{\rightsquigarrow}_i (s, \bar{v}')$ iff $\bar{v}' = \bar{v} +_i d$ and $I_i(s_i)(\bar{v}')$ holds, we obtain that $\mathit{succ}_i^{\Delta}(\psi_l) = \psi_l \Uparrow^i \wedge I_i(s_i)$, which is again a difference constraint.

Combining action and delay steps, we obtain:
$\mathit{succ}_l^{\mathrm{Z}}(\psi_l, a) = ([\exists_{X_a} . \psi_l \wedge \psi_a \wedge \bigwedge_{t_i,t_j \in T_a} t_i = t_j] \wedge \bigwedge_{x \in R_a} t_x = t_{i_x}) \Uparrow^{i_1} \ldots \Uparrow^{i_k} \wedge \bigwedge_{i \in \mathit{active}(a)} I_i(s'_i)$. $\square$

Despite the desynchronization introduced by the local-time model, the representation of a local-time clock zone is still monolithic and relates reset times of clocks to reference times in *all* automata. We prove that for a class of networks the following simpler representation holds:

Proposition 4 *If every synchronization transition in network A resets at least one clock in each participating automaton, a local-time clock zone has the form $\psi_l = \psi_\Delta(T) \wedge \bigwedge_{i=1}^{n} \psi_i(T_i, t_i)$, where:*
- $\psi_\Delta(T) = \bigwedge_{t_x \neq t_y \in T} t_x - t_y \prec c_{xy}$, *with* $c_{xy} \in \mathbb{Z}$
- $\psi_i(T_i, t_i) = \bigwedge_{t_x \in T_i} (t_i - t_x \prec c_{ix} \wedge t_x - t_i \prec c_{xi})$ *with* $c_{ix}, c_{xi} \in \mathbb{Z}$

In this case, we call A a *sync-reset* network of automata. The special form for a clock constraint in this case signifies that there is no need to explicitly maintain constraints that relate the reset time of a clock to the local time of a different automaton. The constraint is composed of a global constraint $\psi_\Delta(T)$ that relates pairs of any two reset times, and of one local constraint ψ_i for each process, comparing the reset times in the automaton A_i to its local clock t_i. A network of automata A may satisfy this additional property if each synchronization transition determines the future timing behavior of both automata involved, and it is thus necessary to refer to its execution time by means of a clock reset in both automata.
Proof: The initial zone can be written as: $init_l^{\mathbb{Z}}(s^0) = \bigwedge_{x,y \in C}(t_x = t_y) \wedge \bigwedge_{i=1}^{n} I_i(s_i^0)$. In the expression of $succ_l^{\mathbb{Z}}$ from Proposition 3 the term $\psi_l \wedge \psi_a \wedge \bigwedge_{t_i,t_j \in T_a} t_i = t_j$ has the required form, save for the equalities $t_i = t_j$. Quantification over X_a introduces constraints between t_x and t_i, for $t_x \in T$ and $i \in active(a)$. By assumption, for every $i \in active(a)$ there exists a clock $x \in R_a \cap C_i$ that is reset, and the new value of t_x is t_i. Therefore, constraints on t_i and t_y can be replaced with constraints between t_x and t_y, which are incorporated in ψ_Δ. Finally, executing $\Uparrow^i$ for $i \in active(a)$ removes the equalities $t_i = t_j$, and adds inequalities of the form $t_u - t_j = (t_x - t_i) + (t_i - t_j) \prec_{ui} c_{ui} + 0$ with $u \notin C_j$. However, if $y \in R_a \cap C_j$, this inequality can already be obtained considering $(t_u - t_y) + (t_y - t_j)$, both terms already present in the desired form. $\Box$

We give an example to show that the reduced representation is not sufficient in the general case. Consider automata A_1 and A_2, with clocks x and y, that synchronize on transition a. After executing the synchronization transition, the full representation of the corresponding clock zone would be $t_1 - t_x > 3 \wedge t_2 - t_x > 3 \wedge t_1 - t_y \geq 0 \wedge t_2 - t_y \geq 0$. Then, transition b can only be executed if $t_2 - t_y < 2$, which given that $t_2 - t_x > 3$, implies $t_x - t_y < 1$. However, the constraints $t_2 - t_x > 3$ and $t_1 - t_y \geq 0$ cannot be part of the simplified representation. If these constraints are ignored, the system could execute transition b regardless of the relation between t_x and t_y, leading to extraneous behaviors.

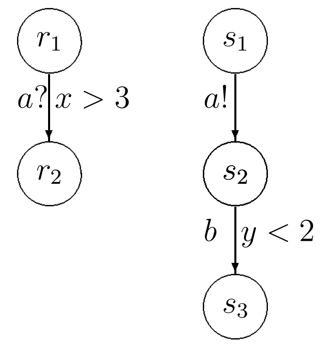

Figure 3.3: Synchronization transitions and zone representation

Clock difference constraints are generally represented as *difference-bound matrices* [Dil89], which are indexed by clock variables whose elements are *bounds*, i.e., pairs of the form $(\prec, c)$ corresponding to an atomic clock constraint. The component ψ_Δ of a local time zone can be represented as a difference bound matrix with $|C|$ rows and columns. Each constraint ψ_i requires $2 * |C_i|$ additional time bounds, for a total of $2 * |C|$, i.e., an additional row and column. Thus, ψ_l can be represented by a matrix with $|C| + 1$ rows and columns, the same size as the DBM used in the standard algorithm.

However, the computations performed on this matrix must take into account that segments of the additional row and column correspond to different automata and thus different reference times. The successor computation for a transition is performed first on the submatrix corresponding to the clocks of the active automata together with their reference times (which have to be equal in this case). If any constraints between clocks are strengthened in this process, the $|C| \times |C|$ submatrix corresponding to ψ_Δ is canonicalized. This may strengthen constraints between clocks in an automaton A_k outside the active set of the transition, which may in turn strengthen constraints in ψ_k between the clocks in A_k and the reference time t_k.

If some automata in the network have synchronization transitions that do not reset clocks, one solution is to introduce in each of these automata an additional clock that is reset on such synchronization transitions. In this way, the network of automata is transformed into a sync-reset network, with potentially fewer than n additional time variables.

In the general case, a smaller difference bound matrix can also be obtained using the clock activity reduction of [DY96]. In this case the dimension of the DBM changes dynamically at each state, by eliminating the clocks that will be no longer used before their next reset. Using the same approach in the local-time model, we can also eliminate reference times in some cases. If all transitions entering local state s in automaton A_i reset clock x, then the strongest constraints at s on the reference time t_i are $t_i \geq t_x$, together with any local invariant of s. Thus, it is possible to represent the local-time zone at s as a DBM without t_i, and add the above-mentioned constraints when the next local transition from s is explored.

3.7 Preservation of LTL$_\Delta$ formulas

In the local-time model $\mathcal{L}(A)$ the executions of the component automata are decoupled from each other, except for synchronization transitions. Consequently, $\mathcal{L}(A)$ accepts a richer set of behaviors than $\mathcal{S}(A)$. This section establishes restrictions on the local-time model which ensure that each of its traces is equivalent with respect to a given LTL$_\Delta$ formula φ to a trace of the standard model.

The semantics of LTL$_\Delta$ is extended to the local time model by defining the satisfaction of an atomic time constraint in a local-time configuration:

$$(s, \bar{v}) \models x - y \prec c \text{ iff } \bar{v}(t_y) - \bar{v}(t_x) \prec c$$

We have $\bar{v}(t_y) - \bar{v}(t_x) = (time_j - v(y)) - (time_i - v(x))$, assuming $x \in C_i$ and $y \in C_j$. Thus, in a synchronized configuration ($time_i = time_j$) the semantics is the same as for the standard model.

Since next-time free LTL formulas are invariant under stuttering, the transitions which affect the truth of the specification are identified as follows:

Definition 11 *(Visibility) A transition $(s, v) \to (s', v')$ is* invisible *with respect to a specification φ if every atomic subformula of φ has the same truth value in (s, v) and (s', v'). A transition which is not* invisible *is called* visible.

Then, a transition in $\mathcal{L}(A)$ is visible if it connects two states which differ by at least one atomic proposition in the specification (visibility in the control space) or it resets at least one clock in the specification, affecting the truth value of a difference constraint (visibility in the time domain). Delay transitions are invisible, since they do not change the control state and do not reset clocks.

For a network of timed automata A and a formula φ in LTL$_\Delta$ denote by $\mathcal{F}_\varphi(A)$ the set of those traces of $\mathcal{L}(A)$ which satisfy the following properties:

- *Ordering* (**O**): Visible transitions occur in increasing order of their execution times. That is, in any trace $\sigma \in \mathcal{F}_\varphi(A)$, for visible transitions τ_k and τ_l with $k < l$, we have $time(\tau_k) \leq time(\tau_l)$ (where $time(\tau) = time_i$ for some $i \in active(\tau)$ is the timepoint at which τ is executed).
- *Fairness* (**F**): Time progress is unbounded in all component automata. That is, for any trace $\sigma \in \mathcal{F}_\varphi(A)$, automaton A_i and time $M \in \mathbb{R}^+$, there exists $k \in \mathbb{N}$ with $time_i(\sigma_k) > M$.

Theorem 4 *Given an LTL$_\Delta$ formula φ, for any execution trace in the model $\mathcal{S}(A)$ there exists an execution trace in $\mathcal{F}_\varphi(A)$ which has the same truth value for φ and vice versa.*

Proof: The direct implication is straightforward: from a trace σ in $\mathcal{S}(A)$ construct a trace σ_l in $\mathcal{L}(A)$ by replacing each global delay transition $\overset{d}{\rightsquigarrow}$ with the sequence of local delay transitions $\overset{d}{\rightsquigarrow}_1 \ldots \overset{d}{\rightsquigarrow}_n$. The trace σ_l also satisfies **O**, since no action transitions are reordered, and **F**, since the same delay transitions are executed in each automaton. Since delay transitions are invisible, this transformation does not change the truth value of the LTL$_\Delta$ formula φ, and $\sigma \models \varphi$ iff $\sigma_l \models \varphi$.

For the reverse implication, we construct σ from σ_l by reordering all transitions so they occur in increasing order of their timepoints. The ordering condition **O** guarantees that no visible transitions are reordered, and the truth value of the formula is not changed. In this transformation, delay transitions may be split and reordered so every action transition is preceded by equal delays in all automata. The fairness condition **F** guarantees that for all automata, local delay transitions totaling the needed amount exist in σ_l. Finally, all local delay transitions between two consecutive action transitions are merged into a global delay transition, resulting in a trace σ of $\mathcal{S}(A)$. $\square$

Based on the above theorem, we proceed as follows: We first define a restricted local-time model $\mathcal{L}^\varphi(A)$ whose traces satisfy the ordering condition **O**. Next, we construct a zone automaton $\mathcal{Z}_l^\varphi(A)$ whose states are local-time *atoms*, i.e., sets of configurations with the same truth value for all atomic subformulas of φ. We show a correspondence between the traces of $\mathcal{L}^\varphi(A)$ and $\mathcal{Z}_l^\varphi(A)$, and then impose a fairness condition corresponding to **F** to ensure equivalence with the standard model. Finally, we apply a *maximization* of the atoms in $\mathcal{Z}_l^\varphi(A)$ to obtain an automaton $\mathcal{M}_l^\varphi(A)$ which is guaranteed to be finite and therefore amenable to model checking.

To preserve the ordering of visible transitions, we introduce an additional reference variable t_v, which denotes the timepoint of the last executed visible transition. The domain of the valuation $\bar{v}$ is extended to include t_v. In the initial configuration, $\bar{v}(t_v) = 0$. The model $\mathcal{L}^\varphi(A)$ is defined in the same way as $\mathcal{L}(A)$, but with the additional restriction $\bar{v}(t_v) \leq \mathit{time}(a)$ for the execution of a visible transition $\xrightarrow{a}$, and $\bar{v}'(t_v) = \mathit{time}(a)$ in the resulting configuration. This guarantees that each visible transition is executed at a later timepoint than the previous one, and thus ensures condition **O**.

With these additional constraints, the zone successor for a visible transition becomes: $\mathit{succ}_a^v(\psi_l) = [\exists_{X_a} \exists t_v \,.\, \psi \wedge \psi_a \wedge \bigwedge_{t_i,t_j \in T_a} t_i = t_j \wedge \bigwedge_{t_i \in T_a} t_v \leq t_i]$ $\wedge \bigwedge_{t_i \in T_A} t_v = t_i \wedge \bigwedge_{x \in C_i \cap R_a} t_x = t_i$. For invisible transitions, the successor operation remains the same.

To perform model checking on the local-time zone automaton, one has to consider zones in which all configurations satisfy the same atomic subformulas of the specification φ (cf. [YS97]):

Definition 12 *(Atom) Given a timed automaton A and an LTL_Δ formula φ, an* atom *is a zone $\langle s, \psi_l \rangle$ such that $\forall \bar{v}_1, \bar{v}_2 \in \psi_l \,.\, \bar{v}_1(t_y) - \bar{v}_1(t_x) \prec c \Leftrightarrow \bar{v}_2(t_y) - \bar{v}_2(t_x) \prec c$ for any constraint $x - y \prec c$ in φ.*

Consequently, two configurations $(s, \bar{v}_1)$ and $(s, \bar{v}_2)$ in an atom $\langle s, \psi_l \rangle$ have the same truth value for all atomic constraints in formula φ. We introduce the additional atomic propositions $q_k \in Q$, $q_k = t_{y_k} - t_{x_k} \prec_k c_k$ for each atomic clock constraint in φ and thus reduce φ to a next-time free LTL formula φ_q. The atoms comprising $\langle s, \psi_l \rangle$ are given by the nonempty intersections between ψ_l and all constraints $t_{y_k} - t_{x_k} \prec_k c_k$, either in positive or negated form:

$at^\varphi(\langle s, \psi_l \rangle) = \{\langle s, \phi \rangle \mid \phi = \psi_l \wedge \bigwedge_{k=1}^m q_k',\ \phi \neq \emptyset, \text{ with } q_k' = q_k \text{ or } q_k' = \neg q_k\}$.

Define transitions between atoms as follows: $z \overset{a}{\Rightarrow} z'$ if $a \in \mathit{enabled}(z)$ and $z' \in at^\varphi(\mathit{succ}_l^Z(z, a))$, and $z \overset{\epsilon}{\Rightarrow} z$ if all local states s_i of z have a trivial invariant $I_i(s_i) = \mathit{true}$, for $1 \leq i \leq n$. Only in this case, the automaton can remain at that state forever. If at least one local control state has an invariant with an upper bound, the system will be forced to a different global state as the local state in that automaton changes. Then, the *atom graph* corresponding to A and formula φ is defined as follows:

Definition 13 *(Atom graph) The atom graph $\mathcal{A}^{\varphi}(A)$ of a timed automaton A with respect to formula φ is a state-transition graph $(Z_l^{\varphi}, Z_l^0, \Rightarrow)$, with Z_l^0 the set of initial local-time zones, $\Rightarrow$ the atom transition relation and Z_l^{φ} the set of atoms reachable from Z_l^0 by repeated application of $\Rightarrow$.*

Then, our problem reduces to LTL model checking:

Proposition 5 *For each execution trace σ_l of $\mathcal{L}^{\varphi}(A)$, there is an atom sequence in $\mathcal{A}^{\varphi}(A)$ that has the same truth value for φ_q as σ_l has for φ and vice versa.*

Proof: The proof is based on reordering the delay transitions in trace σ_l as done in the proof of Proposition 2. Any delay transition $\overset{d}{\rightsquigarrow}_i$ for which σ_l contains a subsequent action transition in the same automaton A_i can be moved towards the beginning of σ_l (possibly merging consecutive delay transitions in the same automaton), until either the preceding action transition involves A_i, or there is no preceding action transition. Let σ_l' be the execution trace obtained by this transformation. We have $\sigma_l' \models \varphi$ iff $\sigma_l \models \varphi$, since delay transitions are invisible and their permutation does not change the truth value of the formula.

We now establish by induction a correspondence between the execution trace σ_l' and an atom sequence $\rho = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \ldots$. If s^0 is the initial control state in σ_l', then $\alpha_0 = init_l^{\mathrm{Z}}(s^0)$ is the first atom of ρ. Indeed, if γ_0 is the configuration reached in σ_l' from $(s^0, 0_C)$ by executing any delay transitions before the first action transition, then $\gamma_0 \in \alpha^0$.

