Formal Aspects of Computing (1998) 10: 469–482
© 1998 BCS

Formal Aspects of Computing

Partial-order Reduction Techniques for Real-time Model Checking

Dennis Dams[1], Rob Gerth[2], Bart Knaack[3] and Ruurd Kuiper[1]

[1]Eindhoven University of Technology, The Netherlands; [2]Strategic CAD Laboratories, Intel Micro Processor Products Group, Hillsboro, OR, USA; [3]Lucent Technologies, Hilversum, The Netherlands

Keywords: Verification; Model checking; Partial-order reduction; Real time; Covering

Abstract. A new notion, *covering*, generalising independence is introduced. It enables improved effects of partial-order reduction techniques when applied to real-time systems. Furthermore, we formulate a number of locally checkable conditions for covering that can be used as the basis for a practical algorithm. Correctness is proven with respect to a chosen discretisation method.

1. Introduction

In recent years, model checking has gained popularity as a verification mechanism, e.g., for communication protocols. A major problem however is the state space explosion. In asynchronous model checkers this explosion is partly due to the interleaving semantics that is used. To alleviate this problem, partial-order techniques have been introduced for untimed model checkers. These reduction techniques are based on the idea that the total set of traces can be divided into a number of equivalence classes, such that checking for absence of failure in the whole set can be done by checking one representative of each class. For an overview of these techniques, see [God96].

Currently the attention is shifting from untimed protocols to timed protocols, where part of the correctness of the behavior of the system depends on timing information. In this paper we model time by introducing densely timed clocks, see e.g. [AlD94]. As the models to be checked become infinite by this addition, we

Correspondence and offprint requests to: Dr Dennis R. Dams, Dept. of Electrical Engineering, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: D.R.Dams@ele.tue.nl.

apply a discretisation method, following the ideas presented in [ACD93]. After the addition of real-time information and discretisation, the state space explosion is worse than before, making it even more urgent to use reduction techniques. An early theoretical result in this field is reported in [Pag96].

The addition of timing information changes the independence relation that forms the corner stone of the partial-order techniques. Often, the reduction of the state space that can be achieved by partial-order techniques is compromised by the global nature of time, see e.g. [Pag97]. Indeed, partial-order reductions are by some considered to be incompatible with real time.

In this paper, a generalisation of the independence relation, called *covering*, is proposed that aims at a better compatibility with real-time systems. Rather than being based on comparing individual states using a symmetric relation, as with classical independence, covering considers *sets* of states and uses the asymmetric set inclusion for comparison. We arrive at a set of criteria that preserve correctness of the model checking and allow a practical algorithm in the sense that they are locally checkable.

In Section 2 we present a formalism to model behavior of real-time systems, together with a discretisation that enables the use of finite-state model checking technology. Section 3 recalls partial-order reduction techniques, discussing the standard condition of independence that needs to be satisfied. In Section 4, which is the heart of this paper, the notion of covering is introduced. Furthermore, a number of locally checkable rules are presented that are sufficient to guarantee covering, and their correcness is proven. Section 5 concludes.

2. Timed Büchi Automata

As syntax we use Timed Büchi Automata [Oli94], an extension of standard Büchi Automata that enables reasoning about timed languages. A Timed Büchi Automaton is a Büchi Automaton extended with a set of real-valued clocks. The intuition is that these clocks act as timers for various activities. To remain consistent with the notion of global time, the clocks are all running at the same speed. However, to capture the timer aspect, each of them can be reset to zero upon traversing an edge of the automaton. Hence time is modeled by the presence of multiple clocks: collectively running at the same speed, but being reset at individual moments. The set of clocks that is reset taking an edge is attached to the edge as the *reset set*. Each edge is furthermore adorned with an *enabling condition*: a transition along it can only occur at a time at which the clocks satisfy this condition.

Traversing the Timed Büchi Automaton, varying amounts of time are spent in states, whereas edges are taken always instantaneously, i.e., during one time instant. Each state is augmented with a *state invariant*. Here the idea is, that the automaton can only remain in a state as long as the state invariant is satisfied, i.e. a transition must be taken before the invariant is violated. The intuition for the state invariant is that one can get stuck in a state (causing deadlock!) because of two reasons. Firstly, this can occur because one arrives in the state too late to comply with the enabling conditions on any of the outgoing edges. Secondly, this can occur because, although arriving in a state in time to allow an outgoing edge to be taken, one waited too long and let all opportunities expire; this is always possible if upper bounds are present in all timing constraints. This second kind of deadlock must – in our opinion – be viewed as a formalism-induced error and if we want to prove our system to be deadlock free we are not interested in these kinds of deadlocks. Appropriately chosen state invariants eliminate this kind of deadlocks. The state invariant provides the possibility to explicitly enforce leaving the state in time, if such is possible.

In the literature sometimes the state invariant is absent. If one is not interested in deadlock detection, enforcing such a form of urgency is not necessary. Even if one is interested in urgency there are other ways of enforcing it. In some frameworks, the requirement to leave the state before the state invariant is violated is for instance replaced by the requirement that if a state is entered before all enabling conditions on the outgoing edges have expired, one edge must be taken in time. We add state invariants, since the intuition about deadlock possibilities and the modeling of a system as a Büchi Automaton is more naturally achieved in a framework with state invariants. Note, however, that translation between the two formalisms is straightforward: adding the state invariant as upper bound to all outgoing edges.

