An efficient algorithm for minimizing real-time transition systems

(Extended Abstract)

Mihalis Yannakakis and David Lee

AT&T Bell Laboratories Murray Hill, NJ 07974, USA

1 Introduction

Time plays an important role in the operation and correct functioning of many systems. The last few years there has been significant progress in the modeling and formal analysis of such systems that incorporate real time. An attractive model for real time systems that is being extensively investigated is the *timed automaton* (or timed graph) model [4, 14]. This is an extension of the familiar finite automaton (state machine) model by a finite set of *clock variables*. The transitions of the automaton may depend on the values of the clocks and can have the effect of resetting some of the clocks. The values of the clock variables increase continuously and spontaneously with the passage of time. The model has provided the foundation for studying a number of issues concerning real time systems, including the computation of reachability information among the states of the system, providing bounds on the delays of events, the verification of temporal logic properties, the development of "timed" languages, and others [1, 4, 9, 13, 20].

The global state of a timed system, called for clarity a *configuration* here, consists of the control state of the timed automaton and the values of the clocks. Thus, there is an infinite (in fact, uncountable) number of configurations. The main algorithmic tool that permits the finite analysis of such systems is the partition of the configuration space into a finite number of *regions* and the construction of an associated *region graph* [4]. Suppose that we can observe the transitions of the system. Configurations in the same region have the property that they are indistinguishable in terms of the future sequences of transitions that can occur. Thus, they all have the same reachability properties and can be collapsed yielding a graph on the regions.

The main problem is that the region graph has size exponential in the number of clocks and the size of the constants that appear in the enabling conditions of the transitions (i.e., their length when written in binary). There are PSPACE-completeness results [4, 13] indicating that in the worst case one cannot avoid incurring exponential complexity. However, this does not mean that it has to occur in every case. In particular, some configurations from different regions may be also indistinguishable. After merging these regions it is possible that we obtain a much smaller minimized graph. Usually we have an initial configuration and are only interested in the portion of the system that is reachable from it. Thus, we wish to construct the relevant *minimal reachable graph* of the system. The

PSPACE-completeness results imply that there are cases where even this graph has exponential size. However, there are many cases where it is considerably smaller, and in these cases we would like to construct the graph directly without going first through the region graph. To take full advantage of this, it is desirable to construct the minimal reachable graph in time that is a low order polynomial function of its size.

Algorithms for performing simultaneously reachability analysis and minimization in general transition systems were proposed recently in [7, 6] and [19]. Alur et al. adapted the first method to timed transition systems and showed how the approach can be utilized for model checking of temporal formulas [2, 3]. Here we shall develop a minimization algorithm based on the second method. We address the following problem. We are given a timed automaton G, an initial configuration p_0, and an initial partition π of the configurations into blocks that respects the transitions of G; i.e., all configurations of a block have the same state component and the same enabled (immediate) transitions. We assume that every block is specified by a set of inequality constraints that place upper or lower bounds on the values of the clocks and their differences. We shall develop an algorithm that conctructs the minimal reachable graph G_m induced by the partition π in time polynomial in the size of the graph G_m.

The general algorithm of [19] is developed at an abstract level in terms of abstract representations for classes of configurations and basic set operations on them such as set intersection, difference etc. The complexity of the algorithm is measured in terms of the number of these operations that are performed. The complexity is polynomial in the size of the final minimal reachable graph G_m, provided we have also an efficient "termination" routine that checks the answer, i.e., whether a given graph is equal to G_m. To apply the general algorithm to a concrete context, such as the model of timed transition systems in our case, we have to implement efficiently the basic operations and address the termination problem. The main technical obstacle is presented by the difference operation: the difference of two conjunctions of constraints (eg. convex polyhedra) is not a conjunction (it is in general nonconvex). We develop a method for circumventing this obstacle, staying with conjunctive explicit representations while maintaining the efficiency of the algorithm. The method does not depend on any special properties of real time transition systems and applies to any deterministic systems.

The rest of this paper is organized as follows. In Section 2 we review the background on general transition systems and timed graphs in particular. In Section 3 we show some properties of the minimized system. In Section 4 we describe our algorithm for minimizing real-time transition systems and in Section 5 we discuss the implementation of the basic operations. Section 6 addresses the termination problem.

2 Preliminaries

We will review first basic notations and definitions about general transition systems, and then discuss in particular the timed systems.

2.1 Transition systems and minimization

A (initialized) *transition system* is a tuple $\Theta = (Q, I, T, p_0, \pi)$ consisting of (1) a set Q of *configurations* (sometimes called system states or points); (2) a finite set I of actions (or inputs); (3) a set T of transition relations on Q corresponding to the actions, i.e., for each action $a \in Q$ there is a relation $R_a \subseteq Q \times Q$; (4) an initial configuration p_0; (5) an initial partition π of Q. The transition relation is deterministic if the transition relation is a (partial) function, otherwise it is nondeterministic.