For the induction step, consider the subsequence of σ_l' starting at configuration $\gamma_k \in \alpha_k$ with $k \geq 0$, ending at a configuration γ_{k+1}, and consisting of an action transition $\xrightarrow{a_k}$ followed by any delay transitions up to the next action transition in σ_l'. These delay transitions must occur either in automata from the active set of $\xrightarrow{a_k}$, or in automata which have no subsequent action transition in σ_l. For the latter transitions, the invariant at the local state must be trivially *true*, since time advances to infinity in σ_l'. Thus, taking these transitions leads to configurations in the same atom. The delay transitions in automata from the active set of $\xrightarrow{a_k}$ are included in the definition of the zone successor for a_k, and consequently, we have we have $\gamma_{k+1} \in succ_l^{\mathrm{Z}}(\alpha_k, a_k)$. Thus, we can define α_{k+1} as the atom from $at^{\varphi}(succ_l^{\mathrm{Z}}(\alpha_k, a_k))$ to which configuration γ_{k+1} belongs, preserving the induction invariant.

Finally, if σ_l' contains only a finite number of action transitions, it means that the resulting state has trivial invariants at each local state. Then we can extend the atom sequence (α_k) with an $\overset{\epsilon}{\Rightarrow}$ transition for each delay transition in σ'_l following the last action transition. Since by construction $\gamma_k \in \alpha_k$, it follows that σ'_l and ρ have pointwise the same truth values for all atomic propositions in $P \cup Q$ (the delay transitions in σ'_l and the $\overset{\epsilon}{\Rightarrow}$ transitions in ρ are stuttering steps).

For the reverse step, since $\alpha \overset{a}{\Rightarrow} \alpha'$ iff *every* configuration in α' is reachable from some configuration in α by executing $\xrightarrow{a}$ followed by delay transitions, it follows by induction that any atom sequence ρ has a witness trace of configurations. Since the constructed configurations belong pairwise to atoms in ρ, the two sequences must have the same truth value for the formula φ. $\square$

It remains to restrict the zone execution sequences such that the included execution traces satisfy the fairness condition **F**. Otherwise, the local-time model may contain traces that do not require all automata to execute, and do not correspond to any trace in the standard model. The fairness condition **F** is violated if in one of the component automata the execution trace cannot make indefinite time progress. This is the case if, starting from some point in the zone sequence, there exists a clock on which each zone imposes an upper bound due to its invariant. The negation of this condition means that any clock which is infinitely often limited by an invariant has to be reset infinitely often, allowing time to diverge. Consequently, the fairness constraint can be written as a temporal logic formula in terms of the underlying state-transition structure of the automaton, $\bigwedge_{x \in C} \mathbf{GF} x.\mathit{bounded} \Rightarrow \mathbf{GF} x.\mathit{reset}$. The model checking problem on the initial network of automata is thus reduced to LTL model checking of a finite Kripke structure with a set of fairness constraints.

The fairness constraint can also be enforced by a more restrictive definition of allowable successor transitions, while also providing a guarantee that the local-time atom graph will not contain more zones than the one constructed for a global-time model. Note that allowing each automaton to execute decoupled, in its own local time scale can lead to some automata overtaking the others and some lagging behind in time. In particular, this may lead to the exploration of control states that do not appear in the original model, because the local reference times do not coincide. This does not affect the correctness of our result, since we have restricted visible transitions to their initial ordering. However, it may cause the local-time model (to which partial order reduction will be applied) to contain more enabled transitions at each state (since they do not have to be executed in time order), and thus more control states.

A local-time zone $\langle s, \psi_l \rangle$ is called *synchronizable* if it contains at least one *synchronized* configuration, with $\bar{v}(t_i) = \bar{v}(t_j)$ for all $i, j \in \overline{1,n}$. In other words, $\langle s, \psi_l \rangle$ is synchronizable iff $\psi_l \wedge \bigwedge_{i \neq j} t_i = t_j$ is satisfiable. A transition is *firable* in zone $\langle s, \psi_l \rangle$ if it is enabled in $\langle s, \psi_l \rangle$ and $succ_l^Z(\langle s, \psi_l \rangle, a)$ is synchronizable. If the atom graph is generated using only firable transitions, this ensures that a transition can be taken in the atom graph iff it can be taken in the original zone automaton. Clearly, this also ensures the fairness conditions, since the time progress of at least one automaton (due to the non-Zeno assumption) together with synchronization implies the time progress of all components towards infinity. In terms of efficiency, this approach trades a potentially smaller size of the model before reduction against a more complex test for firability of a transition.

3.8 Building a finite model

In general, the local-time zone automaton can be infinite, since the difference bounds on clocks can become arbitrarily large. The original formulation of the local-time model [BJLW98] gives a proof that the infinite number of local-time zones can be divided into a finite number of equivalence classes, based on the standard region-graph equivalence. However, this proof is nonconstructive. In particular, it gives no concrete means of determining the equivalence of two unsynchronized local-time zones, which is needed to ensure termination of the state space search.

In this section, we show that, just as in the case of the standard zone automaton, the actual value of the bounds on clock differences does not affect the enabledness of transitions, once a certain value is exceeded. Each local-time zone can therefore be normalized in order to obtain a finite model.

We adapt the *maximization* operation used, e.g., in [Won94] to the local-time model. Let c_{min} and c_{max} be the minimum and maximum constants in the description of the automaton A and the formula φ (assuming all constraints are given in canonical form, $t_u - t_v \prec d$). We adapt the region graph construction of [ACD90] to the local-time model as follows:

Definition 14 *Two valuations $\bar{v}$ and $\bar{v}'$ are called* region-equivalent *(denoted by $\bar{v} \simeq_{\text{reg}} \bar{v}'$) if for any time variables $t_u, t_v \in T^+$, one of the following conditions holds:*

(a) $c_{min} \leq \lfloor \bar{v}(t_u) - \bar{v}(t_v) \rfloor = \lfloor \bar{v}'(t_u) - \bar{v}'(t_v) \rfloor \leq c_{max}$, *and* $\bar{v}(t_u) - \bar{v}(t_v) \in \mathbb{Z} \Leftrightarrow \bar{v}'(t_u) - \bar{v}'(t_v) \in \mathbb{Z}$

(b) $\lfloor \bar{v}(t_u) - \bar{v}(t_v) \rfloor < c_{min}$ *and* $\lfloor \bar{v}'(t_u) - \bar{v}'(t_v) \rfloor < c_{min}$

(c) $\lfloor \bar{v}(t_u) - \bar{v}(t_v) \rfloor > c_{max}$ *and* $\lfloor \bar{v}'(t_u) - \bar{v}'(t_v) \rfloor > c_{max}$

Region equivalence can be extended naturally to configurations by defining $(s, \bar{v}) \simeq_{\mathrm{reg}} (s', \bar{v}')$ iff $s = s'$ and $\bar{v} \simeq_{\mathrm{reg}} \bar{v}'$. *Regions* are the equivalence classes induced by $\simeq_{\mathrm{reg}}$ on the set of configurations Σ_C. The following lemma holds:

Lemma 6 *Let $\bar{v} \simeq_{\mathrm{reg}} \bar{v}'$. Then:*

1. *For any constraint ψ in A or in the specification φ, $\bar{v} \in \psi$ iff $\bar{v}' \in \psi$.*
2. *For any clock set R, $\bar{v}[R \mapsto 0] \simeq_{\mathrm{reg}} \bar{v}'[R \mapsto 0]$.*
3. *For $i \in \overline{1, n}$ and $d \geq 0$ there exists $d' \geq 0$ such that $\bar{v} +_i d \simeq_{\mathrm{reg}} \bar{v}' +_i d'$.*

The proof reduces to the known result for the (global) region graph construction, with the following two observations. First, the local-time model adds the implicit constraints $t_i = t_j$ for synchronization transitions, but the constants in this constraints are 0, and do not influence c_{min} and c_{max}. Second, when performing a local-time delay in automaton A_i, the only variable that changes its valuation is t_i. Therefore, the other reference times t_j, with $j \neq i$ are indistinguishable from ordinary reset times t_x, with $x \in C$, and the situation is identical to the global time model, for which the property is known to hold.

Since the execution of any transition is expressed in terms of conjuncting with the constraints of A, resetting clocks and advancing local time, Lemma 6 implies the following property (cf. [ACD90]):

Proposition 7 *Let $\gamma \simeq_{\mathrm{reg}} \gamma'$ be two region-equivalent configurations in Σ_C.*

1. *If $\gamma \xrightarrow{a} \gamma_1$, there exists $\gamma_1' \simeq_{\mathrm{reg}} \gamma_1$ such that $\gamma' \xrightarrow{a} \gamma_1'$.*
2. *If $\gamma \overset{d}{\leadsto}_i \gamma_1$ with $d \in \mathbb{R}^+$, $i \in \overline{1, n}$, there exists $d' \in \mathbb{R}^+$ and $\gamma_1' \simeq_{\mathrm{reg}} \gamma_1$ such that $\gamma' \overset{d'}{\leadsto}_i \gamma_1'$.*

We define the maximization $\max(z)$ of a zone z as the set of configurations which are equivalent to some region-equivalent configuration in z: $\max(z) = \{\gamma' \in \Sigma_C \mid \exists \gamma \in z \,.\, \gamma \simeq_{\text{reg}} \gamma'\}$. A maximized zone is therefore a convex union of regions, since by including one configuration of a region it has to include all others. It is easily seen that a maximized zone is obtained from the canonical representation of a zone by modifying all constraints outside the range $[c_{min}, c_{max}]$: $t_u - t_v \prec c$ with $c < c_{min}$ becomes $t_u - t_v < c_{min}$ and $t_u - t_v \prec c$ with $c > c_{max}$ becomes $t_u - t_v < \infty$ (trivially true). Furthermore, by point (1) of Lemma 6, a maximized atom is in turn an atom. Define $succ_l^{\mathrm{M}}(z, a) = \max(succ_l^{\mathrm{Z}}(z, a))$ and let $\mathcal{M}_l^{\varphi}(A)$ be the atom graph induced by $succ_l^{\mathrm{M}}$ through repeated application from an initial zone. Since the constants in a maximized zone are bounded, it follows that $\mathcal{M}_l^{\varphi}(A)$ is finite.

By Proposition 7, the same transitions are enabled in every point of a region. Since a maximized atom is the closure of an atom with respect to region equivalence, this implies that the atom graph $\mathcal{A}^{\varphi}(A)$ and the maximized atom graph graph $\mathcal{M}^{\varphi}(A)$ are bisimilar. Putting the previous results together, we obtain the following theorem, which reduces our initial problem to LTL model checking with fairness constraints on a finite model:

Theorem 5 *The model $\mathcal{M}_l^{\varphi}(A)$ with the fairness constraint* $\mathbf{F}$ *is equivalent to the standard model $\mathcal{S}(A)$ with respect to the formula φ.*

3.9 Partial order reduction

Having established the visible transitions in the model $\mathcal{M}_l^{\varphi}(A)$, one needs to determine the transition dependence relation in order to apply partial order reduction. Bengtsson et al. [BJLW98] give a purely structural dependence relation, identical to that for untimed parallel composition: two transitions are independent if the two sets of automata involved in each of them are disjoint. Indeed, Theorem 1 shows that this condition is sufficient for the local-time model $\mathcal{L}(A)$. Since transitions in the zone automaton are composed of action and local delay transitions in the local-time model, the independence condition also follows for the zone automaton: a and b are independent if $active(a) \cap active(b) = \emptyset$, and then we have $succ_l^{\mathrm{Z}}(succ_l^{\mathrm{Z}}(z, a), b) = succ_l^{\mathrm{Z}}(succ_l^{\mathrm{Z}}(z, b), a)$.

However, in the local-time zone automaton, just like in the standard zone automaton, one needs to take into account the fact that transitions which are both enabled in a zone may actually be enabled in different sets of configurations belonging to that zone.

To see this, consider automata A_1 and A_2 with clock sets $\{x, u\}$ and $\{y, v\}$ respectively and assume that the current zone has been reached after executing two synchronization transitions, one resetting x and y, and the second resetting u and v. Thus, we have $t_x = t_y$ and $t_u = t_v$. Assume now that transition a in A_1 has enabling condition $x - u = t_u - t_x < 2$ and transition b in A_2 requires $y - v = t_v - t_y > 3$. Since $t_u - t_x = t_v - t_y$ due to the previous synchronizations, the two conditions cannot be satisfied simultaneously. Exploring either of $\stackrel{a}{\Rightarrow}$ and $\stackrel{b}{\Rightarrow}$ restricts the current local-time zone to a fragment where the other transition is no longer enabled. Thus, even though $\stackrel{a}{\Rightarrow}$ and $\stackrel{b}{\Rightarrow}$ are independent, selecting only one of them as an ample set would violate condition **C0**.

Consequently, when selecting a set of ample transitions, one needs to make sure that condition **C0** is observed and at least one ample transition is enabled in every configuration that has a transition enabled in the unreduced model. Let $guard(a)$ be the enabling condition of $\xrightarrow{a}$ in the local-time model, i.e., $\psi_a \wedge \bigwedge_{i,j \in active(a)} t_i = t_j$. If ψ_l is the current local-time zone at state s, we require $\psi_l \wedge \bigvee_{a \in enabled(s)} guard(a) = \psi_l \wedge \bigvee_{a \in ample(s)} guard(a)$.

A simpler, sufficient condition can be given as follows. Let T_{ample} be the set of all time variables (clock reset times and reference times) in the automata that contain transitions from the current ample set. The remaining enabled transitions do not involve any of these automata and thus depend only on variables in $T^+ \setminus T_{ample}$. If the set of configurations from which an ample transition is enabled, $\psi_l \wedge \bigvee_{a \in ample(s)} guard(a)$, contains any possible combination of variables in $T^+ \setminus T_{ample}$ allowed by ψ_l, then there are no configurations in ψ_l for which transitions outside the ample set are enabled, while transitions in the ample set are not. Thus, condition **C0** is preserved. The corresponding relation is: $\exists_{T_{ample}} \psi_l \wedge \bigvee_{a \in ample(s)} guard(a) = \exists_{T_{ample}} \psi_l$. In particular, this relation is easy to check if the ample set contains a simple transition: it means that after conjuncting with its guard, the projection of the local-time zone onto the remaining automata is unmodified.

The ample set reduction is done according to the criteria outlined in Section 2.6: a set of automata (ideally, a single one) with no locally enabled communication to automata outside the set is found. The cycle closing condition can be ensured both using the traditional depth-first search or using static partial order reduction, based on analyzing the cycle structure of the individual automata. Finally, if at the current point all local control states have trivial invariants, one takes into account that an infinite sequence of self-loop transitions $\stackrel{\epsilon}{\Rightarrow}$ is possible from this state.

If a local state with a nontrivial invariant is explored, one must make sure that when the upper bound of the invariant is reached, at least one of the transitions is enabled, otherwise, deadlock occurs since time cannot progress. If this invariant is of the form $(x_{i_1} \leq d_{i_1}) \wedge \ldots \wedge (x_{i_k} \leq d_{i_k})$, the outgoing transitions are $a_1, \cdots a_l$ and the current clock zone is ψ_l, it has to be true that $\psi_l \wedge ((x_{i_1} = d_{i_1}) \vee \ldots \vee (x_{i_k} = d_{i_k})) \Rightarrow (\psi_{a_1} \vee \ldots \vee \psi_{a_l})$. A similar test can be made in the limit if the invariant inequalities are strict. This is generally considered a correct design issue and is checked statically, with $\psi_l = true$, however, this requirement may be relaxed in favor of dynamic checking.

Since by introducing the auxiliary atomic propositions q_i, the LTL$_\Delta$ formula has been reduced to LTL, the ample set method can be used to construct a reduced model for the automaton $\mathcal{M}_l^\varphi(A)$, and perform model checking by composing it with the tableau for the LTL formula either using a complete construction [VW86] or on the fly [GPVW95].

Although our discussion has been limited to LTL$_\Delta$, a similar approach can be taken for a branching time logic, such as CTL without the nexttime operator. In this case, one can use the result of [GKPP99], which gives an additional condition for partial order reduction: each state which is not fully expanded must have an ample set with a single transition.

3.10 Summary

We have presented a method that allows the application of partial order reduction to systems modeled as a composition of timed automata. The method results in reduction in the state space, as well as in the number of clock zones that are generated for each control state. Compared to previous related work, we have shown that partial order reduction can be used for model checking of properties described in a timed extension of linear temporal logic, rather than just for local reachability analysis. We have also proved that the state space of the local-time zone automaton admits a finite quotient by identifying when two zones are equivalent, and thus made a state space search algorithm possible. For a certain class of automata, we show that the local time zones can be represented as efficiently as standard clock zones. Finally, we give practical conditions for selecting ample sets.

Chapter 4

Reduction for Other Timed Models

4.1 Partial Order Reduction for the Region Graph Automaton

We have so far investigated partial order reduction for timed automata by using the zone automaton construction. There are other ways in which a finite quotient for the timed state space of a timed automaton can be built. The first such construction described in the literature is the *region graph automaton* [AD90, ACD90]. Although the region graph is in general more finely-grained than the zone automaton, it abstracts away from the passage of time and can be encoded as a simple finite-state machine.

