Formal Methods in System Design 11, 113–136 (1997)

An Efficient Algorithm for Minimizing Real-Time Transition Systems

MIHALIS YANNAKAKIS mihalis@research.bell-labs.com
DAVID LEE lee@research.bell-labs.com
Bell Laboratories, Murray Hill, NJ 07974

Abstract. We address the problem of performing simultaneously reachability analysis and minimization of real-time transition systems represented by *timed automata*, i.e., automata extended with a finite set of clock variables. The transitions of the automaton may depend on the values of the clocks and may reset some of the clocks. An efficient algorithm is presented for minimizing a system with respect to a given initial partition that respects the enabling conditions of the transitions of the timed automaton. Our algorithm generates the portion of the minimized system that is reachable from a given initial configuration in time polynomial in the input and the size of the minimal reachable system.

Keywords: timed systems, minimization, verification, transition systems, bisimulation

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, called the region graph. This graph can be used to check temporal logic properties of the timed system.

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 [6, 7] and [19]. An advantage of the method of [19] is that it guarrantees a running time polynomial in the size of the output. Minimization is performed with respect to bisimulation equivalence [21], that is, the algorithms compute the reachable part of the bisimilarity quotient. This can be used as a preprocessing step for checking any properties that cannot distinguish bisimilar states, which includes all properties expressed in standard linear and branching time temporal logics. Alur et al., adapted the method of [6, 7] 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 method of [19].

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. Note that this is the type of constraints that are typically used in the enabling conditions in timed automata. We shall develop an algorithm that constructs 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 do two things: (1) implement efficiently the basic operations and (2) address the termination problem. The main technical obstacle for (1) is presented by the difference operation: the difference of two conjunctions of constraints (e.g., convex polyhedra) is not a conjunction (it is in general nonconvex). This obstacle is circumvented by using an implicit representation of difference, which allows us to stay with conjunctive explicit representations, while maintaining the efficiency of the algorithm. This technique applies in general to deterministic transition systems, and although timed systems are not deterministic, as we observe they almost are for our purposes. For (2) we prove that nothing special needs to be done: our algorithm left on its own will terminate in polynomial time.

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 system is deterministic if the transition relation R_a for every action a 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 labeled 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 $\{q \mid \exists p \in S \,.\, (p, q) \in R_\alpha\}$ and $\alpha^{-1}(S)$ for $\{p \mid \exists q \in S \,.\, (p, q) \in R_\alpha\}$.

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 $B \to C$ is *stable* if every configuration of B has an a-transition to some configuration of C. A block B is stable if all its outgoing arcs are stable; 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 E of arcs 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[1]. 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, called 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 $\emptyset$.[2] We will often use the notation (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' 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. Note that the region graph is exponential in the size of the input timed graph (i.e., the space needed to write it down), because of the factors $k!$ and b^k; the binary representation of the constants needs $w = \log b$ bits, hence $b^k = 2^{wk}$ is exponential in both the number k of clocks and the length w of the constants. 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 3.1. *Consider a partition of the set of configurations into blocks that are zones.*

1. *Either all time trajectories starting in configurations of a block 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 B does not have any such upper bound constraints, then all trajectories will stay in B.
2. Suppose that the immediate time successor of configuration (s, v_1) is block C_1 and that of (s, v_2) is a different block C_2, and both blocks have time arcs from both configurations. In other words, if we consider the time trajectory starting at (s, v_1), first it encounters

a configuration (s, w_1) in C_1 and then later a configuration (s, w_2) in C_2. Similarly, the time trajectory starting at (s, v_2) first meets a configuration (s, z_2) in C_2 and then (s, z_1) in C_1. Since time trajectories are parallel, the line segments $[w_1, z_1]$ and $[w_2, z_2]$ intersect at a point y. By the convexity of zones, (s, y) must belong to both C_1 and C_2, contradicting the fact that they are distinct, disjoint blocks. □

Consider a variant $\bar{\Theta}/\pi$ 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; we will see in Section 5 how to derive a linear constraint description of the appropriate form for $B \cap t^{-1}(C)$ from the constraints of B and C. 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 all stable with respect to t. That is:

Lemma 3.2. *The system Θ/π has all the time arcs stable iff the system $\bar{\Theta}/\pi$ has all the t-arcs stable.*