In the above definition we have included both the initial configuration and the initial partition. The set of configurations may be infinite. We think of a transition system as a graph with set of nodes Q and with an arc from node p to node q labelled by an action a if $(p, q) \in R_a$. A finite sequence of actions $\alpha \in I^*$ corresponds to the relation R_α formed by composing the relations R_a, $a \in \alpha$. If S is a set of configurations, we use the notation $\alpha(S)$ for $\{q | \exists p \in S.(p, q) \in R_\alpha\}$ and $\alpha^{-1}(S)$ for $\{p | \exists q \in S.(p, q) \in R_\alpha\}$.

The transition system Θ induces a *quotient* transition system Θ/π whose configurations are the blocks of π. The quotient graph has the blocks of π as its nodes and has an arc from block B to block C labelled by action a if $(p, q) \in R_a$ for some $p \in B$ and $q \in C$. We say that arc $B \rightarrow C$ is *stable* if every configuration of B has an a-transition to some configuration of C. A block B is stable if all its arcs are; equivalently, for every block C and every action a, either $B \cap a^{-1}(C) = B$ or $B \cap a^{-1}(C) = \emptyset$. The partition π is stable if all its blocks are stable.

Given any transition system $\Theta = (Q, I, T, p_0, \pi)$ there is a unique coarsest stable partition π' that refines π (i.e., each block of π' is a subset of a block of π). Two configurations of Θ are *equivalent* if they belong to the same block of π'. The *minimized* (or *reduced*) transition system of Θ is the system $\Theta' = (Q, I, T, p_0, \pi')$. We will usually identify the minimized system Θ' with its quotient Θ'/π'. We usually only care about the configurations of Θ that are reachable from p_0. The blocks of π' containing them are exactly the blocks that are reachable in Θ'/π' from the initial block (i.e., the one containing p_0). The subgraph of Θ'/π' that is induced by these nodes is called the *minimal reachable graph*.

2.2 Timed graphs

Our model of a real-time system is a *timed graph* (or timed automaton) as in [1, 4]. A timed graph has a finite set S of n control states (the nodes); a finite set C of k *clocks* (or timers) which are nonnegative real-valued variables; and a finite set of arcs E representing the transitions of the system, where each arc a is labelled by an enabling condition K_a on the values of the clocks, and by a (possibly empty) subset C_a of the clocks which are to be reset to 0

when the transition takes place. The enabling conditions are finite conjunctions of inequalities comparing a clock or the difference of two clocks to an integer constant; i.e., every inequality is of the form $x_i \theta c$ or $(x_i - x_j)\theta c$, where x_i, x_j are clocks, c is an integer, and θ is a comparison operator $<, \leq, \geq$ or $>$. The set of nonnegative solutions to a set of such inequalities forms a convex polyhedron, which [3] calls a *zone*; we will identify an enabling condition K_a with the corresponding zone.

A timed graph G is a compact representation of the following transition system Θ. The configurations of Θ are pairs (s, v) consisting of a control state s of G and an assignment v of (nonnegative) real values to the clocks of G. The initial configuration consists of a given initial state s_0 of G and the all 0 assignment to the clocks, (or some other given integral initial assignment v_0). The transitions of Θ correspond either to explicit transitions of G or to implicit transitions due to the passage of time; i.e., Θ has one action a for each arc a of the timed graph G, and one additional action, denoted *time*. Suppose that G contains an arc $a = s \rightarrow s'$ labelled with a condition K_a and a set of clocks C_a that are to be reset. If (s, v) is a configuration such that v satisfies the condition K_a, then Θ has an arc labelled a from (s, v) to (s', v') where the assignment v' agrees with v in all the clocks except for the ones in C_a which are 0. All the clocks proceed at the same rate and measure time since they were last reset, or initialized. Formally, for every configuration (s, v) and every constant $\delta \geq 0$ there is an arc from (s, v) to (s, w) labelled by the action *time*, where $w = v + \delta$ is the assignment obtained from v by adding δ to all the clocks.

Finally, we assume an initial partition π where each block of π is of the form $\{s\} \times Z$ where s is a control state of the timed graph G, the set Z is a zone (represented in terms of a set of inequality constraints as above), and all configurations of the block have the same enabled transitions out of s; i.e., for all arcs a coming out of s we have $Z \cap K_a = Z$ or $\emptyset$.[1] We will often use (s, Z) instead of $\{s\} \times Z$ and use the term zone also for the block itself.

The problem we address is the following. Given a timed graph G, an initial configuration (s_0, v_0) and partition π as above, we wish to compute the minimal reachable graph of the associated transition system Θ. We use G_m to denote the minimal reachable graph.