Recall that a timed automaton is a tuple $A = (S, S^0, C, E, I, \mu)$, where S is the set of states, S^0 the set of initial states, C the set of clocks, E the set of edges, I the invariant function for each node and μ a function labeling states with atomic propositions. The standard model of a timed automaton has timed states of the form (s, v), where $s \in S$ is a control state and $v : C \to \mathbb{R}$ is a clock valuation.

The region graph is the quotient structure induced by an equivalence relation on the timed states of a timed automaton: two states with the same control location are equivalent if all clock values agree on their integral parts and have the same ordering of their fractional parts. Clocks that exceed a certain value (which can be taken as the maximal constant c_{max} in the description of the automaton) are considered equivalent. Formally, we have:

Definition 15 *Two clock valuations v and v' are equivalent ($v \simeq_{\text{reg}} v'$) iff they satisfy the following three conditions:*

1. *For all $x \in C$, $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or both $v(x) > c_{max}$ and $v'(x) > c_{max}$.*
2. *For all $x, y \in C$ with $v(x) \leq c_{max}$ and $v(y) \leq c_{max}$, $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.*
3. *For all $x \in C$ with $v(x) \leq c_{max}$, $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$.*

It is easily shown that the above conditions define an equivalence relation. A *clock region* is an equivalence class of clock valuations with respect to $\simeq_{\text{reg}}$. We denote by $[v]$ the clock region to which valuation v belongs. A *region* is then a pair $\langle s, [v] \rangle$ of a control state s and a clock region $[v]$.

It can be shown that the region equivalence relation is stable, that is, two timed states that belong to the same regions have the same set of enabled transitions. Consequently, one can then define a transition relation between two regions as follows:

- $\langle s, [v] \rangle \xrightarrow{a} \langle s', [v'] \rangle$ iff $(s, v) \xrightarrow{a} (s', v')$
- $\langle s, [v] \rangle \xrightarrow{\delta} \langle s, [v'] \rangle$ iff $\exists t \in \mathbb{R}^+$ such that $(s, v) \overset{t}{\rightsquigarrow} (s, v')$, and the interval $[0, t]$ can be partitioned into two intervals I_1 and I_2, such that $[v + t'] = [v]$ for $t' \in I_1$ and $[v + t'] = [v']$ for $t' \in I_2$.

In the first case, two regions are connected by an action transition if there exists such a transition between two representative timed states, one from each region. For the second case, recall that a timed automaton allows transitions of arbitrary amount, as long as the state invariant is satisfied. In the second case, a transition $\xrightarrow{\delta}$ exists between two regions if there exists a delay transition between two representative timed states that does not traverse other regions. (In an alternate definition for the region graph automaton, a transition between regions corresponds to the combination of an action and delay transition in the underlying timed automaton).

The fine granularity of the regions leads to state space explosion in the region graph automaton, compared to the zone automaton which is more coarse-grained and generally smaller. Consider, for instance, a clock valuation v such that $\{v(x_1)\} < \{v(x_2)\} < \ldots < \{v(x_n)\}$, and the clock valuation $v + 1$ obtained from v after passage of one time unit. The intermediate clock valuations on this delay transition belong to $2|C|$ different regions, as each of the fractional parts of $x_n, \cdots, x_2, x_1$ becomes successively zero, and then nonzero but smallest in sequence.

This is a significant increase in the number of transitions. In the zone automaton, an action transition is followed by an arbitrary amount of time. However, in the region graph, a number of delta transitions can be executed successively, each advancing to a new region, until eventually an action transition is taken. The exploration of some interleavings between action transitions and those that correspond to passage of time can be avoided by using partial order reduction.

Consider an action transition $\xrightarrow{a}$ that does not reset any clocks. We first examine in detail the circumstances under which transitions $\xrightarrow{a}$ and $\xrightarrow{\delta}$ can disable one another:

Proposition 8 *The enabling of action and delay transitions in successor regions is related as follows:*

- $\xrightarrow{a}$ *can disable* $\xrightarrow{\delta}$ *in* $\langle s, [v] \rangle$ *iff one of the following holds:*
  - *the successor state* s' *with respect to* a *has an invariant of the form* $x \leq c$, *and* $v(x) = c$.
  - *the successor state* s' *with respect to* a *has an invariant of the form* $x < c$, $\lfloor v(x) \rfloor = c - 1$, $\{v(y)\} > 0$ *for all* $y \in C$ *and* $\{v(x)\} \geq \{v(y)\}$ *for all* $y \in C$.
- $\xrightarrow{\delta}$ *can disable* $\xrightarrow{a}$ *iff one of the following holds:*
  - $\xrightarrow{a}$ *has a constraint of the form* $x \leq c$, *and* $v(x) = c$.
  - $\xrightarrow{a}$ *has a constraint of the form* $x < c$, $\lfloor v(x) \rfloor = c - 1$, $\{v(y)\} > 0$ *for all* $y \in C$ *and* $\{v(x)\} \geq \{v(y)\}$ *for all* $y \in C$.

Proof: Since $\xrightarrow{a}$ does not reset any clocks, the clock valuations in s and s' after executing a are the same. We examine first the cases where $\xrightarrow{a}$ disables $\xrightarrow{\delta}$. This means that passage of time, which is allowed in control state s, is no longer allowed in state s'. This occurs when the advance of some clock x is limited by an invariant of the form $x \prec c$ in state s', and when advancing time to the next region by means of transition $\xrightarrow{\delta}$ would result in a region that no longer satisfies this invariant. If the constraint in the invariant is nonstrict, $x \leq c$, $[v]$ has to be a boundary region with $v(x) = c$, otherwise an incremental advance of time to the next region will still satisfy $v'(x) \leq c$. If the constraint in the invariant is strict, $x < c$, the invariant will no longer be satisfied if the successor region is the boundary region with $v'(x) = c$. This happens when $\lfloor v(x) \rfloor = c - 1$, and $\{v(x)\}$ is the next fractional part to wrap around to zero, i.e., $\{v(x)\} \geq \{v(y)\}$ for all $y \in C$. In addition, the current region must not itself be a boundary region with some $\{v(y)\} > 0$. Otherwise, the next region is obtained by an infinitesimal advance of time, which increases $\{v(y)\}$ from 0 to positive while maintaining $v(x) < c$.

For the case when $\xrightarrow{\delta}$ disables $\xrightarrow{a}$, the given conditions are analogous, with the enabling condition of the transition replacing the invariant of the destination state. The argument is completely similar. $\Box$

Proposition 9 *If $\xrightarrow{a}$ and $\xrightarrow{\delta}$ are two transitions enabled in region r, none of them disables the other, and $\xrightarrow{a}$ does not reset any clocks, then $\xrightarrow{a}$ and $\xrightarrow{\delta}$ are independent in region $\langle s, [v] \rangle$.*

Proof: The proof follows from the fact that for any $t \in \mathbb{R}^+$, the transitions $\xrightarrow{a}$ and $\overset{t}{\rightsquigarrow}$ commute in (s, v) if neither disables the other and $\xrightarrow{a}$ does not reset any clocks. This is obvious, since $\xrightarrow{a}$ only changes the control location and $\overset{t}{\rightsquigarrow}$ only changes the clock valuation. $\Box$

Based on this dependence relation, partial order reduction can be used in the construction of a smaller region graph for a given timed automaton. Ordinarily, even at a state where only a single transition is enabled, the region graph construction would have to consider either executing the transition or advancing time to the next region. For transitions that do not reset clocks, this method allows the exploration of only one possibility, except for the case when the execution of the transition is forced at the end of its enabling interval. (The other case, where a time invariant is strengthened in the successor state rarely appears in practice).

As opposed to the local time model, this method does not make use of the structuring of a system into components, and can be used on a single timed automaton. Furthermore, the region graph, being time-abstract can be represented symbolically using binary decision diagrams (BDDs). Thus, if a static technique is used for partial order reduction, this method can potentially combine partial order reduction and symbolic model checking.

4.2 Partial Order Reduction for Timed Event/Level Structures

A model of timed systems which is well suited for describing hardware circuits, in particular asynchronous ones, is provided by the so-called *timed event/level (TEL) structures*. This model can express both event causality, as well as dependence on signal levels. Early work by Rokicki and Myers [RM94] gave an algorithm that reduced the number of geometrical timing regions generated during state space search. This approach was later extended by Belluomini and Myers [BM98] using so-called partially ordered sets of events (POSETs). We show how to apply partial order reduction to this model and obtain additional savings in the generated control state space.

4.2.1 Timed Event/Level Structures

We start with a presentation of timed event/level structures and the POSET algorithm, following the account given in [BM98]. A timed event/level (TEL) structure is a tuple $T = \langle N, S_0, A, E, R, \# \rangle$, where:

- N is the set of (boolean) signals,
- $S_0 \subseteq \{0,1\}^N$ is a set of initial states, specified by a boolean value for each signal,
- $A \subseteq N \times \{+,-\} \cup \$$ is the set of actions,
- $E \subseteq A \times \mathbb{N}$ is the set of events, where $\mathbb{N}$ is the set of natural numbers,
- $R \subseteq E \times E \times \mathbb{N} \times (\mathbb{N} \cup \{\infty\}) \times \mathcal{B}(N)$ is the set of rules, where $\mathcal{B}(N)$ is the set of boolean functions $b : \{0,1\}^N \rightarrow \{0,1\}$,
- $\# \subseteq E \times E$ is the (symmetric) conflict relation between events.

An action $a \in A$ can be either a rising or a falling transition of a signal $x \in N$. There is also the dummy action \$ which does not result in any signal transition. An event $e \in E$ is a pair $\langle a, i \rangle$, with $a \in A$ and $i \in \mathbb{N}$, denoting the i^{th} occurrence of action a. A rule $r \in R$ is a tuple of the form $\langle e, f, l, u, b \rangle$, where e is the event enabling the rule, f is the event enabled as effect of the rule, (l, u) is a pair of upper and lower integer time bounds, and the enabling condition $b \in \mathcal{B}(N)$ is a boolean function on signal values.

The semantics of TEL structures can be described informally as follows: A rule becomes *enabled* once its enabling event has occurred and its boolean enabling condition is true for the current signal assignment. After the lower time bound l passes since the enabling of a rule, the rule is called *satisfied*; from this time point on, the rule can *fire*. After the passage of the upper time bound u since its enabling, a rule becomes *expired*. In the absence of conflicts, an event has to occur after all rules enabling it are satisfied, and before any of them expires. Should a rule's boolean enabling condition become false after the rule is enabled, this constitutes a hazard and represents a failure during verification.

The conflict relation # can be used to model choice and disjunctive behavior. If two events e_1 and e_2 are marked as being in conflict, $e_1 \# e_2$, one of the two can occur, but not both. If two rules r_1 and r_2 have the same enabling event e, but conflicting events $e_1 \# e_2$ as effect, then only one of the rules can fire, causing the corresponding effect to occur. This models nondeterministic choice. Conversely, if an event e appears as an effect of two rules with conflicting enabling events, only one of these events needs to happen (and only one rule needs to fire) for the effect e to occur.

4.2.2 State Space Exploration Using POSETs

We next describe the data structures and the exploration algorithm used in the POSET approach of Belluomini and Myers [BM98], to establish a comparison point for the application of partial order reduction. In TEL structures, a timed state is represented as a tuple $(s_{\mathrm{c}}, R_{\mathrm{m}}, M, R_{\mathrm{f}})$, where:

- s_{c} is the control state representing the values of the signals,
- R_{m} is the set of *marked* rules, whose enabling event has occurred,
- M is the *constraint matrix*, a difference bound matrix containing the maximum differences between the enabling times of all enabled rules
- R_{f} is the set of rules that have already fired

The set of marked rules R_{m} together with values of the signals in s_{c} determine the set of enabled rules R_{en}. These are the rules for which timing information is maintained in the constraint matrix M. For the fired rules in R_{f}, no timing information about them needs to be maintained in the constraint matrix, but the fact that they have fired must be recorded.

A state space exploration step in a TEL structure consists of determining the set of satisfied rules R_{s}, choosing a satisfied rule to fire, and computing the resulting new timed state. A depth–first search of the state space would consider in turn the firing of each rule among the satisfied rules in R_{s}. However, each interleaving of rule firings would typically generate a different constraint matrix M (that is, a different timing region), leading to an exponential number of different timed states. The POSET method generates a timed state space consisting of fewer and larger timing regions. To this effect, the algorithm maintains in addition to the constraint matrix (which contains separation times between enabled rules) another difference bound matrix, called POSET matrix, which keeps track of relationships between event firing times that are allowed by the given rule firing sequence. As a result, the timing behaviors represented in the constraint matrix are only constrained by the causality in the firing sequence, and no longer by its total order, resulting in a significantly reduced number of timed states.

However, the method still requires multiple rule interleavings to be explored, even though with the use of POSETs the same timing region is generated in the state space. Also, some computation steps for the constraint matrix still take into account the chosen total order of rule interleavings, which results in unnecessary overhead. In the following, we present the POSET algorithm by working through an example which showcases both its strengths and limitations, and finally present an improved algorithm which takes advantage of partial order reduction.

The POSET algorithm decouples rule firing from event firing: A rule can fire as soon as it is satisfied, i.e., it has been enabled for at least its lower time bound. An event fires only once all its enabling rules have fired. The *causal rule* r_{c} for an event e is therefore the last rule that fires and consequently enables the event. Conversely, the *causal event* for a rule $r = \langle e_{\mathrm{c}}, e, l, u, b \rangle$ can be either the enabling event e_{c} or some later event that causes the enabling condition b to be satisfied. Finally, note that the *causal event* e_{c} of an event e is the causal event of its causal rule r_{c}, and the minimum and maximum separation times between e_{c} and e are consequently given by r_{c}.

Taking these causality relations into account, the POSET algorithm proceeds as follows: from the timed state $(s_{\mathrm{c}}, R_{\mathrm{m}}, M, R_{\mathrm{f}})$, the set of satisfied rules is computed and a rule r that can fire first among these is selected. The rule r is removed from the set of marked rules R_{m} and added to the set of fired rules R_{f}. Next, the algorithm checks whether as a result of firing r any event can fire. If yes, the untimed state is updated, the enabling rules of the event are removed from R_{f}, and any conflicting rules are removed from R_{m} and R_{f}. Finally, the POSET matrix is updated and the new event separations are used to update the constraint matrix.

When adding a new event e to the POSET matrix, the separation times to the events that influence e (and therefore exist in the POSET matrix) must be taken into account. This includes the causal event of e, the enabling events of any rules that enable e, and the events occurring in the boolean conditions of these rules. Determining these separation times is straightforward and is described in detail in [BMH99]. The separation times between the new event e and any other events in the POSET matrix are simply a consequence of existing separation times and are computed by canonicalizing the matrix using the all–pairs shortest paths algorithm. After this step, all events which are no longer relevant to the evolution of the system (i.e., are not causal for any of the marked rules in R_{m}) are removed from the matrix.

As a last step, all rules enabled by the firing of the new event need to be added to the constraint matrix M. Since the enabling time of a rule is simply the timepoint of its enabling event, the needed minimum and maximum separation times between the new rules and the existing ones can simply be copied from the POSET matrix. The constraint matrix is then canonicalized, which can further constrain some of its entries, since the age of a rule cannot exceed its maximum bound u. Finally, the rule whose firing caused this computation step (and which is thus no longer in R_{en}) is removed from the constraint matrix.

We illustrate the application of the POSET algorithm by means of a small example, taken for purposes of comparison from [BM98]. Figure 4.1 depicts a timed event/level structure, in which events are represented as nodes and rules as directed edges (labeled with time bounds) connecting them. For simplicity, no level dependencies are included in this case, which means that all boolean conditions of the rules are true. Thus, the sole triggering condition for a rule is its enabling event.

Initially, event A has just fired, and the set of marked (and enabled) rules is $R_{\mathrm{en}} = \{\langle A, B\rangle, \langle A, C\rangle\}$ (we can unambiguously denote a rule by its triggering and resulting events). The POSET matrix is trivial and contains the single event A. The constraint matrix compares the *ages* of the enabled rules, i.e., the amount of time passed since each rule has been enabled. These are quantities that increase at the same rate with passage of time, just like the clocks in a timed automaton. Similarly, the matrix contains a dummy clock which has always age 0.