Proof: The one direction follows from Lemma 3.1: if all time arcs are stable then so are all the t-arcs. For the converse, suppose that two configurations (s, v), (s, v') of the same block B do not have time arcs to the same blocks, say (s, v) has a time arc to block D, but (s, v') does not have such an arc. That is, the time trajectory of (s, v) contains a configuration $(s, w) \in D$ $(w = v + \delta)$, but the trajectory of (s, v') does not have any configuration of D. Let D be the earliest such block in the time trajectory of (s, v), and let C be the previous block on the trajectory. Then there is a t-arc from C to D. By our choice of D, block C has a configuration on the time trajectory of (s, v') and the immediate time successor of this configuration is not D. Therefore the t arc from C to D is not stable. □

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

Theorem 3.3. *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 guaranteeing 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 intersection, 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, this algorithm may not produce the minimal reachable graph (because of nondeterministic choices made in the implementation of the difference operator) and can run in time exponential in the size of the input and the output.

Example. Consider the timed system with two clocks x, y represented by the timed graph of figure 1(a). There are $n+1$ states $s_0, \ldots, s_n$, and for each $i < n$ there are two transitions from s_i to s_{i+1}; the first transition is conditioned on $y = 0$ and does not change the clocks, and the second transition has condition $y = 2^i$ and resets y to 0. Let M be a large number, say $M \geq 2^n$, and assume that the initial partition π contains five blocks (zones) for each state s_i with the following constraints on the clock values: $\{0 \leq x \leq M, y = 0\}$, $\{0 \leq x \leq M, 0 < y < 2^i\}$, $\{0 \leq x \leq M, y = 2^i\}$, $\{0 \leq x \leq M, y > 2^i\}$, $\{x > M\}$. Note that the initial partition respects the conditions on the transitions. The system starts from state s_0 with $x = y = 0$. It can be seen then that the minimal reachable system contains five blocks for each state s_i, as shown in figure 1(b), where $d_i = 2^i + \cdots + 2^n = 2^{n+1} - 2^i$. That is, the five reachable blocks for s_i are given by the following constraints:

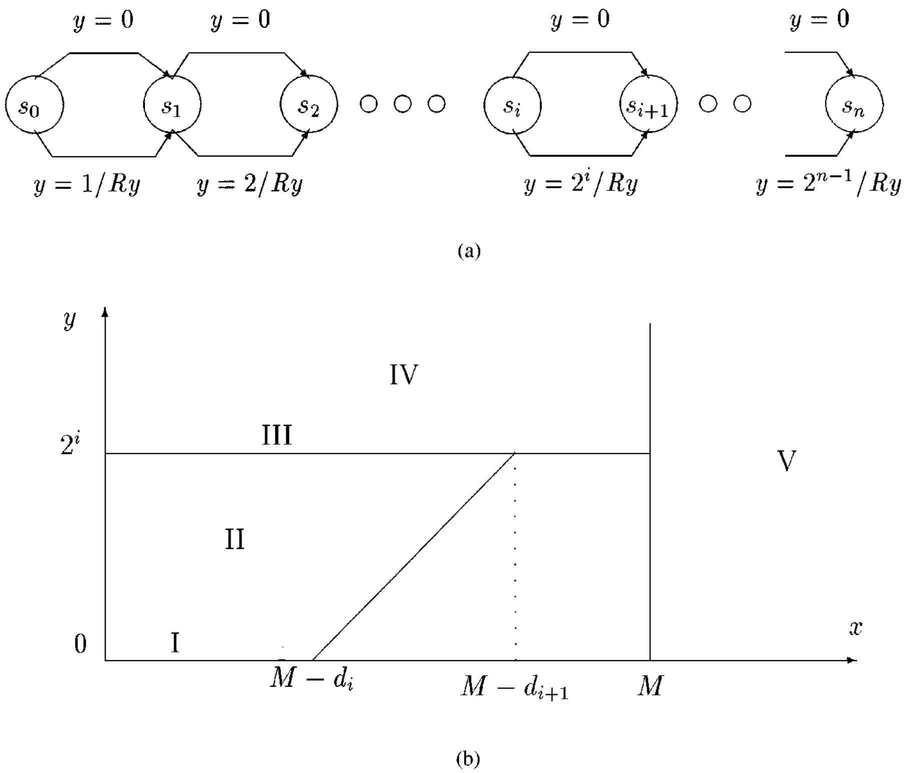

Figure 1.