Although Θ is infinite, the main result of [4] implies that the minimized system has only a finite number of blocks. Let b be the largest constant that appears in an enabling condition of G (and the initial partition π). Let ρ be the partition where two configurations (s, v) and (s', v') are in the same block iff they have the same state component $s = s'$ and their clock assignments v, v'

[1] Alur et al.[3] start their algorithm with an initial partition that has for each state s of G, only one block (s, R^k) containing all configurations with state component s. However, as soon as the block is reached, it is split into zones with the above property, respecting the conditions on the arcs out of s. We prefer to include this initial splitting in the initial partition to make it explicit in the statement of the problem that is being solved. Once this is done, the blocks of the desired minimized system are zones, they are uniquely determined and there is no need for nondeterminism.

satisfy exactly the same inequalities of the form $x_i \theta c$ or $(x_i - x_j)\theta c$, where x_i, x_j are clocks and c is an integer that does not exceed b. Clearly, ρ is a finite partition. It has $O(k!nb^k)$ blocks, called *regions*. Every region is a zone, i.e. can be specified by inequalities of the appropriate form. Furthermore, ρ refines π and is stable [4]. The quotient graph Θ/ρ is called the *region graph*. Thus, the region graph is in general a refinement of the minimal graph. However, it may be a proper refinement and it may be much larger than the minimal graph. In view of the exponential size of the region graph, it is desirable to construct directly the minimal reachable graph G_m, and furthermore, to construct it in time that is not much larger than its size, so that we can take advantage of the cases where G_m is small. That is, if G_m has N nodes, we want to spend time that is a low polynomial function of N when $N \ll k!nb^k$.

3 Properties of the minimal system

Let G be a timed graph, and π the initial partition as defined in Section 2. Let Θ be the corresponding timed transition system, π' the coarsest stable refinement of π.

The transition system Θ is "almost" deterministic: For every configuration (s, v) and every transition a of G out of state s, there is at most one a-arc in Θ out of (s, v). The only source of nondeterminism is the *time* action. Clearly, the subgraph of Θ induced by these edges is acyclic (except for self-loops corresponding to the passage of 0 time; we will ignore *time* self-loops in the following). Furthermore, the same is true of the subgraph of Θ/π, because all the blocks are convex. Consider a configuration $(s, v) \in B$ and the "time trajectory" $(s, v + \delta)$ as δ increases from 0 to infinity. If the trajectory exits B, let B' be the first block that it hits; we say then that B' is the *immediate time successor* of (s, v).

Lemma 1. *1. Either all time trajectories starting in configurations of B exit B or they all stay in B.*

2. If two configurations of B have time arcs to the same blocks then they have the same immediate time successor.

Proof. (1) The basic reason is that B is convex, in fact a zone. First note that constraints of the type $(x_i - x_j)\theta c$ remain invariant under the *time* action. On the other hand, if the inequalities describing B contain an upper bound constraint $x_i < c$ or $x_i \leq c$ for some clock x_i then all trajectories starting at configurations of B will exit the block; if they do not, then all trajectories will stay in B.

(2) Suppose that the immediate time successor of configuration (s, v) is block C_1 and that of (s', v') is a different block C_2, and both blocks have time arcs from both configurations. It is easy to see that this contradicts the fact that C_1 and C_2 are convex and disjoint. □

Consider a variant $\overline{\Theta}/\pi$ of the quotient system (and graph) Θ/π where instead of the *time* action (and arcs) we have another action t standing for "immediate time successor". There is an arc labelled t from block B to block C if

some configuration of B has block C as its immediate time successor. We let $B \cap t^{-1}(C)$ denote the set of configurations of B with immediate time successor C. For zones B and C, the set $B \cap t^{-1}(C)$ is also a zone. It is possible that $B = B \cap t^{-1}(C)$, i.e., B is stable with respect to t, yet B is not stable with respect to the *time* transitions. However, it is easy to see that all time successor blocks of B are stable with respect to the *time* action if and only if they are stable with respect to t. Thus:

Lemma 2. *The system Θ/π has all the time arcs stable iff the system $\overline{\Theta}/\pi$ has all the t-arcs stable.*

The minimal graph $\overline{\Theta}/\pi'$ is also deterministic with respect to t (besides the ordinary, explicit transitions). Let π' be the partition of the minimized system.

Theorem 3. *Every block of π' is a zone.*

Proof. We can form the minimized system by iteratively refining the partition splitting unstable blocks. For the purposes of the proof, we do not need to worry about efficiency. The class of zones is closed under the intersection operator and inverse image a^{-1}, but not under the difference operator. We do the refinement as follows so that we avoid the difference operator, and at all times we have a partition whose blocks are zones.

Suppose that there is a block B that has at least two a-arcs for some transition a of the timed graph. Let $C_1, \ldots, C_r$ be the blocks that intersect $a(B)$. Replace the block B in the partition by the blocks $B \cap a^{-1}(C_1), \ldots, B \cap a^{-1}(C_r)$. Note that these sets are disjoint and their union is equal to B because all configurations of B have the same enabled transitions, thus they all have exactly one a-transition. Assuming inductively that B and the C_i's are zones, the same is true of the sets $B \cap a^{-1}(C_i)$.