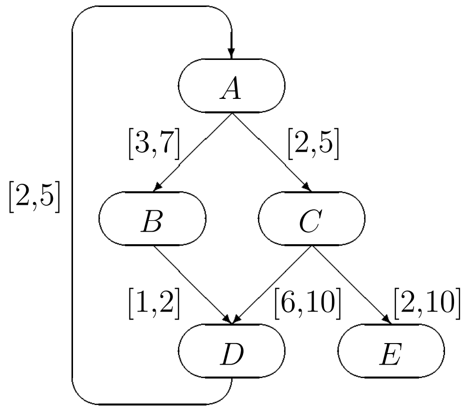

Figure 4.1: Sample timed event/level structure

The representation defined in [BM98], which we observe for reasons of consistency, defines the matrix entry m_{ij} to be $c_j - c_i$, where c_i is the age of the rule r_i. Thus, rows and columns are swapped compared to the usual DBM representation. In an alternate view, we can state that $m_{ij} = t(e_i) - t(e_j)$, where e_i is the causal event for rule r_i and $t(e_i)$ its firing time. In this case, the zero row and column denotes the current time t, and $m_{0i} = t - t(e_i)$.

The entries in row 0 are thus set to the maximum possible age for each rule, given by its upper bound u, since the constraint matrix contains rules which have not yet fired. In this case, both rules are enabled by the same event A and therefore have identical enabling times, $m_{AB,AC} = m_{AC,AB} = 0$. We have $t - t(A) \leq 7 = m_{0,AB}$ due to rule $\langle A, B\rangle$, and $t - t(A) \leq 5 = m_{0,AC}$ due to rule $\langle A, C\rangle$. The latter bound is stronger and thus constraining for both rules, after the matrix is canonicalized. The elements of column 0 are 0, since the only constraint on the ages of rules is that they be positive. The state of the TEL structure is therefore as follows:

Constraint matrix

$$\begin{array}{c|ccc} & 0 & \langle A,B\rangle & \langle A,C\rangle \\ \hline 0 & 0 & 5 & 5 \\ \langle A,B\rangle & 0 & 0 & 0 \\ \langle A,C\rangle & 0 & 0 & 0 \end{array}$$

POSET matrix

$$\begin{array}{c|c} & A \\ \hline A & 0 \end{array}$$

Next, either rule $\langle A, B\rangle$ or rule $\langle A, C\rangle$ can fire. Consider first the firing of $\langle A, B\rangle$ which causes event B to occur. Event B is added to the POSET matrix, with rule $\langle A, B\rangle$ giving the minimum and maximum separation times of 3 and 7 from event A: $3 \leq t(B) - t(A) \leq 7$. Rule $\langle B, D\rangle$ triggered by the new event B is added to the constraint matrix and rule $\langle A, B\rangle$ which has fired is removed. The new constraints are $t(A) - t \leq m_{AC,0} = -3$ (at least 3 time units have passed since A, since B has fired), $m_{0,BD} = 2$ (maximum firing time of rule $\langle B, D\rangle$), and $t(A) - t(B) \leq m_{AC,BD} = -3$ (again, due to the firing of B after A). The remaining entry $m_{BD,AC} = 5$ is obtained from canonicalization, which reduces it compared to $t(A) - t(B) \leq 7$ from the POSET matrix. The resulting state is:

Constraint matrix

	0	$\langle A, C\rangle$	$\langle B, D\rangle$
0	0	5	2
$\langle A, C\rangle$	-3	0	-3
$\langle B, D\rangle$	0	5	0

POSET matrix

	A	B
A	0	-3
B	7	0

In this state, either rule $\langle A, C\rangle$ (implying event C) or rule $\langle B, D\rangle$ can fire and we explore the firing of the former. Event C is added to the POSET matrix, with a separation time from A between 2 and 5, given by the fired rule. At this point, all rules triggered by A have fired and the event can be removed from the POSET matrix. The remaining separations in this matrix are: $t(B) - t(C) = (t(B) - t(A)) - (t(C) - t(A)) \leq 7 - 2 = 5$ and $t(C) - t(B) = (t(C) - t(A)) - (t(B) - t(A)) = 5 - 3 = 2$. Likewise, the fired rule $\langle A, C\rangle$ is removed from the constraint matrix and the two rules newly enabled by event C are added to it. The new constraint is $t - t(B) \leq 2 = m_{0,BD}$, from the upper firing bound of rule $\langle B, D\rangle$. By canonicalization, we obtain $m_{0,CD} = m_{0,BD} + m_{BD,CD} = 2 + 5 = 7$. Finally, the last two rows and columns are identical, since their rules have the same causal event.

Constraint matrix

	0	$\langle B, D\rangle$	$\langle C, D\rangle$	$\langle C, E\rangle$
0	0	2	7	7
$\langle B, D\rangle$	0	0	5	5
$\langle C, D\rangle$	0	2	0	0
$\langle C, E\rangle$	0	2	0	0

POSET matrix

	B	C
B	0	5
C	2	0

Two characteristics of the POSET method become apparent at this step. First, even though in the current rule firing sequence B happens before C, the POSET matrix does not contain this restriction. The separation times between B and C in the POSET matrix are only determined by their causal dependence on A. Second, this is also true of the constraint matrix, which also contains all timing assignments allowed by the *causality* in the firing sequence, in particular, assignments where C fires *before* B.

The exploration process would continue here using the same algorithm. Once both rules $\langle B, D\rangle$ and $\langle C, D\rangle$ have fired, event D fires, and in this case two cases must be analyzed, depending on whether B or C is causal. We will return to this example in the next section, to illustrate how the POSET algorithm can be improved by using partial orders.

4.2.3 An Improved Algorithm for TEL Structures

Besides its improvements in reducing the number of generated timing regions, the POSET algorithm still suffers from inefficiencies. First, the method still has to exploit redundant interleavings of rule firing sequences. For instance, in the example above, after choosing $\langle A, B\rangle$ to fire ahead of $\langle A, C\rangle$, the algorithm still has to consider the alternate interleaving, which in the POSET approach leads to the same timing region. A second overhead resulting from firing rules in a total order is that time separations which are copied from the POSET matrix to the constraint matrix have to be adjusted to account for the fact that the age of a rule cannot exceed its upper time bound. In fact, the constraint and POSET matrices duplicate a significant amount of information. We address these issues in a new algorithm. Optimizations to remove redundant rule interleavings are also discussed in the thesis of Belluomini [Bel99], with the goal of generating only one POSET matrix per causal rule. However, they seem limited to certain timing conditions, whereas we address the problem in the general framework of partial orders.

As before, denote by s_{c} the state of the signals in the model, by R_{m} the set of marked rules (whose enabling event has fired), and let E_{m} be the set of events enabling these rules. An event is added to E_{m} as it fires, and removed when all the rules enabled by it have either fired or expired. We maintain information about the time separation of events from E_{m} in a difference bound matrix M_{e} which we call *event matrix* and which serves the same role as the POSET matrix in the approach presented above. Since the separation time between rules is determined directly by the separation times between the corresponding events, we will attempt as much as possible to avoid the inclusion of rule timings in the data structures describing a state.

To apply partial order reduction, we next need to define the key notion of *visibility* and *dependence* for system transitions (i.e., event occurrences). We focus on the verification of next-time free linear temporal logic, and assume that the atomic propositions are defined in terms of signal values and time differences between events. Then, a visible transition is either an event on a signal mentioned in the specification, or an event that appears in a time constraint in the specification. All other events produce changes in the timed state that are the specification cannot observe.

Let us examine the dependence relation between events. Clearly, two events are dependent if they are defined as being in conflict, $e_1 \# e_2$. (If they are both caused by a rule with the same enabling event, the conflict relation specifies that only one of them can happen). Whether this completely defines the dependence relation depends on the *disabling* or *non-disabling* semantics [BM97] adopted for the TEL structure. In the non-disabling semantics, once a rule is enabled, it cannot become disabled because of a change in state. In the disabling semantics, an enabled rule can become disabled because of another event that causes its boolean condition to become false. In the latter case, for an event to fire, all of the rules causing an event need to be continuously enabled up to its firing time.

Denote by $\mathit{disable}(e)$ the set of events that can disable an event e. In the non-disabling semantics, we have $\mathit{disable}(e) = \{e' \mid e \# e'\}$, since except for choice conflicts, nothing can disable any of the rules causing e, once they are enabled. In the disabling semantics, e can also be disabled by an event that falsifies the boolean condition on a rule enabling e. We approximate this conservatively with the set of all events on the signals appearing in the boolean conditions of these rules (a more detailed analysis of these conditions may restrict this set on a case by case basis). Formally, define $\mathit{disable}(e) = \{e' \mid e \# e'\} \cup \{s\pm \mid (e', e, l, u, b) \in R \textit{ and } s \textit{ appears in } b\}$, where $s\pm$ denotes an arbitrary rising or falling event on signal s. We call events e_1 and e_2 independent if $e_2 \notin \mathit{disable}(e_1)$ and $e_1 \notin \mathit{disable}(e_2)$. Here, the definition of *disable* ensures the enabledness condition, whereas the commutativity condition is trivially satisfied since the effect of an event on a state is merely to toggle a signal.

Having defined the notions of visibility and dependence, we can proceed to define an ample set of transitions to explore at a given state s. We need to ensure condition **C1**, i.e., that a transition which conflicts with an ample transition is either enabled and included in the ample set, or disabled and cannot be enabled without executing an ample transition. To guarantee this, we adapt the approach taken by Valmari for stubborn sets [Val90] and then by Yoneda et al. [YS97] for time Petri nets.

An event e' is *relevant* for the execution of another event e at a given state if either e and e' are dependent or if both are visible. To handle relevant events which are disabled at the current state, we say that a set of events E_{n} is *necessary* for a disabled event e at a given state if e cannot be enabled without executing an event from E_{n} first. In general, an event e is enabled by multiple rules. If these rules have non-conflicting enabling events, then by definition, all of these events have to fire in order for e to fire. Thus, any enabling event of e forms a necessary set by itself. If some of the enabling events are conflicting, several of them may have to be chosen to form a necessary set.

For every event e' which is relevant to another event e we consider a set $necessary^*(e')$ which contains e' and is transitively closed under necessity, i.e., if $e'' \in necessary^*(e')$ is disabled, there exists a set of events E_{n} which is necessary for e'' and included in $necessary^*(e')$. Finally, a set of events $dependency(e)$ is called a dependency set for e if for any event e' which is relevant for e there exists a set $necessary^*(e')$ for which all enabled transitions belong to $dependency(e)$.

In general, including $dependency(e)$ in the ample set together with any ample event e is sufficient to guarantee condition **C1**. However, a timed system has characteristics that make it possible to define smaller ample sets than in the untimed case [YS97]. Specifically, of all the events that can occur at a given timed state, only a subset can occur *before* any other event. We call such an event *firable*, since it can fire first, before any other event. Since an event sequence executed from a given timed state can only start with an event which is firable at that state, our ample set will also consist only of firable events. We can therefore modify a procedure to select an ample set in an untimed system as follows:

1. Start with a $ample(s) = \{e\}$ for some firable invisible event e. If there is none, simply return the set of all firable events as an ample set.

2. For any event e' that belongs to $dependency(e)$ for some $e \in ample(s)$, if e' is firable before all events from $ample(s)$, add e' to $ample(s)$. Iterate until a fixpoint is reached.

Every event added to $ample(s)$ by the above algorithm is firable. The transitive closure operation in step 2 ensures that all firable events which might eventually lead to an event dependent of an ample event belong to the ample set, and **C1** is satisfied. Likewise, the ample set contains at least one invisible transition if one exists, and includes in step 2 all visible transitions if one is included. This ensures condition **C2**. Condition **C3** is ensured dynamically in case of depth–first search, or using static partial order reduction.

We can also choose an ample set that contains non-firable events if all events in the ample set are invisible. In this case, the invisibility condition ensures that no additional behaviors are added by exploring a non-firable event first. However, with such a choice, an actual reduction of the state space is not guaranteed, since the ample set algorithm may explore transitions which are not firable in the original system.

With the selection of ample sets in place, the partial order exploration proceeds as follows. A timed state is a tuple $(s_{\mathrm{c}}, R_{\mathrm{m}}, E_{\mathrm{m}}, M_{\mathrm{e}})$, consisting of the signal state, the set of marked rules, the marked events and the event matrix containing their time separations. From this timed state, one determines the set of enabled events and an ample set. Each event e in the ample set is selected in turn for firing, assuming a depth–first search. Once the event fires, it is added to E_{m} and to the event matrix M_{e} with the appropriate timing separations given by its enabling rules. Next, these rules are removed from R_{m} and the rules whose enabling event is e are added to R_{m}. Finally, any event that is no longer enabling for any of the rules in R_{m} is removed from E_{m} and the event matrix.

Our algorithm no longer considers the rule firing times explicitly and separately from the firing of events. However, it still has to take into account which rule is causal to the firing of an event. Recall that a rule can fire anytime after it has been enabled for its lower time bound, and before its upper time bound expires. An event fires at the same time as its last enabling rule. In our algorithm, this is done as follows. Consider an event e enabled by k rules, $r_i = \langle e_i, e, l_i, u_i, b_i \rangle$, with $1 \leq i \leq k$, and let $t(r_i)$ be their firing times. Since the event e fires after all of its enabling rules, we have $t(e) \geq t(r_i)$, and the lower bounds on the rules imply $t(e) - t(e_i) \geq t(r_i) - t(e_i) \geq l_i$, for $1 \leq i \leq k$. Potentially, each of the rules r_i can be causal, in which case we also have $t(e) = t(r_i)$, and the upper bound implies $t(e) - t(e_i) = t(r_i) - t(e_i) \leq u_i$. Considering each causal rule separately, we generate potentially k different successor regions (some may be empty, overlap or generate convex unions). This procedure shows another advantage of our approach: if multiple rules enable the same event, we only need to distinguish which rule fires last, instead of generating all interleavings.

To illustrate the algorithm using the same example as in the previous section, consider the state reached after firing B and C. We have $E_{\mathrm{m}} = \{B, C\}$, $R_{\mathrm{m}} = \{\langle B, D\rangle, \langle C, D\rangle, \langle C, E\rangle\}$, and the event matrix is:

	B	C
B	0	5
C	2	0

Next, events D and E can fire, and either of them can fire first. If we select D, we need to analyze the possible causal events, B and C. If B is causal, the bounds on the rules imply $1 \le t(D) - t(B) \le 2$ and $6 \le t(D) - t(C)$. The upper bound on $t(D) - t(C)$ is obtained by canonicalizing the matrix, $t(D) - t(C) = t(D) - t(B) + t(B) - t(C) \le 2 + 5 = 7$. If C is causal, we have $1 \le t(D) - t(B)$ and $6 \le t(D) - t(C) \le 10$, and $t(D) - t(B) = t(D) - t(C) + t(C) - t(B) \le 10 + 2 = 12$ is obtained from canonicalization:

B causal to D

	B	C	D
B	0	5	-1
C	2	0	-6
D	2	7	0

C causal to D

	B	C	D
B	0	5	-1
C	2	0	-6
D	12	10	0

In this particular case, the region obtained is a superset of the one above. In both cases, since all rules with enabling event B have fired, B can be now removed from the event matrix.

A note about the computation of ample sets. Since we need to ensure that each ample event is firable, this entails adding all enabled events to the the event matrix (which contains all relevant fired events) and checking which event can fire first. Of all enabled events, only the one currently fired needs to be retained in the matrix, yet all others may need to be added again when the check for firable events is done in the next exploration step. To avoid the recomputation of separation times, we can manipulate during state space search a matrix that contains separations between both past events and currently enabled rules (events). However, only the event matrix proper (containing fired events) needs to be stored in the set of reached states. This results in savings over the use of the constraint matrix in the POSET approach.

Chapter 5

A Partial Order Reduction Framework for Timed Systems

5.1 Background and Motivation

In this chapter, we present a general method for applying partial order reduction to timed systems. Our goal is to compare and unify the various approaches to partial order reduction that have been employed so far for models such as time Petri nets, timed automata and timed event/level structures. We identify a common approach to partial order reduction, and present how some of the discussed algorithms could benefit from it.

We use a general timed model, for which we present a trace-based semantics which relaxes some constraints on the time ordering of transitions. This avoids unnecessary dependencies related to timing. We show how this relaxed semantics can be used with an exploration algorithm based on timed regions, and how the semantics naturally leads to the application of partial order reduction. Finally, we discuss how this framework can be particularized for some commonly used timed models.

The approaches to partial order reduction for timed systems have so far been quite diverse, and at the same time heavily dependent on the choice of the model. We briefly reexamine and compare the commonalities and differences in some of these methods.

One of the first approaches has been presented by Yoneda, Schlingloff et al. [YSSC93, YS97] for time Petri nets, which have a lower and upper firing bound associated with each transition. Time variables are introduced for firing times of transitions and for the timepoints when a place receives or loses a token. The state space exploration algorithm operates on regions (called *atoms*) which consist of a marking of the net and a conjunction of difference inequalities over the time variables of the net. The reduced set of transitions chosen for exploration (called *ready set*) is adapted from the stubborn sets of Valmari [Val90] for untimed Petri nets.