We deliberately postpone giving the precise form of the enabling conditions and the state invariants. These are motivated by the manner in which model checking of time-dependent properties is kept a finite activity. This subject, discretisation, is treated in Subsection 2.1 below. For now, we let $\Phi(C)$ denote the set of all possible predicates on the clock set C, i.e. both on edges (enabling conditions) and in states (state invariants).

Definition 2.1 (Syntax). A *Timed Büchi Automaton* is a tuple $(\Sigma,S,C,E,S_{init}, F,Inv)$ where:

Σ is a finite input alphabet of actions.

S is a finite set of states.

C is a finite set of clocks.

$E \subseteq S \times \Sigma \times S \times 2^C \times \Phi(C)$ is an edge relation. In addition to an action, each edge has a reset set and an enabling condition associated to it.

$S_{init} \subseteq S$ is the set of starting states.

$F \subseteq S$ is the set of acceptance states.

$Inv : S \to \Phi(C)$ gives the state invariants.

We fix a Timed Büchi Automaton $A = (\Sigma, S, C, E, S_{init}, F, Inv)$ for the rest of this paper.

We now turn to semantics. In general, an automaton accepts sequences of actions (traces). A *run* of an automaton records which states are visited in accepting such a sequence, and in which order. For a Timed Büchi Automaton not only the states, but also the times at which they are visited are recorded.

Definition 2.2. A *clock valuation function* v assigns a non-negative real-valued number to each clock. For $\delta \geq 0$, $v + \delta$ denotes the clock valuation v' with $v'(x) = v(x) + \delta$ for all $x \in C$.

A *timed state* is a state $s \in S$ augmented with a clock valuation v, i.e. a tuple (s,v).

In an untimed Büchi Automaton, an action σ is contributed to the trace by taking a transition from some state s such that the corresponding edge from s is labelled with σ. In a Timed Büchi Automaton, an action σ is contributed to the (timed) trace after time duration δ by taking a transition from timed state (s,v) such that the corresponding edge from s is labelled with σ, where the enabling condition on this edge is satisfied by the clock valuation $v + \delta$ and the state invariant is satisfied at all time points between v and $v + \delta$.

Definition 2.3. $(s, v) \xrightarrow{\delta,\sigma} (s', v')$ is a *timed transition* of A if there exists Reset $\in 2^C$ and $\mu \in \Phi(C)$ such that:

- $(s, \sigma, s', Reset, \mu) \in E$
- $\forall x \in C$
$$v'(x) = \begin{cases} v(x) + \delta & \text{if } x \notin Reset \\ 0 & \text{if } x \in Reset \end{cases}$$
- $v + \delta \models \mu$
- $\forall_{\delta' \in [0,\delta]} v + \delta' \models Inv(s)$

In this case, the timed transition is said to be *from* (s, v) *to* (s', v') *along* edge $(s, \sigma, s', Reset, \mu)$. An edge e is enabled in a timed state t if there exists a timed state t' and a timed transition from t to t' along e. $e(t)$ denotes the set of all states t' to which there is a transition along e from t.

Now that we have timed states and transitions, we can define the runs of A:

Definition 2.4. A *timed run* (or *run* for short) of A is a sequence $(s_0, v_0) \xrightarrow{\delta_0,\sigma_0} (s_1, v_1) \xrightarrow{\delta_1,\sigma_1} \cdots$ such that:

- (consecution) for all i, $(s_i, v_i) \xrightarrow{\delta_i,\sigma_i} (s_{i+1}, v_{i+1})$ is a timed transition in A
- (initiation) $s_0 \in S_{init}$ and $\forall_{x \in C}\ v_0(x) = 0$
- (time progress) if the sequence is infinite, then for any number c there exists an n such that $\sum_{i=0}^{n} \delta_i > c$.

The *timed traces* of A are now the sequences of pairs (δ,σ) occurring along runs. A run α is *accepted* (by A) if there is at least one state from A's acceptance set F that occurs infinitely often in it, denoted $\mathsf{Inf}(\alpha) \cap F \neq \emptyset$. A trace is accepted if it occurs along an accepted run. The semantics of A is now taken to be the set of timed traces that it accepts.

Since the time durations δ given with each action in a timed trace are relative to the entrance to the previous state, the total time taken by a part of the timed run is the sum of the time durations δ_i that are associated with this part. In this respect we differ from the notation of [TCH96] where the time durations are absolute with respect to some begin point. The two notions however can easily be transformed into one another. We opted for this notation as it corresponds to observable behavior based directly on clock readings.

2.1. Discretisation

In the formalism given so far, due to the infinite (time) domain that is added by the real-valued clock valuation function, infinitely many different timed states can occur on the timed runs that A accepts. To enable the model checking of such systems we limit the type of clock predicates that are allowed in A. We introduce an abstract semantics based on abstract timed states. We prove that, in terms of model checking, the abstraction is emptiness preserving.

The idea behind the abstract semantics is that given a transition we do not consider each possible delay δ separately, but consider all possible δs at once. We therefore do not have to specify which δ we use, but are only interested in the action occurring along that transition.

To describe the abstract semantics we first have to give some definitions.

Definition 2.5. An *abstract timed state* is a state s augmented with a set M of clock valuations, i.e. a tuple (s, M).