Suppose that the current partition is not stable with respect to the *time* edges. Then it is not stable either with respect to the immediate time t edges. There is a block B with at least two t-arcs. Let $C_1, \ldots, C_r$ be the blocks that have t-arcs from B. Replace B by the blocks $B \cap t^{-1}(C_1), \ldots, B \cap t^{-1}(C_r)$. Note again that these sets are zones, are disjoint and their union is equal to B. □

The reason that it is important to restrict the blocks to being zones is the fact that they have a succinct representation, which does not grow as the refinement progresses. When we only care about the reachable portion of the minimized system, it requires more care to avoid the difference operator, while guarranteeing time complexity that is polynomial in the size N of the reachable portion.

4 The minimization algorithm

Let G be a timed graph, (s_0, v_0) the initial configuration, and π the initial partition as defined in Section 2. Let Θ be the corresponding timed transition system, π' the coarsest stable refinement of π. We wish to construct its reachable part, i.e., the minimal reachable graph G_m.

General-purpose algorithms for constructing the minimal reachable graph of a transition system are proposed in [7](see also [6]) and [19]. Both methods try to combine searching of the graph, i.e., the forward inference of reachability information, with splitting of unstable blocks, the backward inference of inequivalence information. They both use essentially similar abstract symbolic operations on blocks, such as set interection, difference, inverse image etc. They differ primarily in the order in which they search and split blocks; briefly, the first method gives priority to splitting rather than searching, while the second one does the opposite and also it splits unstable blocks in a fair manner (FIFO order). Alur et al. [2, 3] have applied the first method to the minimization of timed transition systems. In general, the complexity of the algorithm is exponential in the size of the output. For instance, one can adapt an example given in [19] (for the model of affine transitions) to construct a timed graph with two clocks and n states, and with $O(n)$ arcs and blocks in the initial partition, such that the minimal reachable graph has $O(n)$ nodes, but the algorithm performs 2^n splits, and hence takes 2^n time (regardless of the implementation of the operations). We defer the example to the full paper.

We shall develop an algorithm based on the method of [19] with the goal of providing guarrantees on its time complexity, measured as a function of both the input and the output. The algorithm in [19] was analysed at an abstract level, i.e., in terms of the number of basic set operations that it performs. It was shown that if the minimal reachable graph G_m has N nodes and M arcs, then the algorithm will construct the graph G_m after $O(NM)$ basic operations. However, at this point the algorithm may not know that it has constructed the right graph, the blocks may not be stable yet and the algorithm may keep refining them (but of course no more than the minimal system). If we have a "termination" routine of complexity $q(N, M)$ which determines whether a given graph H is actually the desired reachable minimal graph G_m (or just tells us if it has the correct number of nodes), i.e., solves the easier problem of checking the answer (as opposed to constructing it), then the whole construction takes $O(NM + q(N, M))$ steps.

When we want to specialize the algorithm to a concrete context, as we wish to do here in the case of timed transition systems, we need to choose a representation for the blocks, implement the basic operations, and address the termination problem. Once we have done this, then we can talk about the concrete complexity of the algorithm in the usual models of computation. Note that if we cannot solve the termination problem in polynomial time, we obviously cannot hope to construct the minimal reachable graph in time polynomial in its size.

We describe the algorithm first in general terms, and then we shall discuss the implementation choices. At all times we maintain a partition, which is initially π. A subset of the blocks are *marked.* These are the blocks that we have discovered so far to be reachable. Initially only the block containing the initial configuration is marked. Every marked block B is of the form (s_B, Q_B) where s_B is a state of the timed graph and Q_B is a set of clock assignments. For every marked block B we have a marked configuration $p_B = (s_B, v_B)$, one that we know is reachable

from the initial configuration. Once a configuration gets marked it never gets unmarked; i.e., *marked configurations do not get updated.*

We maintain a marked graph H whose nodes are the marked blocks. Every marked block B has exactly one arc for each transition a that is enabled in its configurations; the arc is directed into the block that contains the image $a(p_B)$ of its marked configuration p_B. Also, B has an arc labelled t to the block containing the immediate time successor of p_B (if there is one). Note that we do not include all arcs from the marked blocks; only the arcs of the marked configurations. As a consequence, every node of the marked graph has "small" degree: at most 1 more than the degree of the corresponding state in the timed graph. Also, *unmarked blocks do not have any arcs.* At the end of the algorithm, the marked graph coincides with the minimal reachable graph G_m. Obviously, the marked graph is at all times no larger than the final graph G_m.

The algorithm is conceptually simple. It starts by marking the initial configuration (s_0, v_0) and the block of π that contains it. Throughout the execution, it gives preference to searching forward rather than splitting blocks to stabilize them. The algorithm explores marked configurations by examining their transitions to find new reachable blocks to mark. At any point in time, if there is an unexplored marked configuration $p_B = (s_B, v_B)$, we find the blocks containing its images $a(p_B)$ under all transitions a, including the "immediate time" t-transition. If any of these blocks is not marked, say block C containing $a(p_B)$, then we mark the block C and let its marked configuration be $p_C = a(p_B)$ (if a is the t-transition, we may choose any configuration $p_C = (s_B, v_B + \delta)$ of C that is obtained from p_B by advancing the clocks by an equal amount).