Despite exploring a reduced set of transitions, the partial order algorithm still accounts for all execution sequences by using less restrictive timing constraints. Without partial orders, each explored transition has to fire at an earlier time than any other enabled transition. This serialization constraint causes the generation of a distinct timed region for each transition interleaving. In contrast, the partial order algorithm only requires a transition from the ready set to fire earlier than any other transition from the ready set.

Avoiding a specific time ordering for independent transitions is a very general approach. However, the correctness proof in [YS97] relies heavily on the particular form of the constraints in time Petri nets. Moreover, the proof is complicated by the fact that the state space explored using partial orders is not a subset of the original one. Lilius [Lil98] proposes to obtain better reduction by not storing *any* information on transition firing order in the timed state. However, this approach only preserves the reachable markings of the net, and not the timing information.

The POSET approach to the verification of timed event/level structures operates, as its name states, on partially ordered sets of events. However, as discussed in Chapter 4, it is still effectively based on exploring a total order of rule firings, a fact which is reflected in its dual data structures for rules and events. Optimizations presented in [Bel99] avoid redundant rule interleavings in some cases, but are based on specific details of the event/level model, rather than on a general notion of partial orders.

For timed automata, the initial approaches of Pagani [Pag96, Pag97] as well as of Dams et. al [DGKK98] offer relatively little potential for reduction, because the global passage of time leads to inherent transition dependencies. The local time model of Bengtsson et al. [BJLW98], extended in Chapter 3 removes this synchronization and restores the transition independence of the underlying untimed system. It applies the same general principle as [YS97], allowing independent transitions to be explored without being serialized in time. Yet, the approach relies on the system's structure as product of parallel components, setting it off from Petri nets and TEL structures. Moreover, it depends on the fact that clocks cannot be shared between automata.

In the following, we define an approach to partial order reduction technique which encompasses and refines the fundamental ideas mentioned above, and apply it to a generic timed model, which uses the basic notions of timed states and timed transitions.

5.2 Timed Structures and Traces

Definition 16 *A timed structure is a tuple* $Q = (S_\mathrm{t}, S_\mathrm{t}^0, T, N)$*, where:*

- S_t *is a set of* timed states
- $S_\mathrm{t}^0 \subseteq S$ *is a subset of* initial timed states
- T *is a finite set of* transitions
- $N : S_\mathrm{t} \times (\mathbb{R}^+ \times T) \to S_\mathrm{t}$ *is a partial next-state function that defines a set of* timed transitions (t, a) *with* $t \in \mathbb{R}^+$ *and* $a \in T$*.*

Consider a state $s \in S_\mathrm{t}$, a transition $a \in T$ and a timepoint $t \in \mathbb{R}^+$. If $(s, t, a) \in \mathrm{dom}\ N$, we say that transition a can be taken from state s at timepoint t, and leads to state $s' = N(s, t, a)$. We denote this by $s \stackrel{t,a}{\to} s'$.

A timed transition (t, a) is *enabled* at state $s \in S_\mathrm{t}$ if for some state $s' \in S_\mathrm{t}$ we have $s \stackrel{t,a}{\to} s'$. As before, we denote the set of all such transitions by $\mathit{enabled}(s)$. A transition a is *future enabled* at state s and time t if it can be executed at some timepoint $t' \geq t$. We denote this set by $\mathit{enabled}^+(s, t) = \{a \in T \mid \exists t' \geq t \,.\, (t', a) \in \mathit{enabled}(s)\}$. The upper bound on the firing time of a at s is denoted by $\mathit{fire}_\mathrm{max}(a, s) = \sup\ \{t \in \mathbb{R}^+ \mid (t, a) \in \mathit{enabled}(s)\}$. We write $t \prec \mathit{fire}_\mathrm{max}(a, s)$ to denote a strict inequality if the upper firing bound is not reached, and a non-strict inequality otherwise.

A model for a timed structure Q is a state-transition graph, defined by means of its execution traces, on which we impose two conditions. First, the execution times of transitions have to form a monotonically increasing sequence. Second, an enabled transition has to fire if it is not disabled before its maximum firing time. Consequently, *some* transition has to fire at a state before the maximum firing time of *any* enabled transition elapses.

For example, assume that the system has reached the timed state s at timepoint $t = 1$, and that the enabled transitions are a and b with upper bounds of 5 and 7, respectively. Then, the next transition has to be executed from state s before or at timepoint 5, which is the smallest of the two upper bounds. For instance, the next transition cannot be b at timepoint 6, since a would have had to fire earlier than that.

Thus, if state s is reached at time t, the firing time t' of the next transition has to satisfy $t \leq t' \prec \mathit{fire}_{\max}(a, s)$, for all $a \in \mathit{enabled}^+(s, t)$.

Definition 17 *The family $\mathcal{L}_s(Q)$ of execution traces of a timed structure Q contains all infinite sequences $\sigma = s_0 \stackrel{t_1,a_1}{\rightarrow} s_1 \stackrel{t_2,a_2}{\rightarrow} s_2 \ldots \stackrel{t_i,a_i}{\rightarrow} s_i \ldots$, such that $t_i \leq t_{i+1} \prec \mathit{fire}_{\max}(a, s_i)$ for all $i \geq 0$, $a \in \mathit{enabled}^+(s_i, t_i)$ (where $t_0 = 0$).*

In the following, we restrict our attention to non-Zeno traces, in which only a finite number of transitions can occur within any finite interval. Consequently, in any non-Zeno trace, time grows unbounded towards infinity.

5.3 A Relaxed Timing Semantics

5.3.1 Preliminaries

In practice, state space exploration algorithms operate on sets of timed states, usually called timed regions, which are represented using timing constraints. Requiring a strict time ordering of explored transitions causes transitions to be serialized even if they are independent. As a result, supplementary constraints on transition ordering are added to the representation of a timed region. Thus, a distinct timed region is generated for each interleaving of transitions, leading to an explosion in the number of generated regions.

We approach this problem by defining a modified semantics for a timed structure, which relaxes some of the time ordering constraints specified for the traces in $\mathcal{L}_s(Q)$. Recall that in a trace $\sigma = s_0 \stackrel{t_1,a_1}{\rightarrow} s_1 \stackrel{t_2,a_2}{\rightarrow} s_2 \ldots \stackrel{t_i,a_i}{\rightarrow} s_i \ldots$ from $\mathcal{L}_s(Q)$, each subsequent timed transition $\stackrel{t_{i+1},a_{i+1}}{\rightarrow}$ has to satisfy:

- a relative ordering condition on transition timings: $t_i \leq t_{i+1}$
- a bound on the firing time: $t_{i+1} \prec \mathit{fire}_{\max}(a, s_i)$, for all $a \in \mathit{enabled}^+(s_i, t_i)$

We will now give similar, but less restrictive conditions for our new semantics, and discuss them intuitively before giving a formal proof.

First, the relaxed semantics must preserve time ordering due to causality. Given an execution trace $\sigma = s_0 \stackrel{t_1,a_1}{\rightarrow} s_1 \stackrel{t_2,a_2}{\rightarrow} s_2 \ldots \stackrel{t_n,a_n}{\rightarrow} s_n \ldots$ we define $\mathit{caused}(a_i)$ to be the set of transitions which become enabled as a result of executing a_i: $\mathit{caused}(a_i) = \{a \in T \mid a \notin \mathit{enabled}(s_{i-1}) \wedge a \in \mathit{enabled}(s_i)\}$.

We say that transition a_i is *causal* to a_j, with $i < j$, if $a_j \in \mathit{caused}(a_i)$ and $a_j \in \mathit{enabled}(s_k)$ for $i \leq k < j$. In other words, a_j is not enabled prior to the execution of a_i, but becomes enabled at s_i and remains enabled until executed. (A self-loop transition which disables and re-enables another transition, such as in Petri nets, is considered causal to the affected transition). If a_i is causal to a_j we naturally require that it occur earlier: $t_i \leq t_j$. (1)

Next, we consider transitions which are independent, in the same sense used previously with partial order reduction. If transition (t, a) is enabled in state s, and $s \stackrel{t,a}{\to} s'$, we denote the successor state s' with $\mathit{succ}_{t,a}(s)$.

Definition 18 *Two* timed *transitions* (t, a) *and* (t', a') *are independent iff for any timed state* s *such that* $(t, a), (t', a') \in \mathit{enabled}(s)$ *the following relations hold:* $(t, a) \in \mathit{enabled}(\mathit{succ}_{t',a'}(s))$, $(t', a') \in \mathit{enabled}(\mathit{succ}_{t,a}(s))$ *and* $\mathit{succ}_{t',a'}(\mathit{succ}_{t,a}(s)) = \mathit{succ}_{t,a}(\mathit{succ}_{t',a'}(s))$. *Two untimed transitions* a *and* b *are independent if the timed transitions* (t, a) *and* (t', a') *are independent for any* $t, t' \in \mathbb{R}^+$, *and are dependent otherwise.*

The goal of our relaxed semantics is to ensure that each execution trace is stuttering equivalent to a trace of the original model. Consider the timed transitions $\stackrel{t,a}{\to}$ and $\stackrel{t',a'}{\to}$, with $t \leq t'$. It is clear that the interleaving which explores $\stackrel{t',a'}{\to}$ followed by $\stackrel{t,a}{\to}$ is equivalent with the original one if a and a' are independent and at least one of the two transitions is invisible.

To characterize the opposite situation, we define $\mathit{conflict}(a) = \{b \in T \mid a \text{ and } b \text{ are dependent or } a \text{ and } b \text{ are visible}\}$. Thus, $\mathit{conflict}(a)$ is the set of all transitions that are dependent on a, to which the set of visible transitions is added, if a itself is visible. If a_i and a_j in trace σ are in conflict, our second requirement is that they be explored in the order of their execution timepoints: $i < j \Rightarrow t_i \leq t_j$. (2)

The ordering conditions (1) and (2) are the less restrictive version of the strict time ordering enforced on $\mathcal{L}_s(Q)$. We next examine a counterpart for the restriction on the next transition firing time.

For an execution trace σ, denote by σ_i the prefix containing the first i transitions: $\sigma_i = s_0 \stackrel{t_1,a_1}{\to} s_1 \ldots \stackrel{t_i,a_i}{\to} s_i$. Denote by $\mathit{enabled}^*(\sigma_i)$ the set of finite or infinite transition sequences $\rho = a_{i+1}a_{i+2}\ldots$ such that for some $t_{i+1}, t_{i+2}, \cdots$ the trace $\sigma' = s_0 \stackrel{t_1,a_1}{\to} s_1 \ldots \stackrel{t_i,a_i}{\to} s_i \stackrel{t_{i+1},a_{i+1}}{\to} s_{i+1} \ldots$ satisfies conditions (1) and (2). Then, let $\mathit{fire}_{\max}(a_k, \sigma_i\rho)$ be the upper bound on the firing time t_k of transition a_k over all such execution traces σ'. We also use the shorthand $a_k \in \rho$ to denote that transition a_k is part of the sequence ρ.

Our final condition requires a transition a_{i+1} to fire before the last enabling time of any conflicting transition that could occur on a continuation of the trace prefix σ_i. That is, $t_{i+1} \prec \mathit{fire}_{\max}(b, \sigma_i\rho)$ for all $b \in \mathit{conflict}(a_{i+1})$ and $b \in \rho \in \mathit{enabled}^*(\sigma_i)$. This ensures that condition (2) is feasible: if the firing time of a_{i+1} were greater than the maximum firing time of transition $b \in \mathit{conflict}(a_{i+1})$, then b could not be explored subsequently while observing $t_{a_{i+1}} \leq t_b$, required by (2).

5.3.2 Traces with relaxed timing

We are now ready to define our semantics in which not all timed transitions have to be executed in the order of their timestamps.

Definition 19 *A relaxed timing semantics for a timed structure Q is given by a family $\mathcal{L}_{\mathrm{r}}(Q)$ of traces over the state space S_{t}, starting at an initial state in S_{t}^0, where each execution trace $\sigma = s_0 \stackrel{t_1,a_1}{\rightarrow} s_1 \stackrel{t_2,a_2}{\rightarrow} s_2 \ldots \stackrel{t_n,a_n}{\rightarrow} s_n \ldots$ satisfies the following conditions for all $i, j \geq 1$:*

(1) a_i *causal to* $a_j \Rightarrow t_i \leq t_j$

(2) $a_j \in \mathit{conflict}(a_i) \wedge i < j \Rightarrow t_i \leq t_j$

(3) $t_{i+1} \prec \mathit{fire}_{\max}(b, \sigma_i\rho)$ *for all* $b \in \mathit{conflict}(a_{i+1})$, $b \in \rho \in \mathit{enabled}^*(\sigma_i)$ *together with the following fairness constraint:*

(**F**) $a \in \mathit{enabled}(s_i) \wedge \mathit{fire}_{\max}(a, \sigma_i) < \infty \Rightarrow \exists k \geq i \,.\, (a \notin \mathit{enabled}(s_k) \vee a = a_k)$

The first three conditions have been discussed in turn. The fairness condition **F** prohibits an indefinite postponement of a transition a which has a finite upper firing bound.

With this definition, we can now prove:

Theorem 6 *The set of relaxed traces $\mathcal{L}_{\mathrm{r}}(Q)$ is a superset of the set of standard traces $\mathcal{L}_{\mathrm{s}}(Q)$. Moreover, each relaxed trace is stuttering equivalent to some standard trace.*

Proof: It is clear that all traces of $\mathcal{L}_{\mathrm{s}}(Q)$ are also traces of $\mathcal{L}_{\mathrm{r}}(Q)$. Indeed, in $\mathcal{L}_{\mathrm{s}}(Q)$ a timed transition has to be firable with respect to *all* transitions enabled at that state, and the ordering condition between timepoints holds between *all* pairs of transitions. The fairness condition is ensured in $\mathcal{L}_{\mathrm{s}}(Q)$

by the non-Zeno assumption: time eventually exceeds any bound, and thus a perpetually enabled transition with a finite firing bound is forced to execute when this bound is reached.

Let us consider a trace $\sigma \in \mathcal{L}_\mathrm{r}(Q)$ and construct a stuttering-equivalent trace $\sigma' \in \mathcal{L}_\mathrm{s}(Q)$. We prove by induction over $k \in \mathbb{N}$ that we can successively construct the execution traces $\sigma^0, \sigma^1, \ldots, \sigma^k \ldots \in \mathcal{L}_\mathrm{r}(Q)$ from σ by permuting transitions, such that $\sigma^k \sim_\mathrm{st} \sigma$, and the first k transitions from σ^k can be executed in the standard semantics. Specifically, σ^k starts with the first k transitions of σ in order of their timepoints, with ties broken in favor of the transition explored earlier. For the base case $k = 0$ we trivially take $\sigma^0 = \sigma$, since the initial states are the same in both trace families.

For the induction step, assume the property is true for some $k \geq 0$. Let (t_j, a_j) be the transition in σ^k with the next smallest timepoint after the transitions $a_1, a_2, \cdots, a_k$ of σ^k. If $j = k+1$, we trivially take $\sigma^{k+1} = \sigma^k$. Otherwise, for $k < i < j$, condition (1) guarantees that a_i is not causal for a_j, otherwise $t_i \leq t_j$ and we would have chosen a_i instead of a_j. Likewise, condition (2) ensures that a_i is not in conflict with a_j, since otherwise again $t_i \leq t_j$. Consequently, a_i and a_j are independent for $k < i < j$ and thus a_j can be successively commuted with $a_{j-1}, \ldots, a_{k+1}$, resulting in a new execution sequence σ^{k+1}. Furthermore, since a_j and a_i are not in conflict, either a_j is invisible, or all a_i with $k < i < j$ are. In either case, $\sigma^{k+1} \sim_\mathrm{st} \sigma^k \sim_\mathrm{st} \sigma$.

We still have to prove that $\sigma^{k+1} \in \mathcal{L}_\mathrm{r}(Q)$. It suffices to show that commuting adjacent non-conflicting transitions into time order preserves conditions (1) through (3). This is clear for conditions (1) and (2), since the transitions which are commuted now occur in increasing time order. For condition (3), we examine the case where the fragment $s \stackrel{t_2,a_2}{\longrightarrow} s_1 \stackrel{t_1,a_1}{\longrightarrow} s'$ of σ_k, with $t_1 < t_2$, is permuted to $s \stackrel{t_1,a_1}{\longrightarrow} s_2 \stackrel{t_2,a_2}{\longrightarrow} s'$ in σ_{k+1}. We need to show condition (3) at states s and s_2, where the explored transitions differ in σ_k and σ_{k+1} (Figure 5.1).