I$=\{0 \leq x \leq M - d_i, y = 0\}$, II$=\{0 \leq x \leq y + M - d_i, 0 < y < 2^i\}$, III$=\{0 \leq x \leq M - d_{i+1}, y = 2^i\}$, IV$=\{0 \leq x \leq M, y > 2^i\}$, V$=\{x > M\}$. For each s_i, there are t transitions I → II → III → IV → V, and in addition there are ordinary transitions from blocks I and III of s_i to block I of s_{i+1}.

This timed graph (and transition system) is acyclic. In this case, the best way to compute the minimal reachable graph is to search forward from the initial configuration to reach s_n and then split to stabilize the blocks backwards in bottom-up order.

Suppose however that we use a top-down strategy to search and split. That is, we first stabilize all the blocks that are known to be reachable at any given moment before searching forward (initially, only the block of the initial configuration is known to be reachable). Also, at each point in time, if there are several unstable reachable blocks, assume that we prefer to stabilize a block that is earliest in the acyclic order, i.e., a block that corresponds to a state s_i with the smallest index i with ties broken in favor of smaller y. This strategy will incur an exponential number of splits. To see this, suppose that we have already stabilized the reachable blocks up to level $j - 1$ (the blocks are given by the above expressions with $j - 1$ in place of n), and consider what happens when we process s_j. Because of the time transitions, block I of s_j is modified to $\{0 \leq x \leq M - 2^j, y = 0\}$ and as a consequence, blocks I and III of s_{j-1} become unstable. We first stabilize block I of s_{j-1} modifying it to $\{0 \le x \le M - 2^j, y = 0\}$, as a result of which the previous levels become unstable so we have to reprocess them recursively. We then stabilize block III of s_{j-1} modifying it to $\{0 \le x \le M - 2^j, y = 2^{j-1}\}$. Because of the time transitions, block I of s_{j-1} becomes again unstable and has to be shrunk further to $\{0 \le x \le M - 2^j - 2^{j-1}, y = 0\}$, which again has to be followed by the reprocessing of the previous levels. The number of splits $T(j)$ that are performed to stabilize j levels by this strategy obeys the recurrence $T(j) \ge 2T(j-1)$, thus $T(n) \ge 2^n$.

As mentioned above, in this particular example it is best to split and stabilize the blocks in a bottom-up order, since the graph is acyclic. However, general systems of course are not acyclic, thus there is no top and bottom, and it is not clear a priori in which order to split the blocks. As the example shows, the strategy for searching and splitting can make a significant difference in the complexity. □

We shall develop an algorithm based on the method of [19] with the goal of providing guarantees 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 without changing the graph structure (but of course it will not refine them 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 $O(NM + q(N, M))$ operations.

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 (ordinary) 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 that is 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. We remark that, although all configurations of B' (the new value of B) agree in the blocks of all their transitions with respect to the partition *before* the split of B, they may disagree with respect to the partition *after* the split, i.e., block B may still be unstable after the split. This may happen if B has self-loops, and also it may happen because of the t-transition.

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 (i.e., it is 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 lengths 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 further splits and so on, and there is no guarantee that the time will not be exponential in the number of reachable blocks[3]. 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 guaranteeing polynomial time performance. The approach is applicable in general to deterministic transition systems. Although timed transition systems are not quite deterministic because of the *time* transitions, as we discussed they are "almost" deterministic if we use instead the t actions (and the minimized quotient system is indeed deterministic).

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 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 (i.e., not a leaf), 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. "Processing" a node will have the effect of making its block into a zone. Recall that all blocks of π are zones. It will always be the case that processed nodes correspond to zones. Furthermore, during the algorithm all internal nodes will be processed (and marked).

The sets corresponding to the leaves of F form the current partition ρ. In addition, each level l of F induces a (generally coarser) partition ρ_l consisting of the blocks corresponding to nodes at level l and leaves at smaller levels. Initially, ρ is the initial partition π, and F has two levels: the roots at level 0 and the blocks of π at level 1. Mark the initial configuration (s_0, v_0) and the level-0 and level-1 node containing it.

The basic strategy of the algorithm is as follows. As mentioned before, we give priority to searching rather than splitting. The order of splitting reachable blocks is guided by the forest F, giving priority to lower-level nodes: we first process them to turn them into zones and then split to stabilize them, but only with respect to the partition induced by this level. In more detail, the general step of the algorithm is as follows.

