7-2014

Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions

Henri HANSEN

Shang-Wei LIN

Yang LIU

Truong Khanh NGUYEN

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Citation

HANSEN, Henri; LIN, Shang-Wei; LIU, Yang; NGUYEN, Truong Khanh; and SUN, Jun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. (2014). *Proceedings of the 26th International Conference, CAV 2014, Vienna, Austria, July 18–22*. 391-406.
Available at: https://ink.library.smu.edu.sg/sis_research/4956

Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions

Henri Hansen[1], Shang-Wei Lin[2], Yang Liu[3], Truong Khanh Nguyen[4], and Jun Sun[5]

[1] Tampere University of Technology, Department of Mathematics
henri.hansen@tut.fi
[2] Temasek Laboratories, National University of Singapore
[3] School of Computer Engineering, Nanyang Technological University
[4] National University of Singapore
[5] Singapore University of Technology and Design

Abstract. A major obstacle for using partial order reduction in the context of real time verification is that the presence of clocks and clock constraints breaks the usual *diamond structure* of otherwise independent transitions. This is especially true when information of the relative values of clocks is preserved in the form of diagonal constraints. However, when diagonal constraints are relaxed by a suitable abstraction, some diamond structure is re-introduced in the zone graph. In this article, we introduce a variant of the stubborn set method for reducing an abstracted zone graph. Our method works with all abstractions, but especially targets situations where one abstract execution can simulate several permutations of the corresponding concrete execution, even though it might not be able to simulate the permutations of the abstract execution. We define independence relations that capture this "hidden" diamond structure, and define stubborn sets using these relations. We provide a reference implementation for verifying timed language inclusion, to demonstrate the effectiveness of our method.

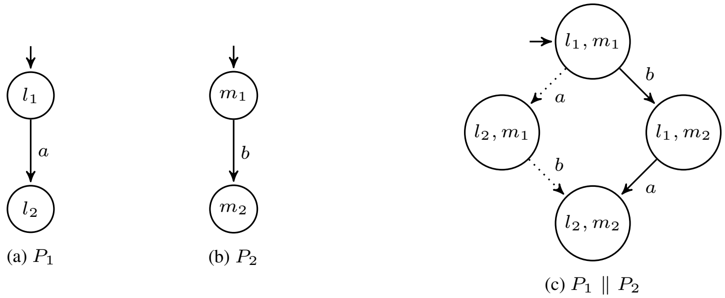
1 Introduction

State space methods for timed systems have to deal with not only *state explosion* but also *clock explosion*, i.e., complexity resulting from time constraints of the runs of the system. In a non-timed system, state explosion caused by concurrency and interleaving semantics can often be alleviated by commutativity based reductions, a.k.a. *partial order reductions*, that work by eliminating unnecessary interleaving of sequences.

Fig. 1 shows a simple example of how partial order reduction works. Two processes P_1 and P_2 can perform events a and b, respectively, as shown in Figs. 1 (a) and (b). The concurrent behaviors, ab and ba, of P_1 and P_2 constitute a diamond structure as shown in Fig. 1 (c). If the property checks for the reachability of state l_2m_2, it is sufficient to only explore the representative path ba marked in solid arrows.

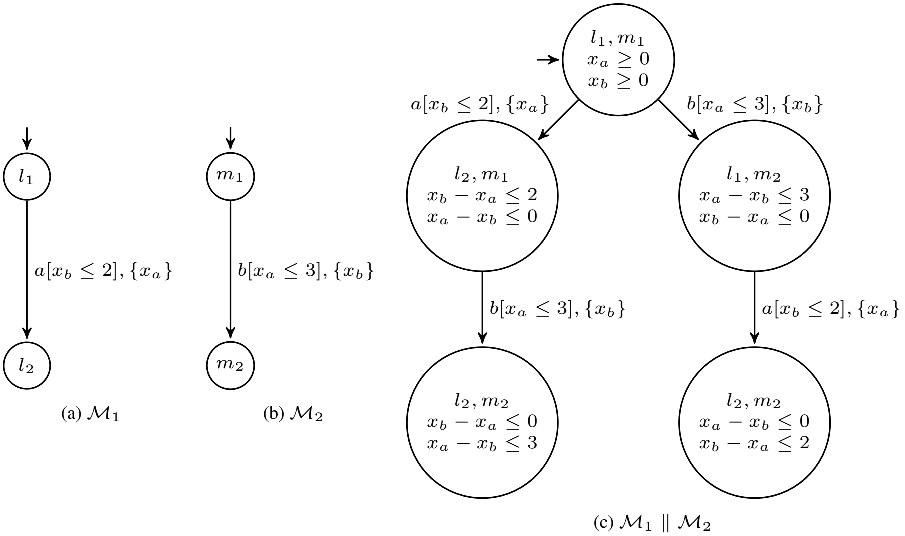
The presence of clocks interferes with partial order reduction, because the relative order of events is preserved in time stamps. Consider a simple timed system of two concurrent events, a and b, and two clocks x_a and x_b, which record the time elapsed since the previous occurrence of the events. If both events occur, but a takes place before

A. Biere and R. Bloem (Eds.): CAV 2014, LNCS 8559, pp. 391–406, 2014.



(a) P_1 (b) P_2 (c) $P_1 \parallel P_2$

Fig. 1. Diamond Structure


l_1
$a[x_b \leq 2], \{x_a\}$
l_2
m_1
$b[x_a \leq 3], \{x_b\}$
m_2
l_1, m_1 $x_a \geq 0$ $x_b \geq 0$
$a[x_b \leq 2], \{x_a\}$
$b[x_a \leq 3], \{x_b\}$
l_2, m_1 $x_b - x_a \leq 2$ $x_a - x_b \leq 0$
l_1, m_2 $x_a - x_b \leq 3$ $x_b - x_a \leq 0$
$b[x_a \leq 3], \{x_b\}$
$a[x_b \leq 2], \{x_a\}$
l_2, m_2 $x_b - x_a \leq 0$ $x_a - x_b \leq 3$
l_2, m_2 $x_a - x_b \leq 0$ $x_b - x_a \leq 2$

(a) $\mathcal{M}_1$ (b) $\mathcal{M}_2$ (c) $\mathcal{M}_1 \parallel \mathcal{M}_2$

Fig. 2. Broken Diamond Structure

b, then the time constraint $x_a \geq x_b$ will hold, and if the order is reversed, then $x_b \geq x_a$ will hold. Fig. 2 shows the broken diamond structure that results from time constraints.

Abstraction in this article refers to relaxing of some constraints of a system so that we will lose the ability to distinguish between some configurations. We deal exclusively with time abstraction and safety in this article. When verifying safety properties, abstractions give over-approximations, so that all errors are preserved, and some new errors may be introduced. *Abstraction refinement* means that verification starts with a coarse over-approximation which is then refined until either the property is verified or a concrete counterexample is found.

The objectives of this article are the following. Firstly, we define novel relations called *weak and strong independence* for an abstract transition system. They guarantee that one order of executing two independent abstract events can simulate the other order. Strong independence is symmetric, but weak independence is not. Fig. 2 serves as an example. Observing the bottom left configuration, if we relax the constraint $x_b - x_a \leq 0$

and replace it with a constraint $x_b - x_a \leq n$ for any sufficiently large n, the resulting abstract configuration can simulate the configuration on the bottom right of the same figure. The independence relations preserve their validity when an abstraction is made coarser, which is summarized in Theorem 1.