The intuition behind the abstract step is that, given a (begin) set M of clock valuations, a state s and an edge, we can go to another (end) set, M', of clock valuations and another state, s', such that each clock valuation that is reachable from a clock valuation in the begin set — taking a transition corresponding to the given edge — is in the end set; and each clock valuation in the end set should be reachable from a clock valuation in the begin set by taking a transition corresponding to the given edge.

Definition 2.6. $(s, \mathsf{M}) \xrightarrow{\sigma} (s', \mathsf{M}')$ is an *abstract timed transition* of A iff

- for all $v \in \mathsf{M}$ and all δ and v', if $(s, v) \xrightarrow{\delta,\sigma} (s', v')$ is a timed transition of A then $v' \in \mathsf{M}'$, and
- for all $v' \in \mathsf{M}'$, there exist $v \in \mathsf{M}$ and δ such that $(s, v) \xrightarrow{\delta,\sigma} (s', v')$ is a timed transition of A.

Although Definition 2.6 defines the abstract states we are interested in, it is of no use unless we can find a constructive mechanism to find the sets M'. Also, only finitely many abstract states should ensue. Once we have such a mechanism and we have proven it correct, we can use it to construct an implementation. The constructive method should — given a set M of clock valuations and an edge of A — determine the set M' of clock valuations such that $(s, \mathsf{M}) \xrightarrow{\sigma} (s', \mathsf{M}')$ is an abstract timed transition of A, where s and s' are the begin- and end-state of the given edge. This method is referred to in the rest of the paper as the next-time-state function.

We now fix the type of enabling conditions that are allowed to be used on the edges. This will enable us to define the constructive mechanism.

Definition 2.7. A *basic clock inequality* is an inequality of the form $x - y \sim c$ (called *difference inequality*) or $x \sim c$ (called *boundary inequality*) where x and y are clocks, $\sim \in \{<, >, =, \leq, \geq\}$ and $c \in \mathbb{N}$.

The reason for the distinction between difference and boundary inequalities is that passage of time can not change the validity of a difference inequalities, since both clocks run at the same pace, whereas the valuation of boundary inequalities may be changed by the passage of time. Therefore, if we are interested in the time we can let pass before taking an edge we only have to check the boundary inequalities. If difference inequalities are used in an enabling condition they can restrict the number of clock valuations from which this transition can be taken.

Thus, before computing the new timing information we first have to take these restrictions into consideration.

Also the state-invariants are assumed to be basic clock inequalities, that are furthermore required to be *history closed*:

Definition 2.8. A set M of clock valuations is *history closed* if, whenever $v \in \mathsf{M}$, then for all v' and $\delta \geq 0$ such that $v = v' + \delta$, we have $v' \in \mathsf{M}$.

To arrive at a constructive algorithm for the next-time-state function, and to limit the abstract states to a finite number, we require the set of clock valuations of an abstract state to be expressible as a finite conjunction of basic clock inequalities.

The algorithm is based on the following function to compute the result set M' of clock valuation equations given a start set M and an edge $(s, \sigma, s', Reset, \mu)$:

1. Extend M with all timed states that are reachable by passage of time from some point in M (call the result M_1).
2. Restrict M_1 to those time-points that satisfy the state-invariant (M_2).
3. Restrict M_2 to those point satisfying the enabling condition μ (M_3).
4. Change in all clock valuations in M_3 the clocks in *Reset* to zero (M').

An implementable version of this description for so-called *Difference-bound Matrices (DBM)* can be found in [Dil89].

Lemma 2.9. Given a non-empty abstract state (s,M) and an edge $(s, \sigma, s', Reset, \mu)$, the next-time-state algorithm given above computes M' such that $(s, \mathsf{M}) \xrightarrow{\sigma} (s', \mathsf{M}')$ is an abstract timed transition (see Definition 2.6).

Proof. In this proof we first characterize the four intermediate sets we get by the definition above

$\mathsf{M}_1 = \{(s, v') \mid \exists \delta \in R_0^+, (s, v) \in \mathsf{M}$ such that $v' = v + \delta\}$

$\mathsf{M}_2 = \{(s, v') \mid \exists \delta \in R_0^+, (s, v) \in \mathsf{M}$ such that $v' = v + \delta \wedge v' \models Inv(s)\}$. Note that, since we know that the state invariants are historically closed, any point between v and v' satisfies the state-invariant as well.

$\mathsf{M}_3 = \{(s, v') \mid \exists \delta \in R_0^+, (s, v) \in \mathsf{M}$ such that $v' = v + \delta \wedge v' \models Inv(s) \wedge v' \models \mu\}$. The timed states in M_3 are those that are reachable from (s,M) and from which the given transition can fire without delay.

$\mathsf{M}' = \{(s, v') \mid \exists v'', \delta \in R_0^+, (s, v) \in \mathsf{M}$ such that $v'' = v + \delta \wedge v'' \models Inv(s) \wedge v'' \models \mu \wedge (\forall_{x \in Clocks \setminus Reset(s)} v'(x) = v''(x)) \wedge (\forall_{x \in Reset(s)} v'(x) = 0)\}$

To prove the lemma we match the characteristics of the set M' with the definition of timed transitions (see Definition 2.3) and abstract timed transitions (see Definition 2.6).