1. If there is an unexplored marked configuration p, then explore it as follows. For every transition a, including the t transition, do the following. Compute $a(p)$; for the case $a = t$ (the t transition), if the time trajectory from p exits the block of the current partition ρ that contains p, then we let $t(p)$ be any configuration on the time trajectory in the next block of ρ that it enters. If the block of the current partition containing $a(p)$ is unmarked (including the case $a = t$), 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 the parent of u. Split the block B_u of u into two parts, B' and B''. The first part B' is the new value of B_u, and it consists of all the configurations of B_w that agree with p_u in all transitions (including the t transition) with respect to the partition ρ_{l-1}: B' is formed by taking the intersection of B_w with the inverse images $a^{-1}(B_x)$ of the blocks B_x of ρ_{l-1} that contains $a(p_u)$, for all transitions a. In the case of the special t transition, if the time trajectory of p_u exits block B_w and enters next block B_x of ρ_{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; if the trajectory of p_u does not exit block B_w, then we do not need to do anything about the t transition. Node u is now considered processed. The second part B'' is the rest of the original block B_u and it is associated with a new child z of w that is not marked or processed. Update the arcs of the marked graph incident to B_u: for each arc directed into B_u, e.g., arc $B_x \to B_u$ labelled a, we check if $a(p_x) \in B'$ (the new B_u), and if not, then we redirect the arc into B_z ($=B''$), i.e., we replace it by arc $B_x \to B_z$ labelled a. After updating the marked graph, if any arc is now directed into B_z, then mark z and choose a marked configuration $p_z = a(p_x)$ for it, where a-arc $B_x \to B_z$ is any arc directed into B_z.

   (b) Otherwise (all marked 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 consists of the configurations of B_u that agree with p_u with respect to the partition ρ_l; i.e., B_y is the intersection of B_u with the inverse images $a^{-1}(B_x)$ of the blocks B_x of ρ_l that contains $a(p_u)$, for all transitions a. Again, in the case of the special t transition, if the time trajectory of p_u exits block B_u and enters next block B_x of ρ_l, then we form the intersection $B_u \cap t^{-1}(B_x)$ consisting of all configurations of B_u whose time trajectory passes directly from B_u to B_x; if the trajectory of p_u does not exit block B_u, then we do not need to do anything about the t transition. 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). We do not actually compute explicitly the difference $B_u - B_y$; this is just a symbolic representation. Update the arcs of the marked graph incident to u as in Case 2a above, and if an arc gets directed into B_z, then mark z and choose a marked configuration for it.

The algorithm terminates, when the depth of the forest increases by 1, say from level l to $l + 1$, without changing the number of leaves (i.e., the number of blocks in the partition, and hence the partition itself). This happens when all the marked leaves are at the same, maximum level l, we attempt to split the corresponding blocks in Step 2b to stabilize them, and each one spawns only one child, which means that the marked blocks are all stable.

There are some obvious optimizations that one can do in the algorithm. First, as we mentioned, we do not need to include the blocks of π explicitly in the forest, but generate them, only as needed. Second, in our description of 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. Also, if the marked graph is not strongly connected, then in the splitting part (Step 2) it is better to handle separately the strong components of the marked graph in a bottom-up order (recall our earlier example). That is, split a (marked) block only if all the blocks in lower strong components of the marked graph are stable.

We will prove now the correctness of the algorithm and analyze its complexity in terms of set operations. The implementation of the operations will be discussed in the next Section, and the termination issue will be addressed in Section 6. The following lemma lists some properties that hold throughout the algorithm. The properties imply in particular that whenever the algorithm intersects in Step 2a or 2b a block B_u or B_w with the inverse image $a^{-1}(B_x)$ or $t^{-1}(B_x)$ of another block B_x, then all these blocks are zones.

Lemma 4.1.
1. *All internal nodes* (*except possibly the roots*) *are marked and processed.*
2. *All processed nodes correspond to zones.*
3. *Every unprocessed leaf has at least one sibling*, *and all its siblings are processed* (*and marked*).

Proof: Initially, F contains the roots and the blocks of the initial partition which are zones and are considered processed, so the statements hold. Consider a step of the algorithm and suppose that the claims hold up to that point. Step 1 just marks some new leafs, and does not affect the claims. Step 2a processes an unprocessed marked node (leaf) u, by modifying B_u and creating a sibling z of u. Since l is the lowest level with a marked leaf, all previous levels consist of internal nodes and unmarked leaves. Since p_u is marked and all the marked configurations have been explored, if B_x is the block of the partition ρ_{l-1} that contains $a(p_u)$ for some transition a (including the special transition t), then B_x cannot be an unmarked leaf, and hence it must be an internal node and thus a zone. The new block B_u is formed by intersecting B_w (a zone) with the inverse images of blocks B_x (zones), and thus the new B_u is a zone. Since u was an unprocessed leaf before this step, all its siblings were zones, and therefore the same is true of the new unprocessed leaf z.