For state s, consider a transition b in conflict with a_1, such that a transition sequence ρb can be executed at s. Moreover, choose ρ to be minimal, in the sense that each transition in ρ is necessary to cause b. If a_2 is independent of all transitions in the sequence ρb, this sequence remains enabled at s_1 after executing a_2, and $t_1 \prec \mathit{fire}_{\max}(b, s_1\rho b)$, since condition (3) holds at s_1 in σ_k. Otherwise, a_2 is in conflict with some transition c in the sequence ρb, and thus $t_2 \prec \mathit{fire}_{\max}(c, s\rho b)$, by condition (3) at s in σ_k. Since c and b are connected by causality, we have $t_c \leq t_b$ and thus $\mathit{fire}_{\max}(c, s\rho b) \leq \mathit{fire}_{\max}(b, s\rho b)$. Since $t_1 \leq t_2$, we obtain by chaining the inequalities that $t_1 \prec \mathit{fire}_{\max}(b, s\rho b)$.

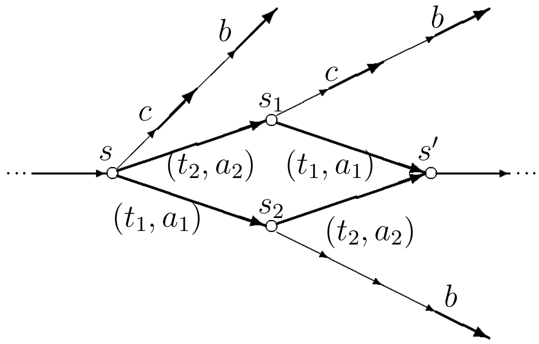

Figure 5.1: Commuting non-conflicting transitions preserves condition (3)

Thus, condition (3) holds at s.

For state s_2, if the transition sequence ρb (where b conflicts with a_2) is executable from s_2, then the sequence $a_1 \rho b$ is executable from s. Applying condition (3) at state s in σ_k we obtain that $t_2 \prec \mathit{fire}_{\max}(b, s a_1 \rho) = \mathit{fire}_{\max}(b, s_2 \rho)$, which is precisely the condition needed at state s_2.

It remains to show that t_j is a legal firing time in the standard semantics. Consider a transition b enabled in s_k, after the first k transitions of σ^{k+1}. If b has an infinite firing bound, we have nothing to prove. Otherwise, if the upper bound on the firing time of b is finite, the fairness condition **F** ensures that b either fires or is disabled at some point. In the first case, if t_b is the firing time of b, we have $t_j \leq t_b$, otherwise, b would have been chosen instead of a_j in the induction step. In the second case, b must be disabled by some transition a_l, thus $t_l \prec \mathit{fire}_{\max}(b, s_{l-1})$, and again $t_j \leq t_l$ because of the choice of a_j. In both cases, t_j does not exceed the maximum firing time of b and thus satisfies the standard semantics. $\Box$

In the relaxed timing semantics defined above, it is possible to fire a transition a_{i+1} from a state s_i even though the minimum firing time of a_{i+1} exceeds the maximum firing time of some other transition b enabled at s_i. If b does not conflict with a_{i+1}, this does not violate condition (3). However, a_{i+1} is not firable from s_i in the standard semantics, since b has to be fired first. This means that the number of untimed transitions which can be fired from a given state in the relaxed semantics can be larger than the number of transitions firable in the standard semantics. Thus, a state search algorithm based on timed regions, which makes one exploration step for each untimed transition from T enabled at a state, may explore more transitions in the relaxed semantics than in the standard semantics.

To ensure that the partial order reduction procedure does not operate on a larger state graph than initially, we can restrict the enabledness condition in the relaxed semantics. Namely, a transition (t_{i+1}, a_{i+1}) is firable after trace σ_i only if a_{i+1} can be fired earlier than the maximum firing time of all enabled transitions, i.e., $\exists t' \in \mathbb{R}^+$ such that $(t', a_{i+1}) \in \mathit{enabled}(\sigma_i)$ and $t' \prec \mathit{fire}_{\max}(b, \sigma_i)$ for all $b \in \mathit{enabled}(\sigma_i)$. In contrast to the standard semantics, this condition does not restrict the maximum firing time of a_{i+1}, it merely requires that a_{i+1} be firable before all other enabled transitions. With this modification, a transition from T is firable in the relaxed semantics iff it is firable in the standard semantics, with no penalty in state space increase.

5.3.3 Enforcing timing conditions

The family $\mathcal{L}_{\mathrm{r}}(Q)$ of traces with relaxed timing is characterized indirectly by a set of conditions. A state-space exploration needs an explicit definition of the transitions that can be explored at any given state. Of the given conditions (1) through (3), the third is difficult to ensure directly, since it is restricts the firing time with respect to all possible future conflicting transitions. To obtain a condition which can be enforced in practice, we draw on the approach of [YS97], which in turn is based on the stubborn set technique of [Val90].

Let b be a transition which is not enabled in the timed state s. A set of transitions is *necessary* for b at s (denoted $\mathit{necessary}(b, s)$) if b cannot be executed on any trace from s without executing some transition in $\mathit{necessary}(b, s)$ first. That is, for any sequence of transitions $a_1, a_2, \cdots, a_k$ starting at s with $a_k = b$ there exists $i < k$ such that $a_i \in \mathit{necessary}(b, s)$. Let $\mathit{necessary}^*(b, s)$ be a set of transitions which contains b and is transitively closed under necessity, i.e., for any $c \in \mathit{necessary}^*(b, s)$ disabled in s, there exists a subset of transitions $\mathit{necessary}(c, s) \subseteq \mathit{necessary}^*(b, s)$ which is necessary for c at s.

Let a be a transition in $\mathit{enabled}(s)$. A set of transitions in $\mathit{enabled}(s)$ is a *dependency set* for transition a at state s (denoted $\mathit{dependency}(a, s)$) if for any transition $b \in \mathit{conflict}(a)$ there exists a set $\mathit{necessary}^*(b, s)$, such that all its transitions that are enabled at s belong to $\mathit{dependency}(a, s)$. Thus, no transition in conflict with a can be enabled starting from s without first executing a transition from $\mathit{dependency}(a, s)$. For both necessary and dependency sets, multiple choices may be possible. In the following, these notations always denote a specific choice of such a set.

The computation of necessary sets depends on the chosen description model. For Petri nets, one can choose the input transitions of an unmarked input place of the disabled transition [YS97]. For communicating processes, a necessary set for a locally enabled communication transition consists of all transitions that precede the corresponding communication points in other processes. For a system containing data variables, a a transition disabled by a false guard has as necessary set all transitions which modify that guard. This can be refined by analyzing the effects of specific variables [Val90].

Let us discuss the enforcement of condition (3) using these notions. We know that in order for any transition $b \in \mathit{conflict}(a_i)$ to fire in the future, an enabled transition $a_j \in \mathit{necessary}^*(b, s_{i-1})$ (and thus in $\mathit{dependency}(a_i, s_{i-1})$) must fire first. Assume that we are requiring $t_i \leq t_j$ for all $j \geq i$ such that $a_j \in \mathit{dependency}(a_i, s_{i-1})$ (and a_j is continually enabled in s_{i-1} through s_{j-1}). Since a_j is necessary for b, a_j is the start of a sequence of causal transitions ρ leading to b, and thus $t_i \leq t_j \leq t_b$. Thus, $t_i \prec \mathit{fire}_{\max}(b, \sigma_{i-1}\rho)$. Consequently, conditions (2) and (3) can be replaced with:

$$t_i \leq t_j \text{ for } i < j \text{ and } a_j \in \mathit{dependency}(a_i, s_{i-1})$$

This analysis can be refined in two ways. First, one can consider transition firing times in the definition of necessary and dependency sets. A transition b which conflicts with an enabled transition a need not affect the firing conditions of a if b cannot fire before the maximal firing time of a.

As a second refinement, the condition $t_i \leq t_j$ for $a_j \in \mathit{dependency}(a_i, s_{i-1})$ can be made less restrictive if a relation between the firing time of a transition $b \in \mathit{conflict}(a_i)$ and the firing time of a transition $a_j \in \mathit{necessary}(b, s_{i-1})$ can be computed. If this relation is of the form $t_b = f(s_{i-1}, t_j)$, for some function f, then we can require $t_i \leq f(s_{i-1}, t_j)$ for $a_j \in \mathit{dependency}(a_i, s_{i-1})$, which replaces conditions (2) and (3).

This second refinement can lead to a reduced branching in the state space. For example, consider a system in which transitions a and b are enabled at the current state, b is in the dependency set of a because it can cause the execution of $d \in \mathit{conflict}(a)$ with $t_d \geq t_b + 2$, and a is in the dependency set of b because it can cause transition $c \in \mathit{conflict}(b)$ with $t_c \geq t_a + 1$. However, if $-1 \leq t_a - t_b \leq 2$ we obtain $t_b \leq t_a + 1 \leq t_c$ and $t_a \leq t_b + 2 \leq t_d$, so under these conditions both a and b can fire, without affecting each other. With our refinement, the timed region $-1 \leq t_a - t_b \leq 2$ is obtained in the relaxed timing semantics regardless of the exploration order between a and b, and can be explored as a whole. With the first definition of dependence sets, taken from [YS97], the interleavings $t_a \leq t_b$ and $t_b \leq t_a$ have to be considered, and thus two regions, $-1 \leq t_a - t_b \leq 0$ and $0 \leq t_a - t_b \leq 2$, are obtained and further explored separately.

5.3.4 Exploration based on timed regions

We have seen that an execution trace with relaxed timing has to satisfy conditions of the form $t_i \leq t_j$ or $t_i \leq f(t_j)$ for $i < j$. These inequalities are enforced either when a_j is caused by a_i or when a_j is in the dependency set or conflict set of a_i. To this effect, the transition execution times t_i have to be part of the timed state, or have to be temporarily added to the timed state as auxiliary variables, for as long as it is needed to enforce such inequalities.

In time Petri nets or TEL structures, the firing time of a transition or event appears explicitly as part of the timed state and the transition relation: a transition fires within specified time bounds of the transition that enabled it. In timed automata, the current time appears as auxiliary variable in the form of the zero clock. The advancing of time after each transition serves to enforce the order among sequentially executed transitions.

A practical state space exploration algorithm does not explore an infinite, uncountable number of timed traces, but operates instead on sets of timed states called timed regions. An exploration step for a given transition a consists in computing the successor region containing all timed states reached by executing that transition from the states of the current timed region r:

$$succ_a(r) = \{s' \in S_{\mathrm{t}} \mid \exists s \in r, t \in \mathbb{R}^+ \mid s \stackrel{t,a}{\rightarrow} s'\}$$

We also write $r \stackrel{a}{\rightarrow} r'$ if $r' = succ_a(r)$. Then, a sequence of timed regions $r_0 \stackrel{a_1}{\rightarrow} r_1 \ldots \stackrel{a_i}{\rightarrow} r_i \ldots$ accounts for all timed traces $\sigma = s_0 \stackrel{t_1,a_1}{\longrightarrow} s_1 \ldots \stackrel{t_i,a_i}{\longrightarrow} s_i \ldots$, where r_0 is the region containing all initial states s_0. Typically, as we have seen for timed automata, time Petri nets and TEL structures, the description of the timed system contains a set of time variables, and timed regions are represented using difference inequalities on those variables.

To incorporate conditions of the form $t_i \leq t_j$ or $t_i \leq f(t_j)$ into the region successor operation, two basic possibilities exist. A first solution retains the firing time t_i of a transition a_i as part of a timed state (and thus, timed region), as long as there are enabled transitions a_j for which a relative constraint between t_i and t_j may need to be enforced. Once no such enabled transitions remain, t_i is removed from the representation of the timed region by existential quantification.

A second solution introduces, upon firing a_i, time variables for all potential future transitions a_j whose firing time t_j is related to t_i. Then, t_i is quantified out, being no longer needed, and likewise the time variables for the transitions which become disabled as a result of firing a_i. Thus, the current timed region contains a time variable for each enabled transition. This is the solution adopted in [YS97, BM98].

The relative tradeoffs of the two approaches depend on the analyzed model. If the branching factor at each state is high, tracking all enabled transitions may lead to a large number of unneeded time variables, since not all enabled transitions are executed. If only past transition times are maintained, no unnecessary variables are introduced. However, the time of a past transition may have to be retained long after its exploration, as long as there are unexplored transitions that need to be related to it.

5.4 Partial Order Reduction

The exploration step *succ* for exploring a transition defines a state-transition graph (*region automaton*) $\mathcal{R}(Q)$ whose states are timed regions. We restrict ourselves to the case when the number of timed regions is finite. The particular models analyzed so far (timed automata, time Petri nets, TEL structures) admit a finite quotient, since their timing is described by elementary difference constraints with integer constants.

Partial order reduction can be applied to the region automaton by finding an ample set of transitions which is sufficient for exploration at each state. We construct the ample sets based on the *dependency* sets discussed in Section 5.3. This notion can be naturally extended to regions, by defining $b \in \mathit{dependency}(a, r)$ iff $\exists s \in r \,.\, b \in \mathit{dependency}(a, s)$. For **C0**, **C2** and **C3**, we use the standard formulation of the ample set conditions. For **C1**, we require of any region $r \in R$ that:

C1' $\quad a \in \mathit{ample}(r) \Rightarrow \mathit{dependency}(a, r) \subseteq \mathit{ample}(r)$

In other words, the ample set of a region is closed with respect to the *dependency* relation. Thus, an ample set can be computed by choosing an enabled transition and successively adding any transitions in the dependency set of an ample transition to the ample set, until a fixpoint is reached.

We can easily show that condition **C1'** subsumes the faithful decomposition condition **C1** required for ample sets.

Proposition 10 *If for every timed region r, the ample set $\mathit{ample}(r)$ satisfies condition* **C1'***, then no transition which is dependent on a transition from $\mathit{ample}(r)$ can be executed before a transition from $\mathit{ample}(r)$.*

Proof: The result is a consequence of the correspondence between the region automaton and the underlying infinite family of timed traces $\mathcal{L}_{\mathrm{r}}(Q)$.

A transition b dependent on $a \in ample(r)$ can be executed in $\mathcal{R}(Q)$ only if a corresponding timed transition (t_b, b) can be executed in $\mathcal{L}_{\mathrm{r}}(Q)$. However, by the definition of dependency sets, some enabled transition from $necessary^*(b, r)$ has to be executed before b, and this transition belongs to $dependency(a, r) \subseteq ample(r)$. This completes the proof. $\Box$

5.5 Discussion

Of the existing approaches to partial order reduction for timed systems, our formalism draws most from the work of Yoneda and Schlingloff [YS97] on time Petri nets. We present the main differences of our approach below.

First, the formalism presented here is significantly more general. The notion of timed state is generic, and the timed transition relation between two states can be more complex than a time separation with lower and upper bounds, as in time Petri nets. In fact, the only conceptual restriction for our model of timed structures is that the resulting region automaton be finite. This is true if the timed transition relation is based on atomic difference constraints between time variables, such as in timed automata. However, the generality does not introduce unnecessary complexity. In fact, for time Petri nets our approach results in an algorithm similar to that of [YS97], with potential improvements discussed below.

Another difference consists in the approach taken to design and prove the algorithm. In [YS97], a region-based state space exploration algorithm without partial order reduction is given first. Then, the time ordering of transitions in the region-based model is relaxed, as a prerequisite to partial order reduction. As a consequence, the proof is quite complex, because the state space obtained using partial orders is not a subset of the original state space. Furthermore, the proof makes extensive reference to the particular representations of transition constraints and timed regions.

Our approach has been to relax time ordering on the family of timed traces underlying the system model. As a result, ensuring stuttering equivalence by enforcing constraints on transitions from a dependency set leads naturally to the selection of a reduced set of transitions for exploration. Consequently, the main burden of the proof falls onto proving stuttering equivalence for timed traces. The application of partial order reduction to the resulting region-based model is straightforward.

The algorithm of [YS97] requires an ample transition to fire before *all* other transitions from the ample set. The algorithm given here is less restrictive, and requires only the firing before all enabled transitions in the dependency set, which is a subset of the ample set. Since an ample transition is independent from the transitions in the difference of these two sets, fewer timed regions can be generated if a time ordering between these transitions does not need to be enforced. Furthermore, if the firing time of a future conflicting transition can be determined from the firing time of a currently enabled transition, the branching in the state space can be further reduced, as shown in the end of section 5.3.3.

The correspondence to timed event/level structures is direct and quite similar to time Petri nets. Time variables in this case are the firing times of events. Causality conditions are expressed directly as part of the rules, and the analysis of dependency relations is done in the same way as for time Petri nets. Similarly, an event time is retained as a part of the timed state as long as events caused by it can still be enabled; it can be quantified out subsequently.