We can see that if there is timed transition $(s, v) \xrightarrow{\delta,\sigma} (s', v')$ with $v \in \mathsf{M}$ then by Definition 2.3 we can deduce that

- $\forall x \in C$

$$v'(x) = \begin{cases} v(x) + \delta & \text{if } x \notin Reset \\ 0 & \text{if } x \in Reset \end{cases}$$

- $v + \delta \models \mu$
- $\forall_{\delta' \in [0,\delta]}(v + \delta' \models Inv(s))$

It can be easily checked that then $v' \in \mathsf{M}'$.

In reverse for any point v' in M' we know that $\exists v'', \delta \in R_0^+, (s, v) \in \mathsf{M}$ such that $v'' = v + \delta \wedge v'' \models Inv(s) \wedge v'' \models \mu \wedge (\forall_{x \in Clocks \setminus Reset(s)} v'(x) = v''(x)) \wedge (\forall_{x \in Reset(s)} v'(x) = 0)$ holds and therefore by Definition 2.3 $(s, v) \xrightarrow{\delta,\sigma} (s', v')$ is a timed transition.

These two observations prove the lemma. □

There are infinitely many different abstract states, even when each one has to be expressible in terms of finite conjunctions of basic clock inequalities. However, it can be argued that the set of abstract states that is *reachable* by A, and which can be computed by iterative application of the next-time-state function, can be kept finite: see [Dil89] and [AlD94]. Technically, the reason is that, by the finiteness of A, there exists a largest integer to which clock values are being compared by the edge conditions.

The rationale for defining abstract states and transitions is that we want to decide the non-emptiness of A by looking at the abstraction.

Definition 2.10. The sequence $(s_0, \mathsf{M}_0) \xrightarrow{\sigma_0} (s_1, \mathsf{M}_1) \xrightarrow{\sigma_1} \cdots$ is an *abstract run* of A if for all i, $(s_i, \mathsf{M}_i) \xrightarrow{\sigma_i} (s_{i+1}, \mathsf{M}_{i+1})$ is an abstract timed transition of A.

The following lemma is direct by the characterisation of abstract successors by the constructive next-time-state function.

Lemma 2.11. For every run $(s'_0, v_0) \xrightarrow{\delta_0,\sigma_0} (s'_1, v_1) \xrightarrow{\delta_1,\sigma_1} \cdots$ of A, there exists an abstract run $(s_0, \mathsf{M}_0) \xrightarrow{\sigma_0} (s_1, \mathsf{M}_1) \xrightarrow{\sigma_1} \cdots$ of A such that for all i, $s_i = s'_i$ and $v_i \in \mathsf{M}_i$.

The following lemma states that any run through the abstract automaton corresponds to a concrete run. This is less obvious: although for every *single* step $(s_i, \mathsf{M}_i) \xrightarrow{\delta_i,\sigma_i} (s_{i+1}, \mathsf{M}_{i+1})$, there must exist a corresponding "underlying concrete step" $(s_i, v_i) \xrightarrow{\delta_i,\sigma_i} (s_{i+1}, v_{i+1})$ with $v_i \in \mathsf{M}_i$ and $v_{i+1} \in \mathsf{M}_{i+1}$ (otherwise there would have been no such abstract step according to Definition 2.6), this same Definition 2.6 does not guarantee that if we consider a continuation $(s_{i+1}, \mathsf{M}_{i+1}) \xrightarrow{\delta_{i+1},\sigma_{i+1}} (s_{i+2}, \mathsf{M}_{i+2})$ of the abstract run, we can find a concrete step that continues from state (s_{i+1}, v_{i+1}). All that Definition 2.6 provides us with is a continuation from *some* clock valuation in M_{i+1} to a clock valuation in M_{i+2}.

Lemma 2.12. For every abstract run $(s_0, \mathsf{M}_0) \xrightarrow{\sigma_0} (s_1, \mathsf{M}_1) \xrightarrow{\sigma_1} \cdots$ of A for which $\mathsf{M}_i \neq \emptyset$ for all i, there exists a run $(s'_0, v_0) \xrightarrow{\delta_0,\sigma_0} (s'_1, v_1) \xrightarrow{\delta_1,\sigma_1} \cdots$ of A such that for all i, $s_i = s'_i$ and $v_i \in \mathsf{M}_i$.

Proof. For all i we have to pick $v_i \in \mathsf{M}_i$ in such a way that there exist δ_i such that $(s'_i, v_i) \xrightarrow{\delta_i,\sigma_i} (s'_{i+1}, v_{i+1})$. The second point of Definition 2.6 can be used to provide such a sequence of interconnected concrete states (s'_i, v_i) for any *finite* prefix $(s_0, \mathsf{M}_0) \xrightarrow{\sigma_0} (s_1, \mathsf{M}_1) \xrightarrow{\sigma_1} \cdots (s_k, \mathsf{M}_k)$ of the abstract run, by reasoning backwards from (s_k, M_k). However, it is not a priori clear that this infinite collection (one for every finite abstract prefix) of finite prefixes determines an infinite concrete run.

Define for every finite number k the subset $N(k) \subseteq \mathsf{M}_0$ as those valuations in M_0 from which there exists a concrete prefix, of length at least k, that corresponds to the length-k prefix of the abstract run. It should be the case that the sequence $\{N(k)\}_{k\in\omega}$ converges to a nonempty set. This is guaranteed by the fact, proven in [AlD94], that there exist a partitioning of the clock valuations (belonging to a single discrete state) into nonempty sets, so-called *(detailed) regions*, such that two timed states within a region cannot be distinguished by the set of possible runs starting from them, in any Timed Büchi Automaton. □

The above two lemmata together ensure that any run of A corresponds to an abstract run, and reversely. In order to talk about emptiness, we need to lift the notion of acceptance as well.