Secondly, we modify the stubborn set method to make use of these relations and reduce an abstract state graph. Our reduction works so that if the original state graph contains a counterexample, then the reduced version of the abstract state graph contains one as well, and this is proven in Theorem 2. Due to the two theorems, our theory is general enough, so that it could be combined with any form of abstraction, as long as one can analyse the independence relations for some finer grained abstraction.

We chose to experiment with the approach in combination with an abstraction refinement loop. The abstraction in our implementation combines a simple family of abstractions that omit some diagonal constraints, with LU-simulation check. Even this rudimentary implementation provides excellent improvement in scalability.

Organization. In the following we discuss how our work relates to previous work in the literature. In Section 2, we define timed automata, timed languages, and the composition of a system from component automata, and their semantics over transition systems. Section 3 defines the stubborn set reduction for an abstract transition system, and explains a state exploration algorithm for checking non-emptiness under reduction. In Section 3.3, we discuss one possible implementation. Section 4 discusses some experiments, while the final section concludes.

Related Work. The seminal work on stubborn sets are [16] and [17]. In particular, [17] explores the use of stubborn sets in a synchronous model. Both deal with *strong* stubborn sets, although earlier work does identify weak sets as well. Dependency and reduction of the control structures for weak stubborn sets have been presented in [9], along with an algorithm for calculating stubborn sets. This article generalizes weak and strong (in)dependence to time constraints. Weak sets have the potential (at least in theory) to reduce more than strong sets.

The theory of timed automata is mostly from [1]. We use the original timed automata definition that does not include invariants. Invariants can be taken into account in our theory as additional guards for transition entering or leaving locations that have them, without compromising safety. Earlier work on partial order reduction for timed automata includes [4] and [12], which identify the problems related to commutativity. Both consider a concept of *local time*, where delays are either global or local to component automata, but provide no empirical evidence. The problematic nature of time zones is also discussed in [5], where a concept called *covering* is applied. Weak independence is a generalization of covering, and localized time can be viewed as an abstraction technique compatible with our method. *Event zones* that record the time elapsed between given events, have been used in [11] and [13] to implement Mazurkiewicz-trace-reduction, which is based on a symmetric concept of independence. Various abstraction techniques for zones exist [6,3,10], we combined our method with the latter two. The idea behind our timed abstraction refinement loop originates from [2].

An alternative approach to using commutativity is discussed in [14]. The method is a search where the zones resulting from different permutations of a set of events are merged. The exact relationship to our method is unknown to us, but we conjecture that the two methods can be combined to increase the effectiveness of both; we leave this for future work.

2 Preliminaries

Let Σ be a finite alphabet and $\mathbb{R}^+$ be the set of non-negative real numbers. A *timed word* over Σ is a finite sequence $w_t = (a_1, t_1)(a_2, t_2)\ldots(a_n, t_n) \in (\Sigma \times \mathbb{R}^+)^*$, such that the sequence $t_1 t_2 \ldots t_n$ of time-stamps is non-decreasing.

Let C be a set of clocks where a clock is a variable over non-negative real numbers $\mathbb{R}^+$. We assume that all clocks progress at the same rate. Let $\sim \in \{<, \leq, \geq, >\}$ and $\prec \in \{<, \leq\}$. An *atomic clock constraint* η is defined as $\eta = x_a \sim n \mid x_a - x_b \prec n$ for $x_a, x_b \in C$, and $n \in \mathbb{Z}$. A *clock constraint* ϕ is a conjunction of atomic clock constraints.

A clock constraint ϕ identifies a convex $|C|$-dimensional polyhedron $[\![\phi]\!] \subseteq (\mathbb{R}^+)^{|C|}$. An *atomic clock guard* is an inequality of the form $x_a \sim n$ for $x_a \in C$, $\sim \in \{<, \leq, >, \geq\}$, and $n \in \mathbb{N}$. A *clock guard* g is a conjunction of atomic clock guards. A clock guard g identifies a $|C|$-dimensional cuboid $[\![g]\!] \subseteq (\mathbb{R}^+)^{|C|}$. We use G_C to denote the set of clock guards over C, and $G_C^A \subseteq G_C$ to denote the set of atomic clock guards.

A *clock valuation* $\gamma : C \mapsto \mathbb{R}^+$ assigns a non-negative real number to a clock. For a clock valuation γ, *clock resetting* $c \subseteq C$, denoted by $\gamma[c \mapsto 0]$, is the clock valuation γ' such that $\gamma'(x) = 0$ for all $x \in c$ and $\gamma'(y) = \gamma(y)$ for all $y \in C \setminus c$. Given a constant $d \in \mathbb{R}^+$ and a clock valuation γ, we use $\gamma + d$ to denote the valuation such that $(\gamma + d)(x) = \gamma(x) + d$ for all $x \in C$. The set of clock valuations is denoted Γ_C.

Definition 1. *Let C be a set of clocks. A* timed automaton *(TA) over C is a tuple $T = (\Sigma, L, L^0, \delta, L^f)$, where Σ is a finite input alphabet, L is a finite set of locations, $L^0 \subseteq L$ is a set of initial locations, $L^f \subseteq L$ is a set of accepting locations, and $\delta : L \times \Sigma \times G_C \times 2^C \mapsto 2^L$ is a partial transition function.*

In a transition $\delta(l, a, g, c)$, l is the starting control location, a is the event, g is a guard and c is the set of reset clocks, while the result is a set of control locations. It is common to think of transitions as edges between two control locations that are decorated with a, g and c. For convenience, sometimes we write $l \xrightarrow{a[g],c} l'$ or even $l \xrightarrow{a} l'$ when $l' \in \delta(l, a, g, c)$ for some $l, l' \in L$, $a \in \Sigma$, $g \in G_C$, and $c \subseteq C$. When such l' exists, we write $l \xrightarrow{a}$. $l \xrightarrow{ab} l'$ means there is some $l^* \in L$ such that $l \xrightarrow{a} l^*$ and $l^* \xrightarrow{b} l'$. We generalise this to longer sequences in the natural way.

We write $\mathcal{R}(a)$ as the union of all c such that $\delta(l, a, g, c)$ is defined for some l and g, i.e., $\mathcal{R}(a)$ is the set of clocks that *could* be reset by executing a. Likewise $\mathcal{G}(a)$ is the set of clocks that appear in some g such that $\delta(l, a, g, c)$ is defined for some l and c.