In comparison to our local-time approach for timed automata, the main difference lies in the firing semantics of transitions. Using the terminology of [BST99], which defines three types of urgency for transitions, in our model of timed structures transitions are *delayable*. They are required to fire before their enabling interval expires; within this interval they can fire at any given time. In timed automata transitions that are not constrained by a state invariant are *lazy*: it is possible for them not to fire even if enabled throughout their firing interval. Transitions on which the state invariant imposes an upper firing bound are delayable just like in timed structures. (A third type, *eager* or *urgent* transitions which execute immediately when enabled can be handled by performing a special check for such transitions at any state).

Lazy transitions can be incorporated in our framework without significant changes. We have chosen to discuss delayable transitions only in order not to complicate the presentation. The only change refers to the requirement that a transition execute at a prior time compared to all transitions in its dependency set. This ensures that if a transition from the dependency set is actually executed, the resulting trace sequence is still consistent with the standard semantics, without conflicting with previously explored transitions. However, since lazy transitions are not forced to execute, they are not subject to this condition. Thus, a transition only needs to fire before all *delayable* transitions from its dependency set.

For timed automata, causality is modeled by advancing the local time in each automaton after a transition. Thus, the new local time represents a possible legal firing time for a new transition, and its advancement ensures that the new transition takes place at a later time than the previous transition in the same automaton. At the same time, our general approach points out an alternative representation of local time. Instead of maintaining one reference time for each automaton (representing a potential time for a future transition), it is possible to maintain the time of the last transition in each automaton instead. With this approach, the number of auxiliary variables needed to represent a timed zone can be less than the number of automata at some states, because several automata can share a synchronization transition as last executed transition.

Concluding, we see as the main benefit of our general approach the fact that it identifies the conditions and the potential for partial order reduction at the elementary level of timed traces, to which a large variety of timed models can be reduced. The fundamental idea is to distinguish between transition causality and serialization due to timing, and to define a semantics which eliminates unnecessary serialization and branching in the state space. The reduction procedure itself is given in terms of several generic notions, such as the enforcement of transition ordering, the computation of dependency sets and the representation of timed regions. To obtain a practical model checking algorithm, the characteristics of the given time model can be taken into account to particularize this method into an efficient implementation.

Chapter 6

Experimental Results

6.1 Implementation

We have evaluated the performance gains that can be obtained by partial order reduction for systems modeled as networks of timed automata. We have concentrated on this particular model both since it is the most complex and expressive among those studied, and because no practical results concerning partial order reduction for timed automata have been reported so far.

In order to isolate the effects of partial order reduction, we have implemented both a standard state space exploration algorithm using timed zones and an algorithm that uses the local-time model described in Chapter 3. In both cases, we represent clock constraints using difference bound matrices, implemented simply as two-dimensional arrays.

To facilitate the comparison with other tools and the analysis of benchmarks commonly used in the literature, the tool that we have implemented operates on timed automata models which are described in the input language of the UPPAAL verifier [LPW95]. The model adopted by UPPAAL extends the definition of networks of timed automata (as presented in Chapter 3) by allowing the system to be augmented with integer variables. These can occur in transition guards and can be assigned as a result of a transition. This extension is useful in a large number of practical cases, and allows a natural modeling of more complex timed systems. It also introduces additional dependencies between the components and transitions of the system. We show in the following how to extend our results concerning partial order reduction to handle shared variables in the model.

If a variable is shared by several components of the system, the usual cases of read-write and write-write dependencies appear. To maintain the correct semantics, we have to ensure that write accesses to the variable are serialized with respect to both reads and other writes in the order of their occurrence in time. That is, if transitions a_i and a_j in a timed execution trace $(t_1, a_1), (t_2, a_2), \ldots, (t_n, a_n), \ldots$ are in conflict with respect to variable v, then $i < j$ if and only if $t_i < t_j$.

One option for ensuring this property is to introduce an auxiliary time variable t_v for each global variable v in the model. This variable would be set by each transition that accesses (reads or writes) v to the execution timepoint of that transition. All such transitions would then become dependent and would be serialized by ensuring that t_v grows monotonically. However, this approach quickly becomes expensive if the model contains many variables. Moreover, it unnecessarily serializes all transitions that read the variable, even though they have the same effect regardless of their relative order.

Instead, we have chosen the following approach. For each variable, the two sets of processes that can read and, respectively, write that variable are statically determined at the time of building the model. If the variable is local to a single process, nothing need be done. Otherwise, if a transition that accesses variable v is added to an ample set, all enabled transitions in the other automata that access v need to be selected as well. Moreover, when selecting such a transition for exploration, its execution timepoint is serialized with respect to the processes that potentially contain a transition which conflicts with respect to v.

Thus, if we use local reference times for each process, as in the local time model of Chapter 3, then a transition a which reads v will be restricted with the conjunct $\bigwedge_{i \in write(v)} t_i \geq t_a$, and a transition b which writes v is restricted by $\bigwedge_{i \in read(v) \cup write(v)} t_i \geq t_b$, where $read(v)$ and $write(v)$ are the sets of process indices which read and write v, respectively. This ensures that in the other relevant processes, the reference time has already advanced past the execution point of the considered transition, and thus any conflicting transitions explored subsequently are serialized in the correct order. If we use instead variables denoting the last transition in a given process, as in Chapter 5, then for a read transition a we require $\bigwedge_{i \in write(v)} t_a \geq t_{\mathrm{last}_i}$, and for a write transition b we require $\bigwedge_{i \in read(v) \cup write(v)} t_b \geq t_{\mathrm{last}_i}$. Here, the inequalities ensure that the transition occurs at a timepoint which is later than that of the last executed transition in any potentially conflicting process.

6.2 Parameterized Benchmarks

Our first comparison is made on a set of benchmarks which has been used in [BMPY97] to compare continuous-time techniques based on difference bound matrices with discrete-time techniques based on numerical decision diagrams (NDDs). The same examples are used in [BM98] to compare the efficiency of the POSET method for TEL structures. These benchmarks highlight specific extreme-case scenarios which appear in the exploration of timed systems.

Benchmark $\mathcal{A}$ (Figure 6.1) consists of a series of N independent timed automata, A_i each with a single state and one clock C_i. Each of the n states has an invariant $C_i < u_i$ and a self-loop transition with a lower bound $C_i \geq l_i$ which also resets C_i. Thus, the global system has a unique control state, but the set of possible time configurations becomes more and more complex as the system evolves, eventually covering the entire possible space of clock values. In [BMPY97] it is shown that standard DBM techniques cannot handle more than 5 of these automata composed together. Our results, shown in Table 6.1 are consistent with those obtained in [BM98] using POSETs. It can be seen that with the local time model, only relatively few timed states need to be generated before the entire state space is finally covered. Since the example contains only one control state, partial order reduction is not applicable, and the improvements are due entirely to the local time model.

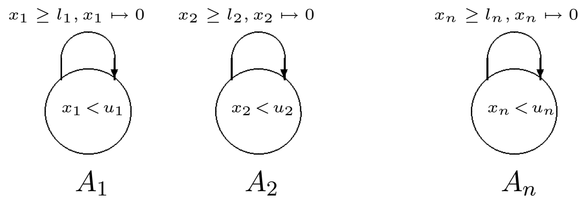

Figure 6.1: Benchmark $\mathcal{A}$

To preserve consistency with the results of [BMPY97], in this example, as well as in the remainder of the benchmarks in this section, the time constants in the model have been generated randomly from the interval [0..7].

A second benchmark $\mathcal{B}$ (Figure 6.2) consists of N two-state automata, between which the automaton switches in a time interval $[l_i, u_i)$. Such an automaton represents a boolean signal for which two successive changes in value are constrained by a lower and an upper time bound. An array of such

N	16	32	48	64	80	96	112	128
states	72	158	229	226	298	382	439	469
time (s)	0	1.4	7.2	15.7	40	84.8	154	252

Table 6.1: Exploration of example $\mathcal{A}$ using a local-time model

automata would be necessary to model the behavior of a circuit under all possible inputs. Again, the results for reachability analysis are similar to those obtained with the POSET method, and significantly better than the standard exploration, which cannot handle more than 4 stages. This model is significantly more complex than the previous one, and the number of timed states increases much faster (the number of control states is 2^N).

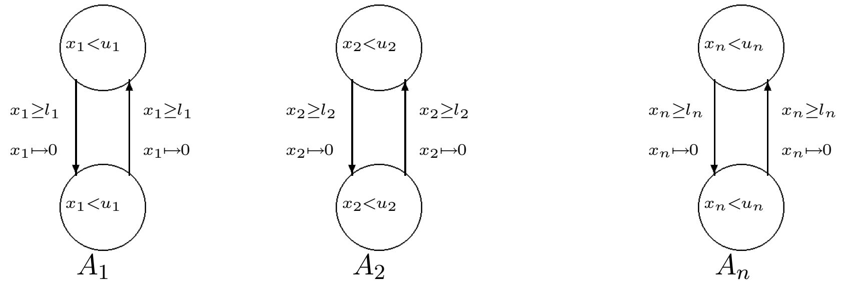

Figure 6.2: Benchmark $\mathcal{B}$

Due to the independence of its transitions, this model is the ideal candidate for partial order reduction. Table 6.2 presents the comparative results for state space search with and without reduction (using the local time model in both cases). The reduction results are given for the best case with no visible transitions (this is the case if $\mathcal{B}$ is part of a model being verified either for deadlock detection or with respect to other visible properties). With partial order reduction, the number of states increases linearly rather than exponentially: 80 automata are analyzed in less time and a fraction of the space compared to 13 automata without reduction.

The final example of this section is an asynchronous circuit consisting of N XOR gates with delays, connected in a ring, in which gate i outputs x_i after some bounded delay, and has as inputs the (delayed) values of x_i and x_{i-1}. Each gate can be represented by a 4-state timed automaton, with states encoding the actual and hidden value of the output signal, and a clock that models the delay [MP95].

N	8	9	10	11	12	13
states	1214	3463	9623	18634	36320	71442
time (s)	0	0.5	2	4.85	11.7	27.7
N	8	16	32	48	64	80
states (red.)	75	262	653	1312	1394	2844
time (s)	0	0	0.5	3	6.8	20.8

Table 6.2: Exploration of example $\mathcal{B}$ using a local-time model

The system is strongly coupled: each change in one of the signals potentially cascades to cause changes in all gates in the ring, and the feedback loops create a high complexity of the resulting state space. We present the results of computing all timed states that are reachable from the initial unstable state in which all signals have the value 1. Several variations of the state space search have been employed. In Table 6.3, *sync* denotes a local-time exploration in which only synchronizable states are explored (cf. Chapter 3). Lines marked with *act* denote results obtained using the clock activity reduction of [DY96], eliminating clocks which are no longer used before they are reset. For a gate modeled as a timed automaton, this reduction occurs at the stable states, from which the clock is reset when switching to an excited state that subsequently causes a change in output.

The results show that, even though the number of timed states is exponential in the number of gates for both standard and local-time exploration, the performance using the local-time model degrades more gracefully, with a factor of more than 20 in running time for 6 gates distinguishing the two. Moreover, it is of significant advantage to restrict the exploration to synchronizable states. Not surprisingly, clock activity reduction improves efficiency for the local-time model as well, and individually it performs even better than the restriction to synchronizable states.

6.3 Case Studies of Timed Systems

We have evaluated the behavior of our local-time state space exploration algorithm in practice by analyzing several models of timed systems that have been presented as case studies in the literature. All of the systems presented here have been previously modeled and analyzed using the UPPAAL verifier.

The first model is a description of the Philips audio control protocol, de-

	4 gates		5 gates		6 gates	
Method	time	states	time	states	time	states
standard	0	1104	0.9s	10992	795s	469706
local	0	1384	2s	12778	>10min	>400k
local + sync	0	1047	0.7s	6901	38s	95087
local + act	0	444	1s	5285	29s	52190
local + act + sync	0	444	1s	5133	27s	49482

Table 6.3: Exploration of a ring of XOR gates

veloped in order to exchange control information using Manchester encoding between audio equipment components. The protocol is modeled using four timed automata, communicating via 12 channels and using four integer variables and two clocks. The input automaton generates valid bit sequences for the sender automaton, which encodes them, determining the necessary delays for the encoding voltage signal. The receiver automaton decodes the bit stream from the sender by measuring the delay between two subsequent signals. Finally the output acknowledgement automaton checks the bits decoded by the receiver. In this model, the components are quite strongly synchronized. After taking variable dependencies into account, there is one single state which has a local transition that can form an ample set by itself. As a consequence, the same results are obtained using the standard and local model, with or without partial order reduction.

States	standard	loc + syn	loc + syn + po
control	145	145	145
timed	151	151	151

Table 6.4: Philips Audio Control Protocol (without bus collision)

The box sorter is a simpler example describing a system, consisting of four timed automata, representing a controller, the behavior of a box travelling through the system, as well as a piston and an observer that interact with the box.

In this example, the network of automata is also quite strongly coupled, with a high density of synchronization transitions, and few possible interleavings, as can be observed directly from the description, or simulating the systems using UPPAAL. Partial order reduction together with the local time

States	standard	local	loc + po	loc + syn	loc + syn + po
control	61	89	66	61	56
timed	558	277	233	226	216

Table 6.5: Box Sorter

model result in a reduction of the state space with a factor of about 2.5, with the local-time model accounting for the greater part. Using the unrestricted local-time model, without regard for synchronizable states, leads to a somewhat higher number of control states (some of which are not reachable in the standard semantics). At the same time, the total number of timed states decreases. Restricting the model to synchronizable states is beneficial, a characteristic which we have observed for all our examples.

The next example is a model of a manufacturing plant. It represents the timing and synchronization mechanism of two robots that transport boxes between a service station and a belt, in either direction. Analyzed with the standard reachability algorithm, the system turns out to be quite complex, resulting in more than 80,000 timed states, even with just five processes and five clocks. The reason for this large state space resides in the time constants that appear in the model: several guards with large integer bounds (> 100) result in a significant number of possible time assignments. The local-time model is especially efficient here, resulting in a 66-fold reduction in the number of timed states, with a small additional gain for partial order reduction.

An implementation variant of the search algorithm concerns testing for inclusion between timed zones. The results presented so far test only whether the newly reached zone is included in one which has been already explored. Conversely, replacing a previously explored zone can be replaced if it is included in the current one, after which the search is continued as usual. This solution may save space, potentially at the expense of time in additional checks. For this example, the space savings due to reduction are increased, while using comparable time.

Finally, we have run our tool on a model of the bounded retransmission protocol, a version of the alternating bit protocol over a lossy communications channel, with a bounded number of retransmissions of any given packet. The protocol is described using a total of seven processes, which model a sender and a receiver (each with its own channels), two lossy communication lines, and an abstraction of the transmitted file. The model contains 5 clocks, 10

Search	States	standard	loc + syn	loc + syn + po
no inclusion	control	211	211	175
	timed	70338	1065	895
with inclusion	control	211	211	173
	timed	63119	926	597

Table 6.6: Manufacturing Plant Model

integer variables and more than a dozen communication channels. Runs have been made with two different sets of model constants, both with and without the double inclusion test. Partial order reduction achieves gains of up to 1/3 even though just two states have ample sets with one local transition.

Variant	States	standard	loc + syn	loc + syn + po
C1 no incl.	control	2477	2513	2038
	timed	25986	22929	17287
C1 with incl.	control	2477	2508	2036
	timed	18612	15581	12315
C2 no incl.	control	6577	6590	5982
	timed	120738	122008	112789
C2 with incl.	control	6552	6574	5966
	timed	70897	65469	60830

Table 6.7: Bounded Retransmission Protocol

In summary, our results for these models, whose characteristics are representative of typical systems targeted for verification, show that the local-time model, when restricted to synchronizable states, always leads to a clear improvement in the size of the reachable state space. In addition, further savings can be obtained by selecting a reduced set of transitions for exploration and applying partial order reduction techniques from the untimed domain. As expected, the gains obtained during the latter step are highly dependent on the structure of the model: small improvements (10% - 20%) are obtained for models which are tightly synchronized and have few internal transitions, but the gains can be orders of magnitude if there are a significant number of mutually independent transitions.

Chapter 7

Conclusions

In this dissertation we have presented solutions for the application of partial order methods to the verification of timed systems. We have given a partial order reduction algorithm for networks of timed automata which preserves formulas in a timed extension of linear temporal logic. The algorithm is based on a modified local-time semantics, which allows individual automata to execute independently except for synchronization transitions. Timed automata constitute the most expressive timing formalism for which partial order reduction has been investigated so far.

More generally, we have investigated the issues that underlie the application of partial order reduction in a continuous-time model. For a general model whose semantics is defined in terms of timed traces, we show how to separate causal dependence of transitions from time ordering due to concurrency and how to obtain general conditions for the application of partial order reduction. As particular instances of this framework we obtain improved algorithms for timed event/level structures and time Petri nets, as well as the algorithm for timed automata based on the local-time model.