Definition 2.13. An abstract run α is *accepted* if $\mathsf{Inf}(\alpha) \cap F \neq \emptyset$.

We now have the following

Corollary 2.14. A is empty iff its abstraction is.

3. Partial-order Techniques

In this section, we briefly recapitulate established partial-order reduction techniques for untimed systems, as a basis for the timed case. The idea is that a reduction can be performed of the transition system on which a model checking algorithm operates using a notion of *independence* of edges. We give precise definitions here, but only provide intuition for the correctness; the approach closely follows [GKP95], where detailed proofs can be found. In the next section we show how this approach, most notably the notion of independence, can be generalised to cover real-time systems as well.

Commonly, partial-order techniques are operative in situations where, firstly, (discrete-valued) variables are present. Secondly, the existence of several *processes* is assumed that operate concurrently. Such a system is modelled by a collection of (untimed) Büchi Automata (BAs), in which edges carry state change information and the "data state" at a node of the BA can vary at different traversals through it. A *transition system* is the unfolding of such a BA in the sense that a node of the transition system represents only one data state; this also implies that no information needs to be present on the edges anymore. Conceptually, algorithms for model checking operate on such transition systems rather than directly on BAs. In constructing the global transition system from the individual BAs, it is especially the use of interleaving (i.e. the asynchronous product of BAs) in modelling execution that causes a blow-up of the state space. Partial-order reduction techniques counter this problem, exploiting the fact that for certain pairs of edges in an execution and for certain properties, permutation of the edges does not influence satisfaction. In those cases, it suffices to consider only one representant of each equivalence class (with respect to such permutations) of executions.

More specifically: two edges e, f are allowed to be permuted precisely then, if for all sequences $\mathbf{v}$,$\mathbf{w}$ of edges: if the execution along $\mathbf{v}\ e\ f\ \mathbf{w}$ (where juxtaposition denotes concatenation) gives rise to an accepted trace, then $\mathbf{v}\ f\ e\ \mathbf{w}$ leads to an accepted trace as well. As this is a direct translation of the requirement that satisfaction is preserved, this is obviously the best possible division in equivalence classes as far as reduction is concerned. However, its value for practical usage is limited, since checking the condition is as expensive as unreduced model checking itself, as it involves properties of the whole trace. For memoryless systems, as BAs happen to be, the current state determines the entire future behavior of the system. In this case, edges can safely be permuted if some requirements are satisfied that can be checked locally, i.e., in a state. The central notion is the following:

Definition 3.1. An *independence relation* I for A is a symmetric and irreflexive relation on the set E of edges such that $(e, f) \in I$ if for all states:

- $e, f \in en(s) \ \rightarrow \ e \in en(f(s))$, and
- $e(f(s)) = f(e(s))$,

where $f(s)$ denotes the state reached after taking a transition corresponding to edge f from s and $en(s)$ denotes the edges that are enabled in s.

Edges that are not independent are called dependent. An edge is *safe* if it is independent from every edge in any other process.

Safety of an edge cannot be checked locally, but a sufficient condition that can be checked locally is that it does not touch any global variables (e.g., this is the rule that is implemented in the model checker Spin, see [HoP94]).

The reduction of the search space is now effected during the depth-first search algorithm that forms the heart of the model checking procedure, by limiting[1] the search from a state s to a subset of the edges that are enabled in s. Such a subset R is called a *safe reduction* and is chosen as follows: If there is a process that has only safe edges enabled, then R consists of these edges, otherwise R is the set of all edges enabled in S.

Searching for accepting runs, we can at every state during the depth-first traversal[2] limit the search to the edges in a safe reduction.

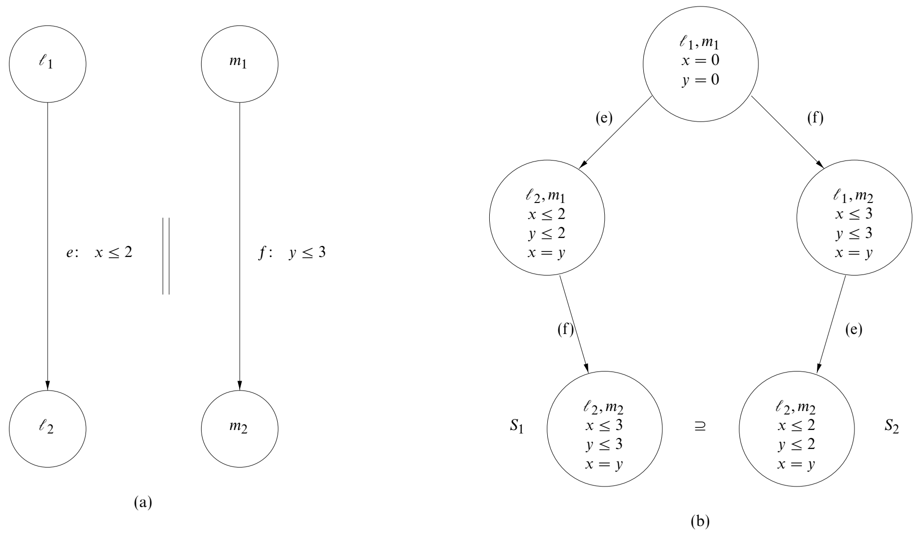
Lemma 3.2. The reduction strategy described above preserves accepting runs.