Suppose there are no unexplored marked configurations. If all arcs of the marked graph H are stable, then H is the minimal reachable graph and the algorithm terminates. Otherwise, some marked blocks are unstable and have to be split. Such blocks are split in a round-robin order; there is a *queue* containing marked blocks that may be unstable. We remove the first block $B = (s_B, Q_B)$ from the queue and split it *into two parts*, using the marked configuration p_B as the guideline: The first part B', which becomes the new value of block B, is still a marked block with p_B as its marked configuration, and consists of all the configurations q of B that "agree" with p_B in the blocks of all their transitions, including the t-transition; i.e., for all transitions a that are enabled in B, the images $a(p_B)$ and $a(q)$ belong to the same block of the current partition. This set is the intersection of B with all inverse images $a^{-1}(C)$ over the arcs $B \to C$ labelled a of the marked graph coming out of B. The second part, say $B'' = B - B'$, consists of all the remaining configurations of B.

A result of the splitting may be that some arcs of the marked graph H coming into block B, become "invalid"; an arc $C \to B$ labelled a is *invalid* if the image $a(p_C)$ of the marked configuration of C is not any more in B, otherwise it is valid. Also, the t edge out of B, say to block D may become invalid; in that case the new t edge of B goes to B'' and that of B'' goes to D. We examine the arcs of the marked graph H coming into block B, to see if they are still "valid", and update H. All such valid immediate predecessors C of B are placed in the

back of the queue of potentially unstable (marked) blocks; note that B itself may thus join again the queue if it has a self-loop, but it goes now in the back of the queue. If some arc into B is no more valid, then we mark block B'', we let its marked configuration be $p_{B''} = a(p_C)$ for some arc $C \to B$ that became invalid, we make all invalid arcs of the marked graph point to B'', and we go back into searching out of the new marked configuration $p_{B''}$. We act similarly if the t edge of B becomes invalid.

The class of zones is closed under the operations of intersection and inverse image of an ordinary transition a or the special t transition. The main problem is that the class is not closed under the difference operator, which is important in the algorithm so that a block is only split into two pieces. If we want to represent explicitly this operator we have to use more general representations for the blocks than zones. The problem with such a solution is that as we operate on them, the length of the representations will in general grow exponentially.

We know from the previous section that zones are sufficient to represent the minimal system, i.e., the difference operator is not inherently needed to compute the final blocks. If we stay with the zone representation for the blocks, one solution is to partition the difference $B - B'$ arbitrarily into disjoint zones [2]. One problem with this approach is that in general it will not compute the minimal system, but it will refine the partition more than is necessary. Also, the useless splits can cascade causing futher splits and so on, and there is no guarrantee that the time will not be exponential in the number of reachable blocks.[2] An alternative solution is to maintain at all times the blocks of the partition as zones, and when we split a block B by some transition a (or t), we partition B completely into the sets $B \cap a^{-1}(C_1), \ldots, B \cap a^{-1}(C_r)$, as in the proof of Theorem 3.3. This approach will produce the correct minimal reachable graph; however, the number of blocks will in general proliferate as the algorithm progresses, yielding exponential complexity in N.

We shall describe now a way of avoiding the explicit application of the difference operator, while guarranteeing polynomial time performance. The approach is applicable in general to deterministic transition systems. We can use a forest data structure F to represent the history of splittings that the blocks undergo. There is one tree T_s in F for each control state s of the timed graph. The root of T_s corresponds to the set (s, R^k) of all configurations with first component s, and has as its children the blocks of the initial partition π with state s. It is not really necessary to list these blocks a priori, but only to produce them as needed when they are reached; i.e., given a configuration (s, v), we need to compute the block of π that contains it. For simplicity in the exposition we will

[2] [2] implemented also an algorithm that uses some ideas from [19], but which lacks the minimality and the performancee guarrantees of [19]: both the output of the algorithm and its time complexity depend on nondeterministic choices made in the calls to the difference routine. The implementation differs also in other significant ways: for example, it uses regions to mark blocks and updates them every time the corresponding blocks are split; it keeps track of all possible arcs from all (marked and unmarked) blocks, which again it updates after every split [25].

regard F as containing initially all blocks of π. The nodes of F are arranged in levels according to their distance from the roots, which are at level 0.

We list now some properties of the forest F. Every node u of F is associated with a set B_u of configurations. If u is an internal node, then the sets associated with its children form a partition of B_u. The nodes of F are partitioned into marked and unmarked. Every marked node u has an associated marked configuration $p_u \in B_u$. All internal nodes (except possibly the roots) are marked. If u is an internal node with marked configuration p_u, then u has a marked child with the same marked configuration p_u. We shall also distinguish the marked nodes as being *processed* or unprocessed; the nodes of the initial partition π (and the roots) are considered processed. Recall that all blocks of π are zones. It will always be the case that processed nodes correspond to zones.