Definition 2. *A* run *σ of a TA $M = (\Sigma, L, L^0, \delta, L^f)$ over a timed word $w_t = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ is a finite sequence of the form*

$$(l_0, \gamma_0) \xrightarrow[t_1]{a_1} (l_1, \gamma_1) \xrightarrow[t_2]{a_2} (l_2, \gamma_2) \xrightarrow[t_3]{a_3} \cdots \xrightarrow[t_n]{a_n} (l_n, \gamma_n)$$

with $l_i \in L$ and $\gamma_i \in \Gamma_C$ for all $0 \leq i \leq n$, satisfying the following requirements:

- $l_0 \in L^0$ *and* $\gamma_0(x) = 0$ *for all* $x \in C$
- *there is a transition* $l_{i-1} \xrightarrow{a_i[g_i],c_i} l_i$ *such that* $(\gamma_{i-1} + t_i - t_{i-1}) \models g_i$ *and* $\gamma_i = (\gamma_{i-1} + t_i - t_{i-1})[c_i \mapsto 0]$ *for all* $1 \leq i \leq n$

A run σ is an accepting run if $l_n \in L^f$. A timed word w_t is *accepted* by M if M has an accepting run over w_t. The *timed language* accepted by M, denoted by $\mathcal{L}(M)$, is the set of all the timed words accepted by M.

We call the set $V = \{a \in \Sigma \mid \exists l \in L \setminus L^f : \exists l' \in L^f : l \xrightarrow{a} l' \vee l' \xrightarrow{a} l\}$ *visible* events. Visible events are the events whose occurrence may change the control location from accepting to non-accepting or vice versa. For future reference, V does not need to be exact, approximating with a larger set will be sufficient.

Given a set of timed automata $M_i = (\Sigma_i, L_i, L_i^0, \delta_i, L_i^f)$ for $i \in \{1, 2, \ldots, n\}$, their *parallel composition* is the timed automaton $M_1 \parallel \cdots \parallel M_n = (\Sigma, L, L^0, \delta, L^f)$ where $\Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i$, $L = L_1 \times \cdots \times L_n$, $L^0 = (L_1^0, \ldots, L_n^0)$, $L^f = L_1^f \times \cdots \times L_n^f$, and the transition relation δ is defined as follows. Let $\Sigma(a) = \{i \mid a \in \Sigma_i\}$. Then $(l_1, \ldots, l_n) \xrightarrow{a[\bigwedge_{i \in \Sigma(a)} g_i], \cup_{i \in \Sigma(a)} c_i} (l_1', \ldots l_n')$, if (1) $l_i \xrightarrow{a[g_i],c_i} l_i'$, whenever $i \in \Sigma(a)$, and (2) $l_i = l_i'$, whenever $i \notin \Sigma(a)$.

If a clock constraint ϕ is satisfiable, there is a unique canonical clock constraint, denoted by $Can(\phi)$, among all the clock constraints identifying the polyhedron $[\![\phi]\!]$, obtained by closing ϕ under all consequences of pairs of conjuncts in ϕ. Let $C_0 = C \cup \{x_0\}$ where x_0 is the *dummy clock*. We assume $x_0 = 0$ at all times. $Can(\phi)$ can always be expressed as $\bigwedge_{x,y \in C_0} x - y \prec_{xy} n_{xy}$. A common canonical representation is the *difference bound matrix* or DBM. A DBM represents $Can(\phi)$ in the following way. Given a numbering $\{0, 1, \ldots, |C|\}$ for the set of clocks, we represent any satisfiable constraint as a matrix $D = \langle n_{ij}, \prec_{ij} \rangle$, where $i, j \in \{0, 1, \ldots, |C|\}$. The conjunct $x_i - x_j \prec_{ij} n_{ij}$ is represented by the entry $\langle n_{ij}, \prec_{ij} \rangle$. The index 0 corresponds to the dummy clock, so that a lower bound $x_i \prec n_{i0}$ is represented by $\langle n_{i0}, \prec \rangle$, and an upper bound $-x_i \prec n_{0i}$ is represented by $\langle n_{0i}, \prec \rangle$.

Given a clock constraint ϕ, we define the *reset* of a set of clocks c in ϕ, denoted by $\phi[c \mapsto 0]$, as $Can(\phi[c \mapsto 0])$. This set of constraints is obtained from $Can(\phi)$ by removing all conjunctions where some $x \in c$ is included, adding the conjunct $x = 0$, and closing w.r.t. the remaining conjuncts. We define the time elapsing of ϕ, denoted by $\phi \uparrow$, as $Can(\phi \uparrow)$ where $\phi \uparrow$ is obtained from $Can(\phi)$ by removing all upper bounds on clocks. For example, given a constraint $\phi : 0 \leq x \leq 3 \wedge 0 \leq y \leq 2$, its canonical form is $Can(\phi) : 0 \leq x \leq 3 \wedge 0 \leq y \leq 2 \wedge -3 \leq y - x \leq 2$, $\phi[\{x\} \mapsto 0] : x = 0 \wedge 0 \leq y \leq 2 \wedge 0 \leq y - x \leq 2$, and time elapsing $\phi \uparrow$: $0 \leq x \wedge 0 \leq y \wedge -3 \leq y - x \leq 2$.

Given a precondition ϕ, and an event a with guard g_a and reset clocks c_a, we define the *strongest postcondition* of a as $sp(\phi, (a, g_a, c_a)) = ((\phi \wedge g_a)[c_a \mapsto 0]) \uparrow$. We define a *abstract postcondition* $POST_\alpha$ as a mapping that satisfies $sp(\phi, (a, g_a, c_a)) \subseteq POST_\alpha(\phi, (a, g, c))$. If $POST_{\alpha_1}$ and $POST_{\alpha_2}$ are two abstract postconditions such that for every a, g and c, $POST_{\alpha_1}(\phi, (a, g, c)) \subseteq POST_{\alpha_2}(\phi, (a, g, c))$, we write $\alpha_1 \sqsubseteq \alpha_2$, and we say that α_2 is a *coarser abstraction*.

Definition 3. *The semantics of a timed automaton $M = (\Sigma, L, L_0, \delta, L^f)$ is defined by an* abstract transition system $(S, S_0, \Longrightarrow_\alpha)$*, where $S \subseteq L \times \mathcal{P}((\mathbb{R}^+)^{|C|})$, and $S_0 = \{(l_0, Z_0) \mid l_0 \in L_0\}$. Z_0 is called the* initial zone. *"$\Longrightarrow_\alpha$" $\subseteq S \times \Sigma \times S$ is defined as follows:*

- *$(l, Z) \overset{a}{\Longrightarrow}_\alpha (l', Z')$, if and only if $a \in \Sigma$ and $l' \in \delta(l, a, g, c)$ for some g and c, and $Z' = POST_\alpha(Z, (a, g, c)) \neq \emptyset$.*

where $POST_\alpha(Z, (a, g, c))$ is an abstract postcondition operation. When such (l', Z') exists, we write $(l, Z) \overset{a}{\Longrightarrow}_\alpha$.

Given $(l, Z) \in S$, we write $en_\alpha((l, Z)) = \{a \mid \exists (l', Z') : (l, Z) \overset{a}{\Longrightarrow}_\alpha (l', Z')\}$, for the set of enabled events at state (l, Z). The abstraction is determined by the postcondition $POST_\alpha(Z, (a, g, c))$. We leave that open for now, as well as what Z_0 really is. We write $POST_\alpha(Z, a)$ and omit the guard g and the set of clocks c if they do not matter or are clear from the context. We abuse the notations to write $(l, Z) \overset{a}{\Longrightarrow}_{sp} (l', Z')$, when $Z' = sp(Z, (a, g, c))$ for some g, c and $l \xrightarrow{a} l'$, even if (l, Z) is not an actual state of the transition system we are discussing. When $Z \subseteq Z'$ we say that the state (l, Z') *simulates* the state (l, Z), and write $(l, Z) \prec (l, Z')$.