Proof. The proof is a simplification of the proof given by [HoP94] (the notion of invisibility is not needed as we focus on nonemptiness only). □

Note that we limit ourselves to the case where reduction is only preserving the emptiness and non-emptiness of the set of sequences accepted by a BA. This enables checking more general properties of a system in the following manner. Consider a system S and a property φ both being represented as BAs, A_S and A_φ, where the latter BA accepts exactly those sequences that do not satisfy φ. Establishing that φ satisfies S then amounts to checking for the emptiness of $A_S \times A_\varphi$, where $\times$ represents the synchronous (lock-step) product of the BAs.

[1] There are certain extra conditions that should be obeyed by the reduced dfs algorithm, like the so-called "cycle proviso", that we do not go into here.

[2] By nesting two depth-first searches, cycles can be detected and hence accepting runs; see [HPY96].



ℓ_1
ℓ_2
$e\colon\ x \le 2$
m_1
m_2
$f\colon\ y \le 3$
(a)
ℓ_1, m_1, $x = 0$, $y = 0$
(e)
(f)
ℓ_2, m_1, $x \le 2$, $y \le 2$, $x = y$
ℓ_1, m_2, $x \le 3$, $y \le 3$, $x = y$
(f)
(e)
S_1
ℓ_2, m_2, $x \le 3$, $y \le 3$, $x = y$
$\supseteq$
ℓ_2, m_2, $x \le 2$, $y \le 2$, $x = y$
S_2
(b)

Fig. 1. Partial-order reduction for abstract automata — first generalisation.

4. A Generalisation of Independence

In this section, we present a generalisation of the notion of independence, resulting in a partial-order techniques that gives better reduction when applied to real-time systems. In order to focus on the essence of the problem, our notion of TBA, that is used to model such systems, abstracts away from the presence of variables other than clocks. Again, we assume that the system under consideration is specified by a collection of individual TBA representing processes. A state in the (global) system is a vector of local states of the processes. In particular, the state invariant (see Definition 2.1) in a global state is the *conjunction* of the state invariants associated to the composing local states.

We start with two motivating examples. First, consider the two edges depicted in Fig. 1(a), which are supposed to be part of a system made up of two concurrent timed processes. Consider the topmost abstract state in Fig. 1(b) where the first process is in location ℓ_1, the second process in m_1, and both clocks are 0. Shown along the left side of Fig. 1(b) are the abstract states that are reached by first taking edge e and then f, along the right the abstract states that correspond to taking the edges in the other order. (The reader is invited to check that these abstract states correspond to Definition 2.6.) Clearly, the abstract state S_1 that is reached by taking e before f is not equal to state S_2 that is reached by taking f before e, and by straightforwardly applying the partial-order theory from the previous section we would conclude that both orders have to be taken into consideration by the model-checking algorithm, i.e., that no reduction is possible. However, observe that we do have that $S_1 \supseteq S_2$, meaning that all concrete states that are (implicitly) explored along the right side are also explored along the left. Thus, the shift of consideration from individual states to sets of states allows us to bring asymmetry into the notion of independence. The independence condition presented in the previous section, which ensured that it does not matter in which order the edges are taken, can be replaced by a notion that restricts this freedom

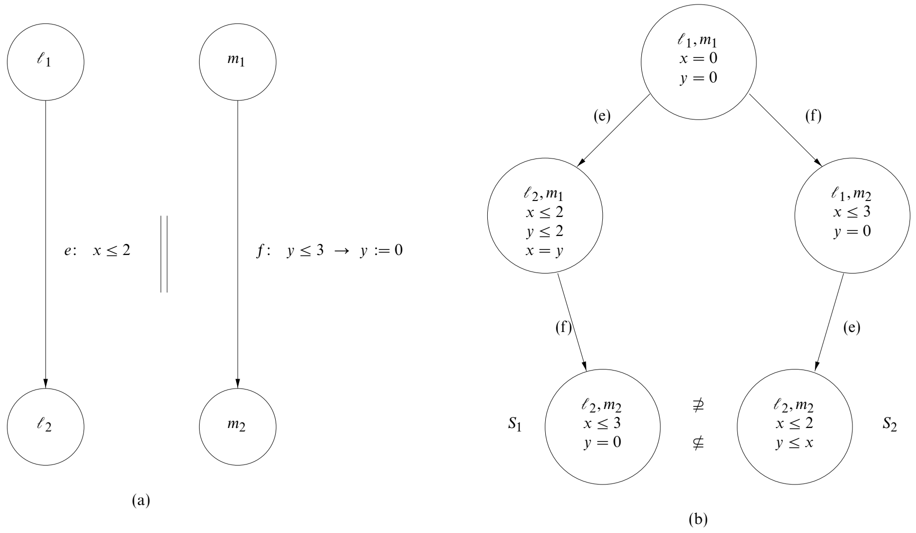

ℓ_1
m_1
e: $x \le 2$
f: $y \le 3 \to y := 0$
ℓ_2
m_2
(a)
ℓ_1, m_1 $x = 0$ $y = 0$
(e)
(f)
ℓ_2, m_1 $x \le 2$ $y \le 2$ $x = y$
ℓ_1, m_2 $x \le 3$ $y = 0$
S_1
ℓ_2, m_2 $x \le 3$ $y = 0$
$\not\supseteq$
$\not\subseteq$
ℓ_2, m_2 $x \le 2$ $y \le x$
S_2
(b)

Fig. 2. Partial-order reduction for abstract automata — second generalisation.

to one specific order that has to be considered (in this example, e before f), but with the advantage that it is a weaker condition ($\subseteq$ instead of $=$). This indeed generalises reduction techniques in general, no use being made of the fact that the variables here represent clocks. We now further extend the notion for clock variables specifically.