We have evaluated the performance of our partial order reduction approach by building a tool which implements the reduction algorithm for networks of timed automata and analyzing several examples. The resulting reduction in state space stems from two sources: the local-time model reduces the number of generated time regions, while the partial order techniques applied from the domain of untimed systems reduce the explored control state space.

Future Work

The research issue that seems most immediately appealing is the combination of partial order reduction and symbolic model checking in the context of timed systems. Symbolic approaches for the representation of the large number of time zones resulting from state space exploration have long been an issue of special interest in real-time verification. However, due to the different nature of the operations performed on control states and time regions, symbolic representations that are applicable to both components have been difficult to find.

Recently, two data structures inspired by BDDs, clock difference diagrams [BLP+99] and difference decision diagrams [MLAH99] have been proposed. The latter data structure provides a unified framework for handling control and timing information, and algorithms to perform conjunction, substitution and existential quantification, the elementary operations of the state space exploration algorithm for timed automata. Moreover, first reported results, although so far only for systems with a very regular structure, have shown that fully symbolic model checking can significantly outperform the traditional algorithms for timed automata.

The state-space exploration algorithm based on the local-time model can be implemented without difficulty using DDDs, since it is based on the same basic operations as the standard zone-based exploration. Also, it is in this context that static partial order reduction can be used to its best advantage, given its independence of the underlying exploration algorithm. Instead of encoding the exploration of all outgoing transitions from a given state, the symbolic representation of the transitions relation will merely contain those transitions which have been selected for execution by the reduction algorithm.

It is well known that the size of a symbolic representation does not bear a direct relation to the number of states represented. Therefore, the combination of partial order reduction and symbolic model checking is not automatically a more efficient technique. However, the main goal of a symbolic representation is to efficiently store and process a set of individual states, whereas the local-time model already coalesces individual time regions into coarser ones. Thus, it can be expected that the local time model would already carry out in part the task of the symbolic algorithm, and furthermore that the selection of a reduced number of transitions may decrease the complexity of a symbolic exploration step.

A second direction of research concerns the applicability of partial order reduction to more expressive models. The present framework for the use of partial order reduction for timed systems depends essentially on the fact that time advances at the same rate in all components of the model. A next step would be to investigate this technique for systems with multi-rate clocks and more generally for hybrid systems, which combine continuous and discrete evolution.

Yet another question concerns the applicability of partial order reduction jointly with other state space reduction techniques. In particular, we have seen partial order reduction applied to two different quotient models: the zone automaton and the region graph automaton. But other models that can be used for efficient verification exist, in particular the quotient with respect to a time-abstracting bisimulation, which can be much smaller. An interesting question is whether partial order reduction can be applied together with this minimization, and in particular with on-the-fly techniques.

Ultimately, the goal of this, as of any other verification technique, is the successful application to practical designs. Even though many different formalisms are used for the modeling of timed systems, we have shown that a quite general principle for the application of partial order reduction can be found. Algorithms for a partial order state space exploration can be extracted based on the particular characteristics of the chosen model, using the same representation as a search without reduction or a slightly modified one. Our results for timed automata, together with prior results for other timed models show that partial order reduction is a feature which can result in significant gains when implemented in a verification system.

Bibliography

[ABH+97] R. Alur, R. K. Brayton, T. A. Henzinger, S. Quadeer, and S. K. Rajamani. Partial-order reduction in symbolic state space exploration. In Grumberg [Gru97], pages 340–351.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In LICS90 [LIC90], pages 414–425.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In M. S. Paterson, editor, *Automata, Languages, and Programming. 17^{th} International Colloquium Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335, Coventry, UK, July 1990. Springer-Verlag.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[AH91] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice. REX Workshop Proceedings*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106, Mook, Netherlands, June 1991. Springer-Verlag.

[AK95] Rajeev Alur and Robert P. Kurshan. Timing analysis in COSPAN. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III. Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 220–231, New Brunswick, NJ, USA, October 1995. Springer-Verlag.

[AMP98] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In Sangiorgi and de Simone [SdS98], pages 470–484.

[Bal96] Felice Balarin. Approximate reachability analysis of timed automata. In RTSS96 [RTS96], pages 52–61.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In LICS90 [LIC90], pages 428–439.

[BD98] Dragan Bošnački and Dennis Dams. Integrating real time in Spin: a prototype implementation. In Stan Budkowski, Ana Cavalli, and Elie Najm, editors, *Proceedings of FORTE/PSTV '96. IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing and Verification*, pages 423–438, Paris, France, October 1998. Kluwer Academic Publishers.

[Bel99] Wendy A. Belluomini. *Algorithms for Synthesis and Verification of Timed Circuits and Systems.* PhD thesis, University of Utah, 1999.

[BF99] Burkhard Bieber and Hans Fleischhack. Model checking of time petri nets based on partial order semantics. In Baeten and Mauw [BM99], pages 210–225.

[BJLW98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Yi Wang. Partial order reductions for timed systems. In Sangiorgi and de Simone [SdS98], pages 485–500.

[BLP+99] G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification. 11th International Conference, CAV'99. Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353, Trento, Italy, July 1999. Springer-Verlag.

[BM97] Wendy Belluomini and Chris J. Myers. Timed event/level structures. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, December 1997.

[BM98] Wendy Belluomini and Chris J. Myers. Verification of timed systems using POSETs. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification. 10th International Conference, CAV'98. Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 403–415, Vancouver, BC, Canada, June/July 1998. Springer-Verlag.

[BM99] Jos C.M. Baeten and Sjouke Mauw, editors. *CONCUR'99: Concurrency Theory.* 10^{th} *International Conference. Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, Eindhoven, Netherlands, August 1999. Springer-Verlag.

[BMH99] Wendy Belluomini, Chris J. Myers, and H. Peter Hofstee. Verification of delayed-reset domino circuits using ATACS. In *Proceedings. Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, Barcelona, Spain, April 1999. IEEE Computer Society Press.

[BMPY97] Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine. Some progress in the symbolic verification of timed automata. In Grumberg [Gru97], pages 179–190.

[BMT99] Marius Bozga, Oded Maler, and Stavros Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods.* 10^{th} *IFIP WG 10.5 Advanced Research Working Conference, CHARME '99*, volume 1703 of *Lecture Notes in Computer Science*, pages 125–141, Bad Herranalb, Germany, September 1999. Springer-Verlag.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BS97] Bonnie Berger and Peter W. Shor. Tight bounds for the maximum acyclic subgraph problem. *Journal of Algorithms*, 25(1):1–18, October 1997.

[BST99] Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference. International Symposium, COMPOS '97. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129, Bad Malente, Germany, September 1999. Springer-Verlag.

[Cam96] Sérgio Vale Aguiar Campos. *A Quantitative Approach to the Formal Verification of Real-Time Systems*. PhD thesis, Carnegie Mellon University, September 1996.

[CCM+94] S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proceedings. Real-Time Systems Symposium*, pages 266–270, San Juan, Puerto Rico, December 1994. IEEE Computer Society Press.

[CCM97] Sérgio Campos, Edmund M. Clarke, and Marius Minea. Symbolic techniques for formally verifying industrial systems. *Science of Computer Programming*, 29(1–2):79–98, July 1997.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, May 1981.

[CGMP99] E. M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *Software Tools for Technology Transfer*, 3(1), 1999. Springer-Verlag.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CK90] E. M. Clarke and R. P. Kurshan, editors. *Computer Aided Verification. 2nd International Conference, CAV'90. Proceedings*, volume 531 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, June 1990. Springer-Verlag.

[Cou93] Costas Courcoubetis, editor. *Computer Aided Verification. 5th International Conference, CAV'93. Proceedings*, volume 697 of *Lecture Notes in Computer Science*, Elounda, Greece, June 1993. Springer-Verlag.

[CP96] Ching-Tsun Chou and Doron Peled. Formal verification of a partial-order reduction technique for model checking. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems. Second International Workshop, TACAS '96. Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 241–257, Passau, Germany, March 1996. Springer-Verlag.

[DGKK98] Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. Partial-order reduction techniques for real-time model checking. In Jan Friso Groote, Bas Luttik, and Jos van Wamel, editors, *Proceedings of the Third International Workshop on Formal Methods for Industrial Critical Systems*, pages 157–169, Amsterdam, The Netherlands, May 1998.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems. International Workshop. Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Grenoble, France, June 1989. Springer-Verlag.

[Dil94] David L. Dill, editor. *Computer Aided Verification. 6th International Conference, CAV'94. Proceedings*, volume 818 of *Lecture Notes in Computer Science*, Stanford, CA, USA, June 1994.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In RTSS96 [RTS96], pages 73–81.

[EJP97] E. A. Emerson, S. Jha, and D. Peled. Combining partial order and symmetry reduction. In *Tools and Algorithms for the Construction and Analysis of Systems. Third International Workshop, TACAS '97. Proceedings*, volume 1217 of *Lecture Notes* *in Computer Science*, pages 19–34, Enschede, The Netherlands, April 1997. Springer-Verlag.

[ELS93] Peter Eades, Xuemin Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, October 1993.

[Esp94] Javier Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, December 1994.

[GKPP99] Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2):132–152, May 1999.

[God90] Patrice Godefroid. Using partial orders to improve automatic verification methods. In Clarke and Kurshan [CK90], pages 176–185.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Conference Record of the Seventh ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

[GPVW95] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification*, pages 3–18, Warsaw, Poland, June 1995. Chapman & Hall.

[Gru97] Orna Grumberg, editor. *Computer Aided Verification. 9th International Conference, CAV'97. Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, Haifa, Israel, June 1997. Springer-Verlag.

[GW91] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In K. G. Larsen and A. Skou, editors, *Computer Aided Verification.*

3rd International Conference, CAV'91. Proceedings, volume 575 of *Lecture Notes in Computer Science*, pages 332–342, Aalborg, Denmark, July 1991. Springer-Verlag.

[HK90] Zvi Har'El and Robert P. Kurshan. Software for analytical development of communication protocols. *AT&T Technical Journal*, 69(1):45–59, Jan.-Feb. 1990.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In W. Kuich, editor, *Automata, Languages, and Programming. 19^{th} International Colloquium Proceedings*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558, Wien, Austria, July 1992. Springer-Verlag.

[HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 394–406, Santa Cruz, CA, USA, June 1992. IEEE Computer Society Press.

[Hoa95] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1995.

[Hol92] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice–Hall, 1992.

[HP94] Gerard J. Holzmann and Doron Peled. An improvement in formal verification. In D. Hogrefe and S. Leue, editors, *Formal Description Techniques VII. Proceedings of the 7^{th} IFIP WG 6.1 International Conference*, pages 197–211, Bern, Switzerland, October 1994.

[KLM+97] R. P. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün. Verifying hardware in its software context. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 742–749, San Jose, CA, USA, November 1997. IEEE Computer Society Press.

[KLM+98] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün. Static partial order reduction. In Bernhard Steffen, editor, *Tools* *and Algorithms for the Construction and Analysis of Systems, 4^{th} International Conference, TACAS'98. Proceedings*, volume 1384 of *Lecture Notes in Computer Science*, pages 345–357, Lisbon, Portugal, Mar.–Apr. 1998. Springer-Verlag.

[KP88] Shmuel Katz and Doron Peled. An efficient verification method for parallel and distributed programs. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. School/Workshop*, number 354 in Lecture Notes in Computer Science, pages 489–507, Noordwijkerhout, The Netherlands, May 1988. Springer-Verlag.

[Kur94] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.

[Lam83] L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing 83. Proceedings of the IFIP 9^{th} World Computer Congress*, pages 657–668, Paris, France, September 1983. North-Holland.

[LIC90] *Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, Philadelphia, PA, USA, June 1990. IEEE Computer Society Press.

[Lil98] Johan Lilius. Efficient state space search for time Petri nets. In P. Jancar and M. Krétinsky, editors, *Proceedings of MFCS'98 Workshop on Concurrency*, Brno, Czech Republic, August 1998. Elsevier.

[LPW95] Kim G. Larsen, Paul Pettersson, and Yi Wang. Model-checking for real-time systems. In *Fundamentals of Computation Theory. $10^t h$ International Conference, FCT'95. Proceedings*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88, Dresden, Germany, August 1995. Springer-Verlag.

[McM92] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In G. v. Bochmann and D. K. Probst, editors, *Computer Aided*

Verification. Fourth International Workshop, CAV'92. Proceedings, volume 663 of *Lecture Notes in Computer Science*, pages 164–177, Montreal, Canada, June 1992. Springer-Verlag.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[McM95] Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, January 1995.

[MF76] P. Merlin and D.J. Faber. Recoverability of communication protocols. *IEEE Transactions on Communication*, COM-24(9):381–404, 1976.

[Min99] Marius Minea. Partial order reduction for model checking of timed automata. In Baeten and Mauw [BM99], pages 431–446.

[MLAH99] Jesper Møller, Jakob Lichtenberg, Henrik R. Andersen, and Henrik Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *SMC'99. First International Workshop on Symbolic Model Checking. Proceedings*, pages 89–108, Trento, Italy, July 1999.

[MP95] Oded Maler and Amir Pnueli. Timing analysis of asynchronous circuits using timed autaomata. In *CHARME'95*, 1995.

[Mye95] Chris J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Stanford University, 1995.

[NSY92] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, September 1992.

[Ove81] W. T. Overman. *Verification of Concurrent Systems: Function and Timing*. PhD thesis, University of California at Los Angeles, 1981.

[Pag96] Florence Pagani. Partial orders and verification of real-time systems. In B. Jonsson and J. Parrow, editors, *Formal Techniques in*

Real-Time and Fault-Tolerant Systems, pages 327–346, Uppsala, Sweden, September 1996. Springer-Verlag.

[Pag97] Florence Pagani. *Ordres partiels pour la vérification de systèmes temps réel (Partial orders for verification of real-time systems).* PhD thesis, Centre d'Études et de Recherches de Toulouse, September 1997.

[Pel93] Doron Peled. All from one, one for all: on model checking using representatives. In Courcoubetis [Cou93], pages 409–423.

[Pel94] Doron Peled. Combining partial order reductions with on-the-fly model checking. In Dill [Dil94], pages 377–390.

[Pel96a] Doron Peled. Combining partial order reduction with on-the-fly model checking. *Formal Methods in System Design*, 8(1):39–64, January 1996.

[Pel96b] Doron Peled. Partial order reduction: Model-checking using representatives. In W. Penczek and A. Szalas, editors, *Mathematical Foundations of Computing Science 1996. 21st International Symposium, MFCS'96. Proceedings*, number 1113 in Lecture Notes in Computer Science, pages 93–112, Cracow, Poland, September 1996. Springer-Verlag.

[PW97] Doron Peled and Thomas Wilke. Stutter–invariant temporal properties are expressible without the next–time operator. *Information Processing Letters*, 63(5):243–246, September 1997.

[RM94] Tomas G. Rokicki and Chris J. Myers. Automatic verification of timed circuits. In Dill [Dil94], pages 468–480.

[RTS96] *Proceedings. 17th IEEE Real-Time Systems Symposium*, Los Alamitos, CA, USA, December 1996. IEEE Computer Society Press.

[SB96] Robert H. Sloan and Ugo Buy. Reduction rules for time Petri nets. *Acta Informatica*, 33(7):687–706, 1996.

[SB97] Robert H. Sloan and Ugo Buy. Stubborn sets for real-time Petri nets. *Formal Methods in System Design*, 11(1):23–40, July 1997.

[SDL93] *Functional Specification and Description Language (SDL). ITU–T Recommendation Z.100*. Geneva, 1993.

[SdS98] D. Sangiorgi and R. de Simone, editors. *CONCUR'98: Concurrency Theory. 9th International Conference. Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, Nice, France, September 1998. Springer-Verlag.

[Taş97] Serdar Taşiran. *Compositional and Hierarchical Techniques for the Formal Verification of Real-Time Systems*. PhD thesis, University of California at Berkeley, 1997.

[Tri98] Stavros Tripakis. *L'Analyse Formelle des Systèmes Temporisés en Pratique (Formal Analysis of Timed Systems in Practice)*. PhD thesis, Université Joseph Fourier, Grenoble, 1998.

[Val90] Antti Valmari. A stubborn attack on state explosion. In Clarke and Kurshan [CK90], pages 156–165.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the Symposium on Logic in Computer Science*, pages 332–344, Cambridge, MA, USA, June 1986. IEEE Computer Society Press.

[Won94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. PhD thesis, Stanford University, December 1994.

[YS97] Tomohiro Yoneda and Bernd-Holger Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):197–215, August 1997.

[YSSC93] Tomohiro Yoneda, Atsufumi Shibayama, Bernd-Holger Schlingloff, and Edmund M. Clarke. Efficient verification of parallel real-time systems. In Courcoubetis [Cou93], pages 321–332.