The sets corresponding to the leaves of F form the current partition ρ. Initially $\rho = \pi$. Mark the node containing the initial configuration (s_0, v_0). The general step of the algorithm is as follows.

1. If there is an unexplored marked configuration p, then explore it: For every transition a (including the t transition) compute $a(p)$. If the block of the current partition containing $a(p)$ is unmarked, then mark it and let $a(p)$ be its marked configuration.
2. Otherwise (i.e., if all marked configurations are explored), let l be the least level with a marked leaf.
   (a) If there is an unprocessed marked leaf u at level l, then let w be its parent. Split the block B_u of u into two parts, B' and B''. The first part B' is the new value of B_u; it is the intersection of B_w with all inverse images $a^{-1}(B_x)$ where x is a (marked) node at level $l-1$ and $a(p_u) \in B_x$. Node u is now considered processed. The second part B'' is the rest of B_u and it is associated with a new child z of w that is not marked. Update the arcs of the marked graph incident to B_u, and if one of them is now directed into B_z then mark z and choose a marked configuration for it.
   (b) Otherwise (all leaves at level l are processed), let u be any marked leaf at level l. Attach a child y to u, and make it a marked, processed node (at level $l+1$) with marked configuration $p_y = p_u$. The corresponding block B_y is the intersection of B_u with all inverse images $a^{-1}(B_x)$ such that x is a (marked) node at level l and $a(p_u) \in B_x$. If $B_y \neq B_u$ then we attach one more child z to u with corresponding block $B_u - B_y$; the node z is unmarked (and unprocessed). Update the arcs of the marked graph incident to u and if B_z has an arc now, then mark z and choose a marked configuration.

Steps 2a and 2b above include also the special t transition as one of the possible actions a. For example, in step 2a if the time trajectory of p_u exits block B_w and enters next block B_x of level $l-1$, then we form the intersection $B_w \cap t^{-1}(B_x)$ consisting of all configurations of B_w whose time trajectory passes directly from B_w to B_x. The following lemma lists some properties that hold throughout the algorithm.

Lemma 4. *1. All internal nodes (except possibly the roots) are marked and processed.*
2. All processed nodes correspond to zones.
3. Every unmarked or unprocessed leaf has at least one sibling, and all its siblings are processed (and marked).

We do not need to represent explicitly the set B_u associated with an unmarked or unprocessed node; the set is implicitly defined as the difference between the block of its parent and those of its siblings, all of which are zones.

An important property is that the forest does not become too deep before we reach the minimal reachable graph.

Lemma 5. *If the minimal reachable graph G_m has N nodes, then by the time the forest reaches depth N, the marked graph is equal to G_m.*

The operations used by the algorithm are: (1) compute $a(p)$ for a configuration p and transition a; (2) test whether $p \in B$ for a block $B = (s, Z)$ where Z is a zone; (3) compute $B \cap a^{-1}(C)$ and test for emptiness for two zones B, C and transition a (including the t transition). Note that we only need forward images $a(p)$ of individual configurations and not images $a(B)$ of blocks. Combining the two lemmas we have.

Theorem 6. *Suppose that the minimal reachable graph has N nodes and M arcs. After at most $O(NM)$ operations the marked graph is equal to the minimal reachable graph. At that point the forest has $O(N^2)$ nodes and depth at most N.*

There are some obvious optimizations that one can do. For example, the way we described the construction, the forest was allowed for simplicity to have internal nodes of degree 1, i.e., identical to their parents; these can obviously be suppressed, which is what one would do in practice.

5 Implementation of the operations

As in [14, 2], a zone can be represented by a $(k+1) \times (k+1)$ matrix A where k is the number of clocks; the 0th row and column of A is indexed by a new dummy variable x_0 that stands for the constant 0, and the remaining k rows and columns are indexed by the clocks x_i. An upper bound $x_i \theta c$, where θ is $<$ or $\leq$, on the value of clock x_i can be equivalently written as $(x_i - x_0)\theta c$, and a lower bound $c \theta x_i$ can be written as $(x_0 - x_i)\theta(-c)$. Thus, every inequality can be written as an upper bound constraint on the difference of two variables; the constraint may be strict (θ is $<$) or weak (θ is $\leq$), and the bound is an integer or ∞. The ijth entry of A gives the upper bound on the difference $x_i - x_j$ and an indication whether it is strict or weak.

A zone Z may have different matrix representations. Among them, there is a unique "tightest" matrix with the sharpest possible bounds. Given a set of inequality constraints bounding the differences of the variables, we can determine

whether the constraints are consistent (i.e., whether the feasible set is nonempty), and find the corresponding tight matrix if it is, by solving an all-pairs shortest path problem on an appropriately constructed directed graph D whose nodes correspond to the variables and whose arcs have the bounds as their lengths (see [11, 14]). The constraints are consistent iff D has no negative length cycles, and in that case the pairwise distances specify the tight matrix. We will represent a zone by its tight matrix.