We modify the example by adding a reset of clock y to edge f, see Fig. 2(a). The resulting (part of the) abstract transition system is shown in Fig. 2(b). Now, we have neither $S_1 \subseteq S_2$ nor $S_2 \subseteq S_1$, so even the generalisation proposed above would not allow any reduction. For example, the timed state $s = \langle \ell_2, m_2, x = 2, y = 2 \rangle \in S_2$ is not included in S_1. However, observe that it is possible to reach s from the state $s' = \langle \ell_2, m_2, x = 0, y = 0 \rangle \in S_1$, by letting 2 time units pass. Hence, any timed transition that can be made from s (by first letting an amount δ of time pass and then making an instantaneous step, see Definition 2.3) can also be made from s', by first letting $\delta + 2$ time pass. So in this respect, any part of the state space that can be reached from $s \in S_2$ is "covered" by the state space reachable from $s' \in S_1$. Indeed, this holds for all states in S_2: letting $S_1^{\uparrow}$ denote the set of states that can be reached from some state in S_1 by letting an arbitrary amount of time pass, we have that $S_1^{\uparrow} \supseteq S_2$.

4.1. Covering

The two examples suggest to replace the definition of independence by an asymmetric notion which we call *covering*. This notion considers the enabledness and execution of edges in *sets* of timed states. Define $e(S) = \bigcup\{e(s) \mid s \in S\}$, and say that e is enabled in S if e is enabled in some $s \in S$. As above, $S^{\uparrow}$ denotes the set of states that can be reached from some state in S by letting an arbitrary amount of time pass.

Definition 4.1. Edge e *covers* f if for all sets S of states, $e(f(S)) \subseteq (f(e(S)))^{\uparrow}$.

In words: any state that can be reached from S by first taking f and then e can also be reached from S by first taking e and then f, and then possibly waiting some time. In the definition of safe edge, we now replace the independence condition by a covering condition, as follows: An edge is safe if it covers any other edge that is concurrent. Sufficient conditions for covering that are locally checkable, are presented below. Finally, a safe reduction R of the edges that are enabled in some set S of states, is constructed in the same way as before: If there is a process that has only safe edges enabled, then R consists of these edges, otherwise R is the set of all edges enabled in S.

Again, the idea is that in order to search for an accepting run from (some state in) S, we only have to consider those runs that start with an edge from some safe reduction. The proof of this is a straightforward adaptation of the proof for the "standard" case. The intuition is that, for an edge e that covers f, by considering their execution in the order e, f, we do not miss any states that we might have reached by performing them in the reverse order. Indeed, by first executing f we may deadlock — but then for sure we do not "miss" any (infinite) accepting runs.

By the quantification over all sets of states in Definition 4.1, computing the covering relation between edges involves the complete transition system. Below, we give sufficient conditions that are locally checkable.

4.2. Criteria for Safety

We now give criteria that are, together, sufficient to guarantee safety (in the new sense) of an edge e in a Timed Büchi Automaton. In contrast to the notion of covering, these criteria are easily seen to be locally checkable, thus enabling automated verification.

The criteria are quite technical; the intuition behind them can best be understood from the proof of Lemma 4.2.

B_1 e's enabling condition c_e only refers to local clocks (i.e. clocks that occur only in enabling conditions or resets of edges in the same process as e);

B_2 e's reset set is empty;

B_3 for any edge f, with enabling condition c_f, that appears in a different process than e, we have:

If $v \in c_f$ then either $v \in c_e$ or $\forall_{\delta \geq 0}\ v + \delta \notin c_e$.

B_4 (a) the state invariant in the target state of e is not stricter than the state invariant in the source state of e

or

(b) the clocks that occur in the invariant of the target state of e are not reset along f.

Lemma 4.2. If an edge e satisfies the above criteria B_1–B_4, then it is safe.

Proof. (Terminology: For a timed state (s, v) and an edge e with enabling condition c_e and reset set R_e, we say that e can be taken from (s, v) if there exists a nonnegative real number δ_e such that $v + \delta_e \in c_e$, and that e can be *immediately* taken if this δ_e equals zero. The resulting state is denoted $(e(s), R_e(v + \delta_e))$.)

First consider the case where no invariants are used, i.e., all invariants are true.