Definition 4. *Given an abstract transition system $(S, S_0, \Longrightarrow_\alpha)$, a sequence $\langle (l_0, Z_0), a_1, (l_1, Z_1), \ldots, a_n, (l_n, Z_n) \rangle$ such that $(l_{i-1}, Z_{i-1}) \overset{a_i}{\Longrightarrow}_\alpha (l_i, Z_i)$ and $l_n \in L^f$ is called* a counterexample *of the transition system.*

The following proposition is the basis of timed verification using transition systems, and it is a standard result [1].

Proposition 1. *If $(S, S_0, \Longrightarrow_{sp})$ is the transition system of the timed automaton $M = (\Sigma, L, L_0, \delta, L^f)$ under strongest postcondition, then $\mathcal{L}(M) = \emptyset$ if and only if the transition system does not have a counterexample.*

A corollary to Proposition 1 follows immediately from the assumption that $sp(Z, a) \subseteq POST_\alpha(Z, a)$: If an abstract transition system of M under α has no counterexamples, then $\mathcal{L}(M) = \emptyset$.

3 Reduction of Abstract Transition Systems

We define reduction functions for abstract transition systems without specifying the abstraction function. The reduction preserves the existence of counterexamples of the concrete system, of which the abstract system is an over-approximation; the existence of spurious counterexamples may not be preserved.

3.1 Stubborn Sets

Definition 5. *Given a timed automaton $M = (\Sigma, L, L^0, \delta, L^f)$ and its abstract transition system $(S, S_0, \Longrightarrow_\alpha)$, we define a* reduction function *as $T : S \to 2^\Sigma$. Given a reduction T, we define the* reduced abstract transition system $(S_T, S_0, \Longrightarrow_\alpha^T)$ *as the minimal transition system such that,*

- $S_0 \subseteq S_T$
- *if* $s \in S_T$, $a \in T(s)$, *and* $s \overset{a}{\Longrightarrow}_\alpha s'$, *then* $s' \in S_T$ *and* $s \overset{a}{\Longrightarrow}{}^{T}_{\alpha} s'$.

Definition 6. *Given a timed automaton* $M = (\Sigma, L, L^0, \delta, L^f)$, $I^S \subseteq \Sigma \times \Sigma$ *is a*

- strong structural independence relation *if for all* $(a, b) \in I^S$ *and all locations* $l, l' \in L$ *such that* $l \xrightarrow{a} \wedge l \xrightarrow{b}$ *we have* $l \xrightarrow{ab} l'$ *if and only if* $l \xrightarrow{ba} l'$, *and*
- *a* weak structural independence relation *if for all* $(a, b) \in I^S$ *locations* $l, l' \in L$, $l \xrightarrow{ba} l'$ *implies* $l \xrightarrow{ab} l'$.

When checking whether an event a is enabled, we consider a to have a set of *structural guards*, all of which need to be satisfied before a is enabled.

Definition 7. *Given a timed automaton* $M = (\Sigma, L, L^0, \delta, L^f)$, *a* structural guard *is a mapping* $g : L \to \{true, false\}$. *We denote the set of structural guards by* G^S. *The relation* $R^S \subseteq \Sigma \times G^S$ *is called a* structural guard relation, *if and only if 1)* $(a, g) \in R^S$ *and* $l \xrightarrow{a}$ *imply* $g(l) = true$, *and 2)* $l \not\xrightarrow{a}$ *implies* $\exists (a, g) \in R^S : g(l) = false$.

In a parallel composition of automata $M_1 \parallel \cdots \parallel M_n$, fix an event a. This a is *structurally enabled* in a location $l = (l_1, \ldots, l_n)$, if and only if for every i such that $a \in \Sigma_i$, $l_i \xrightarrow{a}$ in M_i. These conditions (one for each such i) can serve as structural guards. We can denote them g_i^a, i.e., $g_i^a(l) \Leftrightarrow l_i \xrightarrow{a}$.

The guard relation can be under-approximated, as long as for every disabled action we can find at least one unsatisfied guard.

We say that the event b *structurally enables* the guard g, if there is some l and l' such that $l \xrightarrow{b} l'$ and $g(l') = true$ and $g(l) = false$. A relation $E^S \subseteq G^S \times \Sigma$ is called a *structural enabling relation* if $(g, b) \in E^S$ whenever b structurally enables g.

In the context of a parallel composition we can look at the locations of component M_i. Let l_i, l_i' be locations such that $l_i \xrightarrow{b} l_i'$ and $l_i' \xrightarrow{a}$, then we would have $(g_i, b) \in E^S$.

The safe direction of approximating enabling relations is over-approximation. For instance if g is a guard of a then (g, b) holds for at least all those events that can locally lead to a state where a is enabled, but possibly others.

In Fig. 2, for instance, a would have a structural guard g_1, and any event that moves control on $\mathcal{M}_1$ to l_1, would enable g_1. The events a and b are structurally independent, but the figure demonstrates that this is not sufficient for reducing timed automata, as a and b are dependent in terms of time: After the event a, we have the zone indicated by $\phi_a \Leftrightarrow x_b - x_a \leq 2 \wedge x_a - x_b \leq 0$. After the event b, we have $\phi_b \Leftrightarrow x_b - x_a \leq 0 \wedge x_a - x_b \leq 3$. If we have no reason to know in which order a and b took place, we could merge the two zones into $\phi = \phi_a \vee \phi_b \Leftrightarrow x_b - x_a \leq 2 \wedge x_a - x_b \leq 3$.

Ideally we would like to have an abstraction that exactly removes such information. To achieve a more general theory, we will define independence relations for events, with respect to a given abstraction. The question of abstraction is deliberately left open, as it is relevant only with respect to a particular implementation.

Definition 8. *Given a timed automaton* $M = (\Sigma, L, L^0, \delta, L^f)$ *and an abstract transition relation* $\Longrightarrow_\alpha$. $I^T \subseteq \Sigma \times \Sigma$ *is a*

- strong temporal independence relation *under α , if for all $(a, b) \in I^T$, all clock constraints Z and for all transitions $\delta(l_a, a, g_a, c_a)$ and $\delta(l_b, b, g_b, c_b)$, $Z \models g_a$ and $Z \models g_b$ together imply that*
  1. *$sp(sp(Z, (a, g_a, c_a)), (b, g_b, c_b)) \subseteq POST_\alpha(POST_\alpha(Z, (b, g_b, c_b)), (a, g_a, c_a))$, and*
  2. *$sp(sp(Z, (b, g_b, c_b)), (a, g_a, c_a)) \subseteq POST_\alpha(POST_\alpha(Z, (a, g_a, c_a)), (b, g_b, c_b))$.*
- weak temporal independence relation *under α, if for all $(a, b) \in I^T$ and all clock constraints Z, $Z \models g_b$ and $sp(Z, (b, g_b, c_b)) \models g_a$ imply that*

$$sp(sp(Z, (b, g_b, c_b)), (a, g_a, c_a)) \subseteq POST_\alpha(POST_\alpha(Z, (a, g_a, c_a)), (b, g_b, c_b))$$

Strong temporal independence says that in any configuration, a and b can be executed in either order, and the resulting configuration can simulate all executions of the transition system under sp-semantics. Weak temporal independence promises that if a could be executed after b in the concrete system, the abstract system can execute a first and then b, and still simulate all the executions that were possible in the concrete system. For instance, if the constraint $x_b - x_a \leq 0$ in location (l_2, m_2) of Fig. 2(c) is replaced by $x_b - x_a \leq \infty$ then a is weakly temporally independent of b. Unless $x_a - x_b$ is not similarly relaxed, the converse does not hold, i.e., b is not weakly independent of a.