Step 2b changes a marked leaf u into an internal node by attaching one child y that is a marked, processed leaf, and attaching possibly a second child z. As in Step 2a, if B_x is the block of ρ_l containing $a(p_u)$, then B_x cannot be an unmarked leaf, hence it must be a marked (internal or leaf) node, and therefore processed and thus a zone. It follows that B_y is a zone. The second child z is unprocessed, but clearly it has a processed marked sibling, namely y.

By the lemma, we do not need to represent explicitly the set B_u associated with an 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 4.2. *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.*

Proof: For each level $l > 1$ let t_l be the time at which level l is created and σ_{l-1} the current partition right before that point. A new level is created at Step 2b of the algorithm by splitting a node at the previous level. Note that σ_{l-1} is the partition ρ_{l-1} induced by level $l-1$ at time t_l, and all its marked blocks are processed (zones). Let n_{l-1} be the number of blocks of σ_{l-1} that contain reachable configurations. We remark that n_{l-1} may be greater than the number of marked blocks of σ_{l-1}, because we may not have discovered yet that some of the unmarked blocks are reachable. Clearly every block of the minimal reachable graph is contained in a (reachable) block of σ_{l-1}, hence $N \geq n_{l-1}$. If σ_{l-1} has N marked blocks, then the marked graph is equal to the minimal reachable graph G_m. Suppose that there are less than N marked blocks. We will argue then that $n_l > n_{l-1}$, i.e., by the time the next level is created, at least one more reachable block will be produced.

Suppose that σ_{l-1} has some reachable but unmarked block(s). Since all blocks of G_m are reachable from the initial block (the block containing the initial configuration) which is marked, G_m must contain some arc $B \to C$, where B is contained in a marked block of σ_{l-1}, say block B_u, and C is contained in an unmarked block. Let a be the label of the arc. When level l is created, we have explored the marked configuration p_u of B_u and $a(p_u)$ belongs to some marked block. Before the creation of level $l+1$, we will split at some point B_u to the set B_y of configurations that agree with p_u and the remaining set B_z. Clearly B is contained in B_z, and hence B_z is a new reachable block. Thus, $n_l > n_{l-1}$ in this case. Note, in particular that the argument implies that it cannot be the case that $n_{l-1} = N$ and yet the number of marked blocks is less than N.

Suppose now that all reachable blocks of σ_{l-1} are marked but $n_{l-1} < N$. Every block of G_m is contained in a marked block of σ_{l-1}. Consider the partition τ obtained from G_m by unioning all blocks that are contained in the same block of σ_{l-1}. Note that there is a one-to-one correspondence between the blocks of τ and the blocks of σ_{l-1} where every block of τ is contained in the corresponding block of σ_{l-1}. The partition τ is unstable (because G_m is minimal and $n_{l-1} < N$), thus, some block B of τ is unstable for some transition a, either an ordinary transition or the t-transition. There is a subtlety in the case $a = t$: Note that some blocks of G_m that are contained in B may have t-arcs in G_m to other blocks within B; this is not unstable. The block B of τ is unstable for the immediate time transition if two configurations of B have time trajectories which exit B and then enter different blocks of τ. Note that a time trajectory exits B if and only if it exits the block B_u of σ_{l-1} that contains B. The (if) direction of this fact follows from the containment $B \subseteq B_u$, and the (only if) direction follows from the fact that if the trajectory exits B it must enter another block of τ and that all other blocks of τ are disjoint from B_u. Since B_u is a zone, Lemma 3.1 implies that either all trajectories exit B or they all stay in B.

As above, let B_u be the block of σ_{l-1} that contains an unstable block B of τ. The creation of level $l + 1$ involves splitting a node at level l. Since blocks are split according to level, every block of σ_{l-1} (hence also B_u) gets a turn at being split before level $l + 1$ is created. Some block of G_m that is contained in B disagrees with p_u on the transition a with respect to the partition τ and therefore also with respect to the partition σ_{l-1}. Thus, splitting B_u will create at least one more reachable block, hence $n_l > n_{l-1}$. □

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

Proof: By the time the forest F reaches depth at most N, the marked graph is equal to G_m, the number of marked leaves is N, the number of marked nodes is at most N^2 (there are at worst N paths of length N), and the number of unprocessed (unmarked) nodes is also at most N^2, because each one of them is attached to a a different internal marked node. There are also the $|\pi|$ level-1 nodes corresponding to the blocks of the initial partition. These could be included only as needed if and when they are reached and marked; we included them in F for simplicity, since anyway the size of the initial partition π will be usually much smaller than N.