Let f be an edge, with enabling condition c_f and reset set R_f, from another process than e (with enabling condition c_e — e has an empty reset set by B_2). We have to show that e covers f. Let S be an arbitrary set of timed states. Then $e(f(S)) \subseteq (f(e(S)))^{\uparrow}$ should hold. Let $(s, v) \in S$ and assume that we can take f and then e from (s, v), i.e. (*) $v + \delta_f \in c_f$ and (**) $R_f(v + \delta_f) + \delta_e \in c_e$, for nonnegative δ_f and δ_e. The resulting state is $(e(f(s)), R_f(v + \delta_f) + \delta_e)$. Because R_f does not involve any clock on which c_e depends (this is by B_1), we have from (**) that (***) $v + \delta_f + \delta_e \in c_e$. From (*), (***) we have by B_3 that $v + \delta_f \in c_e$, so we can take e from (s, v), getting to $(e(s), v + \delta_f)$. As f can be immediately taken from $(s, v + \delta_f)$, it can also immediately be taken from $(e(s), v + \delta_f)$, getting to $(f(e(s)), R_f(v + \delta_f))$, which is equal to $(e(f(s)), R_f(v + \delta_f))$ — this is because the effect on the "untimed" part of the state is insensitive to the relative order of execution of the two edges, as they originate from different processes. By letting δ_e time pass, we get to $(e(f(s)), R_f(v + \delta_f) + \delta_e)$, which is the same state that we reached by first taking f and then e.

Secondly, consider the case where there are nontrivial invariants present.

The argument showing that first e and then f can be taken proceeds as in the first part of the proof, but for one proviso: In the (global) state that corresponds to the source state of f and the target state of e, f can only be executed if the invariant has not yet expired. This invariant consists of the conjunction of the source invariant of f and the target invariant of e. We consider the two possible cases.

1. Assume the source invariant of f has expired. But note that the time in this state is $v + \delta_f$ (see first part of proof), and at that time f was still possible by the source invariant of f — as can be seen from the transitions along first f and then e. Contradiction.
2. Assume that (****) the target invariant of f has expired.
   - If $B_4(a)$ holds, then (****) implies that e could not have been taken in the original sequence (first f, then e) already. Contradiction.
   - If $B_4(b)$ holds, then (****) implies that blocking occurs in $e(f(s))$ so that also the original sequence blocks, albeit in a later state. This means that in this subcase also the original sequence could not have been present. □

5. Conclusions and Future Work

We have studied the interaction between partial-order reduction techniques and real-time in model checking. In order to focus on the essence of timed behaviour, our investigations are on a version of Timed Büchi Automata that abstract away from the presence of discrete-valued variables.

We first recalled (from [Dil89]) how to discretise the continuum forming the state space of such automata into *abstract states* in such a way that the results of emptiness checks are preserved. Then, the principles of partial-order techniques were reviewed; most notably the notion of independent edges. Our main results were presented in Section 4, where we have adapted the notion of independence to *covering*, to better suit systems with clocks, and given a number of locally checkable rules. We have proven that these rules are a sufficient condition for an adapted notion of safety of edges that is based on covering instead of independence. These rules can be used as a basis for a model checking algorithm over the abstract state space that selects, in any abstract state, a subset of the enabled edges for further (depth-first) exploration. Thus, it avoids an important cause for the state explosion, namely the exploration of different interleavings that lead to the same abstract states.

Ongoing research considers the generalisation of our results to Timed Büchi Automata that include discrete-valued variables, and the implementation of the resulting set of rules in a model checker.

Two ideas behind the generalisation from independence to covering, namely the lifting to the level of *sets* of states, and the replacement of equality by set inclusion, may turn out to be valuable in the general, untimed case as well. As the resulting condition is weaker, an improved effect in state space reduction may be expected. Furthermore, the formulation in terms of sets of states may open the road to using partial-order techniques in symbolic model checking. We are currently investigating these issues.

Acknowledgements

We would like to thank Maarten Bodlaender, Dragan Bošnački, and Rob Nederpelt for their comments, fruitful discussions and suggestions.

References

[ACD93] Alur, R., Courcoubetis, C. and Dill, D.: Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[AlD94] Alur, R. and Dill, D. L.: A theory of timed automata. *Journal of Theoretical Computer Science*, 126:183–235, 1994.

[Dil89] Dill, D. L.: Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems (CAV89)*, number 407 in LNCS, pages 197–212, Berlin, 1989. Springer-Verlag.

[GKP95] Gerth, R., Kuiper, R., Peled, D. and Penczek, W.: A partial order approach to branching time logic model checking. In *Third Israel Symposium on the Theory of Computing and Systems*, pages 130–139, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[God96] Godefroid, P.: *Partial-Order Methods for the Verification of Concurrent Systems.* Number 1032 in LNCS. Springer, Berlin, 1996.

[HoP94] Holzmann, G. J. and Peled, D.: An improvement in formal verification. In *Proceedings FORTE'94*, 1994.

[HPY96] Holzmann, G., Peled, D. and Yannakakis, M.: On nested depth-first search. In *Second SPIN Workshop*, 1996.

[Oli94] Olivero, A.: *Modélisation et analyse des systèmes temporisés et hybrides.* PhD thesis, Institut National Polytechnique de Grenoble, 1994.

[Pag96] Pagani, F.: Partial orders and verification of real time systems. In Bengt Jonsson and Joachim Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in LNCS, Berlin, 1996. Springer.

[Pag97] Pagani, F.: *Ordres partiels pour la vérification de systèmes temps réel.* PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace, 10, avenue Édouard-Belin, B.P. 4032, 31055 Toulouse Cédex 4, France, July 1997.

[TCH96] Tripakis, S., Courcoubetis, C. and Holzmann, G.: Extending PROMELA and Spin for real time. In *TACAS'96*, number 1055 in LNCS, Berlin, 1996. Springer.

Received March 1998
Accepted in revised form October 1998