Theorem 1. *Let α_1 and α_2 be abstractions, such that $\alpha_1 \sqsubseteq \alpha_2$. If I^T is a strong (weak) temporal independence relation under α_1, then I^T is a strong (weak) temporal independence relation under α_2.*

Events have clock guards G_C, and these need to be taken into account in the reduction. We make no assumptions about the guards other than when an event is disabled due to time constraints, it has at least one (atomic) guard that is false.

Definition 9. *A relation $R^T \subseteq \Sigma \times G_C$ is a* time guard relation *if 1) $(b, g) \in R^T$ and $(l, Z) \overset{b}{\Longrightarrow}_{sp}$ imply that $Z \models g$, and 2) if $l \xrightarrow{b}$ and $(l, Z) \overset{b}{\not\Longrightarrow}_{sp}$ then $\exists g : (b, g) \in R^T \wedge Z \not\models g$. We say that the event $a \in \Sigma$ is* time enabling *for a guard g under α if there exists $(l, Z) \overset{a}{\Longrightarrow}_\alpha (l', Z')$ such that $Z \not\models g$ and $Z' \models g$. A relation $E^T \subseteq G_C \times \Sigma$, is a* time enabling *relation under α, if $(a, g) \in E^T$ if a is time enabling for g.*

In Fig. 2, a has the guard $x_b \leq 2$. If control is locally at l_1, but $x_b > 2$, then this guard is false. b is enabling for $x_b \leq 2$, because it resets x_b.

As with structural guards, the conservative approximation for a guard relation is an under approximation as long as the relation is non-empty. The conservative approximation for enabling is an over approximation. This is reflected in the definition by the fact that the time guard relation is defined in terms of sp-semantics and the enabling relation is defined in terms of abstract semantics.

In the following, let $G = G^S \cup G_C$, the set of all structural and clock guards.

Definition 10. *A relation $I_S \subseteq \Sigma \times \Sigma$ is a* strong independence relation*, if there exist a strong structural independence relation I^S and a strong temporal independence relation I^T such that $I_S = I^S \cap I^T$. A weak independence relation I_W is defined analogously. A relation $R \subseteq \Sigma \times G$ is a* guard relation *if there exist structural and time* *guard relations R^S and R^T such that $R = R^S \cup R^T$. A relation $E \subseteq G \times \Sigma$ is an enabling relation, if there exist structural and time enabling relations E^S and E^T such that $E = E^S \cup E^T$.*

Definition 11. *Let $(S, S_0, \Longrightarrow_\alpha)$ be the abstract transition system for the timed automaton $M = (\Sigma, L, S^0, \delta, L^f)$, and let I_S, I_W, E and R be the strong and weak independence, enabling, and guard relations under α, respectively, and let $(l, Z) \in S$, and let $G_{(l,Z)} = \{g \mid g \in G \wedge (l, Z) \not\models g\}$. let $U \subseteq \Sigma \cup G_{(l,Z)}$. Then U is a* Stubborn set *at (l, Z) if the following conditions hold:*

1. *$\forall a \in en(l, Z) \cap U : (\forall b \in \Sigma \setminus U : (a, b) \in I_S) \vee (\forall b \in \Sigma \setminus U : (a, b) \in I_W)$,*
2. *Either $en(l, Z) = \emptyset$ or $\exists a \in en(l, Z) \cap U : \forall b \in \Sigma \setminus U : (a, b) \in I_S$. When this condition holds for a, then a is called a* key event.
3. *$\forall a \in (\Sigma \setminus en(l, Z)) \cap U : \exists g \in G_{(l,Z)} : (g, a) \in R \wedge g \in U$.*
4. *$\forall g \in G_{(l,Z)} \cap U : \forall a : (a, g) \in E \Rightarrow a \in U$.*

Intuitively, a stubborn set contains events, and for computational convenience, also guards. Condition 1 states that each enabled stubborn event is either weakly independent of all non-stubborn events or strongly independent of all non-stubborn events. Condition 2 states that unless the current configuration is a deadlock, a stubborn set contains an enabled *key event*, which is strongly independent of all non-stubborn events, and as a consequence, non-stubborn events can never disable a key event. Condition 3 states that if a stubborn event is disabled, it has a guard that is inside the set, preventing it from becoming enabled. Condition 4 states that a guard of the set cannot be enabled by a non-stubborn event. In other words, conditions 3 and 4 work to guarantee that non-stubborn events alone cannot enable disabled stubborn events.

Stubborn sets can be easily calculated, for instance, using the modified deletion algorithm presented in [8]. We do not reproduce any algorithm here, as there are numerous algorithms in the literature.

Definition 12. *Let $M = (\Sigma, L, L^0, \delta, L^f)$ be a timed automaton, let $V \subseteq \Sigma$ be the set of visible events, and let $(S, S_0, \Longrightarrow_\alpha)$ be the abstract transition system for M. The reduction function $T : S \to 2^\Sigma$ is a* Stubborn set reduction function *if*

1. *$T(l, Z)$ is a stubborn set at every $(l, Z) \in S$.*
2. *If $a \in en_\alpha(l, Z)$, then there exists a sequence $(l_0, Z_0) \overset{b_1}{\Longrightarrow}_\alpha (l_1, Z_1) \overset{b_2}{\Longrightarrow}_\alpha \cdots \overset{b_k}{\Longrightarrow}_\alpha (l_k, Z_k)$ such that $(l_0, Z_0) = (l, Z)$, b_i is a key event for (l_{i-1}, Z_{i-1}), and $a \in T(l_k, Z_k)$.*
3. *If $V \cap T(l, Z) \cap en_\alpha(l, Z) \neq \emptyset$, then $V \subseteq T(l, Z)$.*

The conditions say: 1) the reduction function must produce a stubborn set, 2) if an action is ignored in a given state, it will be executed in some future state that is reachable using key events, and 3) if one of the enabled events in the stubborn set is visible, then all visible events must be included in the stubborn set. We reduce the abstract transition system, but unlike the usual reductions, our version of stubborn sets does not guarantee that non-emptiness of the abstract transition system is preserved. Instead, we prove only that if the *original system* contains counterexamples, then the reduced abstract transition system contains one.