The intersection of a zone B with another zone C, or with the inverse image $a^{-1}(C)$ by a transition a can be computed easily [2]. For $B \cap C$ form the entrywise minimum of the two representative matrices (and tighten the result if desired). For $B \cap a^{-1}(C)$ replace first the clock variables x_i that are reset to 0 by 0 (x_0) in the constraints of C, and then form the intersection with B. The set $B \cap t^{-1}(C)$ for the immediate time action t can be computed by first forming the boundary between the two regions where a time trajectory may pass from B to C, and then intersecting the inverse *time* image of this boundary with B.

We can mark blocks as follows. A marked configuration p for a block B is a pair (s, v) where each clock value v_i is of the form $c_i + d_i\epsilon$ with c_i a nonnegative integer and d_i an integer between 0 and k, and ϵ is sufficiently small, for example $\epsilon = 1/(k+1)$ will do (or we can treat ϵ as a symbol if we wish). The exact value of ϵ is unimportant; what matters is the integer parts of the v_i's and the relative order of their fractional parts. We think of p as a representative of a reachable region contained in the block B. The marked configuration p does not change when B is split. It is straightforward to test whether p belongs to a given zone, and to compute the image $a(p)$ of p by an explicit transition a. We can compute the immediate time "successor" configuration $t(p)$ in $O(k)$ time by taking the point where the time trajectory from p hits the boundary of B if B is open (the bounding inequality of B is strict), or a point that is outside B by an ϵ amount if B is closed (there is a small subtle point in the latter case that we omit here for lack of space).

Each of the basic set operations takes no more than $O(k^3)$ time. In fact, if B_u is a marked zone with d outgoing transitions, when we refine B_u in step 2a or 2b, we can compute the portion of B_u that agrees with its marked configuration p_u (and tighten its matrix and check if it is a proper subset) in time $O(dk^2 + k^3)$. We can show then:

Theorem 7. *Suppose that the minimal reachable graph has N nodes and M arcs. After $O(k^3N^2 + k^2NM)$ time, the marked graph is equal to the minimal reachable graph.*

In terms of space, we maintain the following information: (1) The marked graph, which does not exceed the final graph G_m. Note that we do not maintain any arcs incident to unmarked blocks. (2) For every marked block in the current partition we have its matrix ($O(k^2)$ space) and its marked configuration. Note that the space overhead of having a marked configuration is small compared to the matrix. On the other hand, by exploiting the marked configuration we can in general avoid creating unnecessarily a lot of useless zones (each of which would

cost k^2 space). (3) The forest F and blocks for the marked nodes. If we suppress degree 1 nodes, then the number of internal nodes is no more than the number of leaves, i.e., the number of blocks in the current partition. This is in general no more than N^2, and could be much less if the forest is bushy and shallow.

6 Termination

We shall only sketch the basic idea here, and postpone the details for the full paper. Let H be the marked graph. The question whether H is equal to the minimal reachable graph can be reduced to a Linear Programming problem. However, we do not need to worry explicitly about termination: the linear program has a special form, which implies that our algorithm left on its own will terminate in polynomial time with the stable reachable system.

Suppose that $H = G_m$ and consider the tight matrix A_u representing each marked block B_u in the final stable system. Viewing the entries of these matrices as variables, we can write a set of linear constraints that have to be satisfied. (1) The entries of each matrix must satisfy the triangle inequalities. (2) The entries of a matrix A_u must be consistent with the corresponding marked configuration p_u, i.e., they must give valid bounds. Furthermore, they cannot exceed the corresponding entries in the current matrix representing the block of node u. (3) For every arc $B_u \rightarrow B_w$ of the marked graph labelled by a transition a, the matrices A_u, A_w corresponding to u and w must be consistent with it; for example, if clocks x_i and x_j are not reset by a, then $A_u[i,j] \leq A_w[i,j]$. (4) Similar inequalities must hold for a t-arc; for example $A_u[i,j] \leq A_u[i,0] + A_w[j,0]$ for all $i,j = 1,\ldots,k$. In addition, there are conditions relating the strictness of the bounds. Conversely, it can be shown that if the above set of linear inequalities has a solution, then $H = G_m$. In this case there is a unique componentwise maximal solution, which gives the tight matrices representing the final stable reachable blocks.

All inequalities in the above linear system are either of the form $y_i \leq \sum y_j + c$ or of the form $0 \leq \sum y_j + c$, where the y's are variables and c is a constant (actually every right-hand side is the sum of at most two variables). Such linear programs can be solved without resorting to a general LP algorithm. Ullman and Van Gelder [22] considered such systems in the context of the evaluation of recursive logical rules and showed that a simple iterative algorithm reaches a solution (a fixpoint), if there is one, in at most n iterations where n is the number of variables (in our case k^2N). It can be seen that one round of splitting each unstable marked block is at least as effective as one iteration of this algorithm. Therefore, if $H = G_m$ our algorithm will stabilize all the marked blocks and terminate after no more than k^2N rounds. If $H \neq G_m$ then a new block will be marked within this time. Summarizing, we have:

Theorem 8. *Let G be a timed graph, (s_0, v_0) an initial configuration, π an initial partition such that every block of π is a zone and its configurations have the same state and enabled transitions. The algorithm computes the minimal reachable*

transition system and terminates in time polynomial in the size of the input and the output.