Theorem 2. *Let $M = (\Sigma, L, L^0, \delta, L^f)$ be a timed automaton, Let $(S, S_0, \Longrightarrow_\alpha)$ be the abstract transition system for M. If T is a stubborn set reduction function for $(S, S_0, \Longrightarrow_\alpha)$, and $\mathcal{L}(M)$ is not empty, then the reduced abstract transition system $(S_T, S_0, \Longrightarrow_\alpha^T)$ has a counterexample.*

Proof. We prove a slightly stronger result, i.e., that if an arbitrary abstract configuration could reach an accepting location under strongest postcondition semantics, then the reduced system will reach one under the abstract semantics.

Let $(l, Z) \in S_T$ be arbitrary. Let β be a sequence of events such that $(l, Z) \xRightarrow{\beta}_{sp} (l^0, Z^0)$ is a minimal length execution to an accepting location l^0 under strongest postcondition semantics, with $|\beta| = n$. We show that there exists a state $(l', Z') \in S_T$, a location $l^1 \in L^f$ $Z^1 \neq \emptyset$, and a sequence of events ρ such that $(l', Z') \xRightarrow{\rho}_{sp} (l^1, Z^1)$, and $|\rho| < n$, which proves the claim by induction.

Let $\beta = b_1 \cdots b_n$, and let us denote $(l, Z) = (l_0, Z_0)$ and $(l_0, Z_0) \xRightarrow{b_1 \cdots b_i}_{sp} (l_i, Z_i)$ for the ith state in the sequence. Due to the minimality of n, no intermediate l_i is accepting, other than $l_n = l^0$. This means, that $b_n \in V$, by definition, as it leads from a non-accepting to an accepting location. The proof branches to two cases based on whether $\exists i : 1 \leq i \leq n \wedge b_i \in T(l, Z)$ holds or not.

As "case A", let us assume $b_i \in T(l, Z)$ for some i. Let $1 \leq i \leq n$ be minimal such that $b_i \in T(l, Z)$. Firstly, we prove $b_i \in en_\alpha(l, Z)$: If b_i is disabled, then there is some, either time or structural guard, that makes b_i disabled at (l, Z), say g, by point 3 of the definition of stubborn sets, and $g \in T(l, Z)$. Then, point 4 would guarantee, that any event that can cause g to become enabled would be in $T(l, Z)$, meaning, none of the b_j with $1 \leq j < i$ could enable it, as they are not in $T(l, Z)$, leading to a contradiction. To prove that b_i is also enabled in all the intermediate states before its appearance in the accepting sequence, notice that if b_i is strongly independent of all the b_j with $j < i$, none of them can disable b_i. If b_i is weakly independent of all b_j with $j < i$, then none of them can enable b_i. If b_i were disabled in some intermediate state, this would lead to a contradiction. We call this case A0.

When $i = 1$, A0 suffices as such. When $i > 1$, b_i, by property 1 of stubborn sets, is independent of b_j for $1 \leq j < i$, either weakly or strongly. By assumption, $(l_{i-2}, Z_{i-2}) \xRightarrow{b_{i-1} b_i}{}_{sp} (l_i, Z_i)$ holds and independence guarantees that $(l_{i-2}, Z_{i-2}) \xRightarrow{b_i b_{i-1}}{}_\alpha (l_i, Z_i^*)$ where $Z_i \subseteq Z_i^*$, which then implies $(l_i, Z_i^*) \xRightarrow{b_{i+1} \cdots b_n}{}_{sp} (l^0, Z^*)$ so that $Z^0 \subseteq Z^*$. Doing the same step i times, we permute b_i to (l_0, Z_0), and we get $(l, Z) \xRightarrow{b_i b_1}_\alpha (l_1', Z_1')$, and $(l_1', Z_1') \xRightarrow{b_2 \cdots b_{i-1}}{}_{sp} (l_i, Z_i^{**}) \xRightarrow{b_{i+1} \cdots b_n}{}_{sp} (l^0, Z^{**})$ so that $Z^0 \subseteq Z^{**}$.

Let us mark the sequence $b_2 \cdots b_{i-1} b_{i+1} \cdots b_n$ with β'. Therefore, we have $(l, Z) \xRightarrow{b_i}_\alpha (l', Z')$ so that $(l', Z') \in S_T$, and we have $(l', Z') \xRightarrow{b_1}_\alpha (l_1', Z_1')$, with $(l_1', Z_1') \xRightarrow{\beta'}_{sp} (l^0, Z'^0)$. We again have a branch, but with three cases. A1) If $b_1 \in T(l', Z')$ the claim is proven, as $(l_1', Z_1') \in S_T$, and $|\beta'| < n$. A2) If $b_j \in T(l', Z')$, with a similar deduction as before, we can find b_j that is, again, weakly (or strongly) independent of the events that precede it in β', and commute it, like before, so that $(l', Z') \xRightarrow{b_j}_\alpha (l'', Z'')$ so that $(l'', Z'') \xRightarrow{b_1}_\alpha (l_1'', Z_1'')$ and $(l_1'', Z_1'') \xRightarrow{\beta''}_{sp} (l^0, Z''^0)$, thereby shortening the distance by one, but otherwise like before: one abstract step, and then an actual counterexample. A2) can only repeat itself until the accepting state is just one abstract step away, otherwise it reduces to A0 or A3; we call case A3) the situation when none of the b_is of β' are in $T(l', Z'')$.

We merge cases B and A3, because they are similar. Let $(l, Z) \xRightarrow{b_1}_x \xRightarrow{b_2 \cdots b_n}_{sp} (l^0, Z^0)$ so that x is either sp or α, and $b_i \notin T(l, Z)$, for $1 \leq i \leq n$. Note that $(l, Z) \xRightarrow{b_i}_{sp}$ implies that $(l, Z) \xRightarrow{b_i}_{\alpha}$, so that at the least $b_1 \in en_\alpha(l, Z)$ holds. We mark the intermediate states on this path with superscripts indicating the number of steps remaining to (l^0, Z^0), so that $(l, Z) = (l^n, Z^n)$.

Stubborn set reduction function property 2 guarantees the existence of sequence of key events $a_1, \ldots, a_k$ with $k \geq 0$, such that $(l, Z) = (l_0, Z_0)$ and $(l_0, Z_0) \xRightarrow{a_1 \cdots a_k}_{\alpha} (l_k, Z_k)$. These subscripts are not to be confused with the notation in the A-case. We mark the intermediate states (l_i, Z_i). On this path – which in its entirety is in S_T – there is a state (l_i, Z_i) for which one of b_js is in $T(l_i, Z_i)$. At the very least, (l_k, Z_k) is such a state, as per property 2 of stubborn set reduction functions.

Let us choose the minimum such i. We must then show that $(l_i, Z_i) \xRightarrow{b_1}_x \xRightarrow{b_2 \cdots b_n}_{sp} (l_i^0, Z_i^0)$, so that l_i^0 is an accepting location; once this is proven, again, the property reduces to one of the cases A0 to A2.

Suppose this property holds for (l_j, Z_j) with $j < i$. It then holds for $j = 0$, as assumed in this case. $(l_{i-1}, Z_{i-1}) \xRightarrow{a_i}_{\alpha} (l_i, Z_i)$ is a key event, and $(l_{i-1}, Z_{i-1}) \xRightarrow{b_1}_x (l_{i-1}^{n-1}, Z_{i-1}^{n-1}) \xRightarrow{b_2 \cdots b_n}_{sp} (l_{i-1}^0, Z_{i-1}^0)$. Because a_i is a key event, it is strongly independent of all b_i, which (almost) gives us the result, so that $x = \alpha$ at (l_i, Z_i).

To show that $l_i^0 \in L^f$, inductive hypothesis gives us $l_{i-1}^0 \in L^f$. Structural strong independence gives us $l_{i-1}^0 \xrightarrow{a_i} l_i^0$, and if $l_i^0 \notin L^f$, then $a_i \in V$ must hold. Bearing in mind that $b_n \in V$, this would contradict either point 3 of stubborn set reduction function or the assumption that none of the b_i is in $T(l_j, Z_j)$ for $j < i$. □

3.2 Ignoring Problem and Key Events

Property 2 of the stubborn set reduction function in Definition 12 is intended to solve the *ignoring problem* [16]. Previously suggested solutions for the ignoring problem include techniques based on strongly connected components [16] and complex conditions that deal with on-stack states [7].

The fact that key events and other events need to be considered separately further complicates the matter. One solution for this problem was given in [8], in the context of the Tarjan algorithm, but here we discuss implementation details for algorithms that do not need to detect strong components.

Let us re-iterate Property 2 from the point of view of a search algorithm that explores the reduced state space: given a state s, with $en(s)$ as the set of enabled events, and $T(s)$ as the set of stubborn events, property 2 says that for every $a \in en(s) \setminus T(s)$, there must be some state s^*, reachable from s using key events, so that $a \in T(s^*)$.

Consider a usual depth-first search, which maintains a stack of states Q (along with other necessary information). Let us assume the top state of the stack is currently s. We can store a bitset of *satisfied* events, denoted $sat(s)$ for every state in the stack.

$a \in sat(s)$ means that we know there is a sequence of key events from s to some state s' such that $a \in T(s')$. Obviously $sat(s) = T(s)$ when state s is initially put on the stack and $T(s)$ calculated.

When we are about to backtrack from s, we check that $en(s) \subseteq sat(s)$; if not, then more events need to be explored. In out test implementation we fully expand the state s, which satisfies the condition trivially, but extending the stubborn set so that at least one new key event gets added would also be correct and potentially result in more reduction.

On the other hand, when we backtrack from s to some state s', this means that $s' \xrightarrow{a} s$ for some $a \in T(s')$. If a is a key event, then all the events that were satisfied in s, are also satisfied in s', so we can set $sat(s') = sat(s') \cup sat(s)$; we say that s is a *key-successor* of s'.

This concept points to alternatives that work for searches other than depth-first search. In a state s, with $T(s)$ as the stubborn set, we propagate information *forward* to one of the key-successors of the current state. The events in $en(s) \setminus T(s)$ need to be satisfied by one key-successor. If in a given state s, the key-successors are all old states, one calculates a larger $T(s)$ until either $T(s) = en(s)$ or an unexplored key-successor is generated. We did not experiment with this solution, as dept-first search makes is easier to extract counterexamples. We leave exploring such solutions for future work.

3.3 Abstraction-Refinement and Independence

An *abstraction refinement* loop in general works by successively refining abstraction until non-emptiness has been decided by either finding a concrete counterexample or an empty abstract transition system. The loop starts with the loosest abstraction, which in our implementation means omitting diagonal timing constraints altogether. In every iteration, we calculate the dependency relations with respect to the current abstraction α.

The abstract transition system is then checked for counterexamples; because we need counterexamples, we used depth-first search in our implementation, with the ignoring conditions as described in the previous subsection.

If no counterexample is found, the system is correct, due to Proposition 1 and Theorems 1 and 2. If we find a counterexample, it is a guarded word that leads to an accepting location in the abstract transition system. We then try to simulate the word using strongest postcondition semantics. If a simulation leads to an accepting location, we have found an actual counterexample.

If the counterexample cannot be replicated, all simulations (the system may be nondeterministic) lead to non-accepting locations or end in empty zones before they end. In this case we tighten the abstraction by considering more timing constraints. The exact details depend on the family of abstractions used, and we will discuss only one example in this section.

The particular abstraction is merely an example. Any abstraction or family of abstractions will work as long as we can calculate independence relations that satisfy Definition 8. Also, any abstraction technique that makes each abstraction more coarse, can be combined with our method, due to Theorem 1.

The example implementation uses an abstraction which we call *pairwise dependence of clocks* (PDC), in combination with LU-simulation [3,10]. The abstraction is implemented by partitioning the clocks into dependency classes. The diagonal constraints between clocks of different classes are omitted. Let $D_\alpha \subseteq C \times C$ be an equivalence relation for clocks. When calculating the post-condition of an event, diagonal constraints of the form $x - y \prec n$ are only considered when $(x, y) \in D_\alpha$, otherwise n is considered to be ∞.

The $POST_\alpha$-operation with respect to D_α is defined as follows. Time zones in the abstract transition system are given by the canonical constraints where $Can_\alpha(Z)$ is of the form $\bigwedge_{(x,y) \in D^0_\alpha} x - y \prec_{xy} n_{xy}$, where $(x, y) \in D^0_\alpha$ if $(x, y) \in D_\alpha$ or if either a of b is the dummy clock x_0 which is always 0.

Any independence relation for the events must meet the criteria of Definition 8 under $POST_\alpha$ to be valid. We propose the following: Let $C(a) = \mathcal{R}(a) \cup \mathcal{G}(a)$, and define temporal dependency relations using the following checklist:

1. If there are clocks x, y such that $x \in \mathcal{R}(a)$ and $y \in \mathcal{R}(b)$ and $(x, y) \in D_\alpha$ then a and b are dependent (both weakly and strongly).
2. If for all clocks x, y: $x \in C(a)$ and $y \in C(b)$ implies that $(x, y) \notin D_\alpha$, then $(a, b) \in I^T_S$, and symmetrically for (b, a). Intuitively, if the events do not share any dependent clocks, they are strongly (and weakly) independent under D_α.
3. If for every $x \in C(a)$ and $y \in C(b)$ such that $(x, y) \in D_\alpha$, the guard of a contains no lower bounds for x, then a is weakly independent of b.
4. If also in the previous case the guard of b contains no lower bounds for y, then a is strongly independent of b.

Lemma 1. *The relations described above are valid temporal independence relations for the abstract transition system under PDC-abstraction.*

4 Experiments

We created an implementation[1] of our method in the PAT framework [15]. The main question to answer is whether and how much the method is able to reduce, and whether the benefits (in reduced states) outweight the cost (in overhead in calculating the sets). Our implementation was an iterative version of the deletion algorithm [8], with optimizations that aim at faster calculation.

We measured the performance of a direct verification of the zone graph, using LU-simulation alone (with BFS), LU simulation and abstraction refinement that uses our stubborn sets, and for comparison, LU- simulation with abstraction refinement but without reduction. Our implementation of abstraction was the PDC-abstraction explained in Section 3.3, and LU-simulation was calculated on top of that. Structural relations were analyzed by examining the control structure of component automata and using simple heuristics for shared variable access, such as write/write and read/write of a shared variable. The algorithm for state exploration for the two AR implementations was a depth-first search, due to the need for counterexamples.