7 Discussion

We presented an efficient algorithm for the minimization of timed transition systems according to "transition equivalence". Starting from an initial partition into zones that respect the enabling conditions of the transitions, the algorithm achieves its efficiency by staying with a representation of the blocks as zones, i.e., *conjunctions* of constraints, carefully avoiding the difference operator that would turn conjunctions into disjunctions. Our technique for enforcing this is quite general and does not depend on any special properties of timed systems; it applies besides them to any deterministic systems.

Disjunctive representations for sets are inconvenient when combined with intersections, because the number of terms (disjuncts) grows very rapidly. It is crucial for our results that we started with a partition into zones. If this is not the case, then we cannot guarrantee polynomial time in the number of reachable blocks; in fact we can show an NP-hardness result in this regard. The problem in that case is that dealing with nonconvex sets (differences of zones) leads to an exponential increase in the size of the representation. That is, even if there is a small number of reachable blocks, their descriptive complexity may be exponentially large.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-timed systems. In *Proc. of the 5th IEEE Symp. on Logic in Computer Science*, pp. 414–425, 1990.
2. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. of Real-Time Systems Symp.*, pp. 157–166, 1992.
3. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. of CONCUR'92*, Springer-Verlag LNCS 630, pp. 341–354, 1992.
4. R. Alur, and D. Dill. Automata for modeling real-time systems. In *Proc. of the 17th Intl. Colloq. on Automata, Languages and Programming*, Springer-Verlag LNCS 443, pp. 322–335, 1990.
5. R. Alur, and T. Henzinger. A really temporal logic. In *Proc. of the 30th IEEE Symp. on Foundations of Computer Science*, pp. 164–169, 1989.
6. A. Bouajjani, J. Fernandez, and N. Halbwachs. Minimal model generation. In *Proc. of the 2nd Workshop on Computer-Aided Verification*, DIMACS Series vol. 3,ACM-AMS, pp. 85–91, 1990.
7. A. Bouajjani, J. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. In *Science of Computer Programming*, to appear, 1992.
8. J. R. Burch,E. M. Clarke, L. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proc. 5th IEEE IEEE Symp. on Logic in Computer Science*, pp. 428–439, 1990.

9. K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. of the 4th Workshop on Computer-Aided Verification*, 1992.
10. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. In *ACM Trans. on Prog. Lang. and Sys.*, vol. 8, pp. 244–263, 1986.
11. T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, McGraw Hill, New York, 1990.
12. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory Efficient Algorithms for the Verification of Temporal Properties. In *Proc. 2nd Workshop on Computer-Aided Verification*, DIMACS Series Vol. 3, ACM-AMS, pp. 207–218, 1990. Full version in *Formal Methods in System Design*, vol. 1, 1992.
13. C. Courcoubetis, and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. 3rd Workshop on Computer-Aided Verification*, 1990. Full version in *Formal Methods in System Design*, vol. 1, pp. 385–415, 1992.
14. D. Dill. Timing assumptions and verification of finite state concurrent systems. In *Automatic Verification Methods for Finite-State Systems*, J. Sifakis, ed., Springer Verlag LNCS 407, 1989.
15. T. Henzinger, X. Nicollin, J. Sifajis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th IEEE IEEE Symp. on Logic in Computer Science*, 1992.
16. G. J. Holzmann. An Improved Protocol Reachability Analysis. In *Software, Practice and Experience*, vol. 18, pp. 137–161, 1988.
17. G. J. Holzmann. *Design and Validation of Protocols*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.
18. P. Kanellakis, and S. Smolka. CCS Expressions, Finite State Processes and Three Problems of Equivalence. In *Information and Computation*, vol. 86, pp. 43-68, 1983.
19. D. Lee, and M. Yannakakis. Online minimization of transition systems. In *Proc. ACM Symp. on Theory of Computing*, 1992.
20. H. Lewis. A logic of concrete time intervals. In *Proc. 5th IEEE IEEE Symp. on Logic in Computer Science*, pp. 380–389, 1990.
21. R. Paige, and R. Tarjan. Three Partition Refinement Algorithms. In *SIAM J. on Computing*, vol. 16, pp. 973–989, 1987.
22. J. D. Ullman, and A. Van Gelder. Efficient Tests for Top-Down Termination of Logical Rules. In *J. ACM*, vol.35, pp. 345-373, 1988.
23. M. Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. 1st IEEE IEEE Symp. on Logic in Computer Science*, pp. 322-331, 1986.
24. C. H. West. Generalized Technique for Communication Protocol Validation. In *IBM J. Research and Development*, vol. 22, pp. 393–404, 1978.
25. H. Wong-Tei, personal communication, 1992.