For reference, we did the tests also with UPPAAL on the same models; the models may not produce exactly the same number of states, as it is possible that there are small

[1] See `https://sites.google.com/site/shangweilin/timedpor` for additional updates on performance.

Table 1. Verification results

model	PAT/BFS+LU		PAT/AR+LU+POR		PAT/AR+LU		UPPAAL BFS	
	$\|S\|$	$time$	$\|S\|$	$time$	$\|S\|$	$time$	$\|S\|$	$time$
CSMACD 5	2705	0,18	**1131**	0,17	2942	0,19	2156	**0.03**
CSMACD 6	12355	0,25	**3488**	0,21	11585	0,79	8976	**0.08**
CSMACD 7	54522	1	**10146**	0,7	44349	3	35739	**0.36**
CSMACD 8	234600	7	**28272**	2	164257	17	137678	**1**
CSMACD 9	991483	40	**76185**	7	592113	78	516751	**6**
CSMACD 10	4139285	232	**199804**	**21**	O/M		1899028	28
CSMACD 11	O/M		**512344**	**62**	O/M		6857723	117
FDDI 5	459	0,02	**41**	**0**	**41**	0,35	286	**0**
FDDI 10	10637	1	**81**	**0,02**	**81**	0,04	6043	0.19
FDDI 15	O/M		**121**	**0,04**	**121**	0,06	105990	34
FDDI 20	O/M		**161**	**0,1**	**161**	0,1	O/M	
Fischer 5	3277	0,04	**807**	0,07	5785	0,38	2958	**0.02**
Fischer 6	15229	0,19	**2570**	0,27	20470	1	12777	**0.08**
Fischer 7	69602	1	**8185**	1	115633	7	54372	**0.42**
Fischer 8	313421	6	**26104**	3	578311	47	229559	**2**
Fischer 9	1393599	37	**83339**	14	O/M		965466	**12**
Fischer 10	6131109	242	**266118**	**56**	O/M		4053445	62
Fischer 11	O/M		**849213**	**220**	O/M		17005200	315
Railways 5	34197	0,7	**1587**	0,16	19217	1	16726	**0.09**
Railways 6	465634	10	**9572**	**0,96**	230714	14	200821	1
Railways 7	7250443	302	**67069**	**7**	O/M		2811642	22

differences in the models, and also, because the optimizations of UPPAAL are different from our implementation. The idea is to give some indication of scalability issues.

Our experiment set consisted of some well-known safe examples, CSMA/CD networking, Fiber distributed data interface (FDDI), the famous Fischer protocol, and a railway controller protocol. We measured the total number of generated configurations and time in seconds. We ran the experiments on a PC with an Intel Core-i7, 3.4GHz and 8GB of RAM. Running times should be taken only to indicate order of magnitude and scalability, because during the tests computer load and similar factors cause substantial variation in running times.

The results of our experiments are given in Table 1. The best performances in terms of number of states generated and execution time are indicated with boldface characters. The results under reduction are given in the second column, and in all the cases, no other approach generated fewer states.

The effects of reduction on scalability mean that eventually it is superior to every other solution in our tests. Comparing execution times, we notice that our method slows state generation down by a significant factor. However, this is more than compensated by the effect on scalability of larger models. Another observation from the first and third columns is that the abstraction refinement implementation without partial order reduction also slows down state generation significantly; It is plausible that our implementation of the PDC-abstraction is far from optimal and the time of actually calculating the abstract successors dominates the execution times.

Some of the reduction in the number of states comes from abstraction itself, (in FDDI, all of it) but for instance, in the Fischer model PDC abstraction actually makes the state space bigger, but when reduction is used, the state space is greatly reduced. UPPAAL was chosen as a reference, as it can be viewed as the gold-standard for timed verification. It performs significantly better than PAT when reduction is not used. UPPAAL also clearly has the advantage that it seems to generate states much faster. However, despite this handicap, our partial order reduction eventually beats even UPPAAL, not only in terms of states, but also in verification time, when the models get large enough.

5 Conclusion

We defined a variant of the stubborn set method for timed verification, which makes use of abstraction. The method uses dependence and independence defined in terms of concrete behaviors that the abstract system must preserve instead of directly defining them on the abstract zone graph. We believe the method overcomes a fundamental hurdle for commutativity based reduction in real time verification, that of clocks causing superfluous dependency. To the best of our knowledge, this is the first successful application of the "standard" partial order reduction methods on timed automata.

In our measurements, our method was able to provide outstanding reduction, but naturally, it can only reduce models that exhibit a high degree of concurrency and interleaving. The theory is general and works with any abstract semantics as long as sufficient conditions for weak and strong (temporal) independence can be extracted. Even the simple heuristics in our reference implementation turned out to be very efficient in reducing the number of states explored during verification of some models.

Acknowledgement. We would like to thank all the anonymous reviewers for helpful and insightful comments that have helped us improve this paper. This research was partly supported by project "IDD11100102" from SUTD, "Formal Verification on Cloud" project under Grant No: M4081155.020, and TRF project "Research and Development in the Formal Verification of System Design and Implementation"

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., Itai, A., Kurshan, R.P., Yannakakis, M.: Timing verification by successive approximation. Inf. Comput. 118(1), 142–157 (1995)
3. Behrmann, G., Bouyer, P., Larsen, K., Pelanek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer (STTT) 8, 204–215 (2006)
4. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)
5. Dams, D., Gerth, R., Knaack, B., Kuiper, R.: Partial-order reduction techniques for real-time model checking. Formal Aspects of Computing 10, 469–482 (1998)

6. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 313–329. Springer, Heidelberg (1998)
7. Evangelista, S., Pajault, C.: Solving the ignoring problem for partial order reduction. International Journal on Software Tools for Technology Transfer 12(2), 155–170 (2010)
8. Hansen, H., Kwiatkowska, M., Qu, H.: Partial order reduction for model checking markov decision processes under unconditional fairness. In: QEST 2011, pp. 203–212. IEEE CS Press (2011)
9. Hansen, H., Wang, X.: Compositional analysis for weak stubborn sets. In: Caillaud, K.H.B., Carmona, J. (eds.) Proceedings of ACSD 2011, pp. 36–43. IEEE CS Press (2011)
10. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Better Abstractions for Timed Automata. In: LICS, pp. 375–384 (2012)
11. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. Theoretical Computer Science 345(1), 27–59 (2005)
12. Minea, M.: Partial order reduction for model checking of timed automata. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 431–446. Springer, Heidelberg (1999)
13. Niebert, P., Qu, H.: Adding invariants to event zone automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 290–305. Springer, Heidelberg (2006)
14. Salah, R., Bozga, M., Maler, O.: On interleaving in timed automata. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 465–476. Springer, Heidelberg (2006)
15. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards Flexible Verification under Fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009)
16. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design 1(1), 297–322 (1992)
17. Valmari, A.: Stubborn set methods for process algebras. In: Proceedings of the DIMACS Workshop on Partial Order Methods in Verification, POMIV 1996, pp. 213–231. AMS Press, Inc., New York (1997)