Step 1 of the algorithm is executed once for each marked configuration. It involves computing $a(p)$ for each transition and finding the block in the current partition that contains it. The total number of transitions out of the marked configurations is equal to M. One obvious way to find the block containing $a(p)$ is just to try one by one all the blocks of the current partition; this can cost in the worst case $O(N^2)$ operations since there may be that many blocks (at least according to the bounds we have proved). A better way that uses $O(N + |\pi|)$ operations is as follows. Starting from the root of F that contains $a(p)$, walk down the tree until the desired block is found. At each internal node we need to determine which child to proceed to. All of the children except at most one are zones, so we can determine the correct child by testing membership of $a(p)$ in each of the zones in turn, i.e., using operations of type (2). At the root, the children are blocks of the initial partition π. At every other internal node, all but at most one of the children are marked and are ancestors of different marked leaves. Thus the number of operations performed to determine the block of $a(p)$ using this method is at most $2N + |\pi|$. We could also first try to see if $a(p)$ belongs to a marked block of the partition (there are at most N such blocks), and only if we fail (which will happen N times) use the method that walks down the tree. Therefore, the total number of operations in Step 1 is $O(N(M + |\pi|))$. It is an interesting question whether there are geometric data structures to expedite this computation, i.e., for maintaining the partition so that we can locate quickly the block containing a given point.

Step 2a is performed at most once for each node of G_m. Step 2b is performed at most N times for each node of G_m because F has depth N. The number of operations needed to split a block B_u at level l is the number of (outgoing and incoming) arcs incident to it at that point. The total number for a level l is at most $2M$. Thus, the total number of operations in Step 2 is $O(NM)$. □

5. Implementation of the operations

A zone can be represented by a "difference-bound matrix" due to Dill [14]. This is 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 the inequality is strict or weak.

For succinctness, we will use the following notation. For each real number c, there is a corresponding *strict* number, denoted c^-; the number c itself is called *weak*. We call the set of strict and weak numbers, *typed* numbers (there are two types: 'strict' and 'weak'). For each typed number $\alpha = c$ or c^-, we use $\#\alpha$ to denote the corresponding untyped number c. We compare typed numbers as follows: $\alpha < \beta$ if $\#\alpha < \#\beta$ or if $\#\alpha = \#\beta$ and α is strict and β weak (i.e., $c^- < c$ for all untyped numbers c). A zone is represented by a matrix of typed numbers (actually, typed integers, since all our coefficients are integers). In the case of a weak inequality $x_i - x_j \leq c$, we have $A[i, j] = c$, and in case of a strict inequality $x_i - x_j < c$, we have $A[i, j] = c^-$. Thus, the ij inequality, whether strict or weak, can be written as $x_i - x_j \leq A[i, j]$. We extend addition to typed numbers as follows: add their corresponding untyped versions and make the result strict if at least one of the numbers is strict.

A zone Z may have different matrix representations. Among them, there is a unique "tightest" matrix with the sharpest (i.e., minimum) 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 [11, 14]. The graph has $k+1$ nodes corresponding to the variables x_i (including x_0), and has an arc $x_i \to x_j$ of length β for every constraint $x_i - x_j \leq \beta$ (the lengths can be typed numbers to represent both weak and strict inequalities). 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 x_0 in the constraints of C, and then form the intersection with B.

Before describing how to compute the inverse image for the t transition, we mention first a few relevant facts. Consider a time trajectory that exits from zone Z_1. It is either the case (i) that there is a point $p \in Z_1$ and every further point $p + \delta$, $\delta > 0$, of the trajectory is not in Z_1, or (ii) there is a point $p \notin Z_1$ such that $p - \delta \in Z_1$ for all small enough $\delta > 0$. In case (i) the trajectory crosses a weak upper bound inequality $x_i \leq c$ of Z_1, and we say that p is a *closed exit point*, while in case (ii) the trajectory crosses a strict upper bound inequality $x_i < c$ of Z_1, and we say that p is an *open exit point*. In general a zone Z_1 may have both some closed and some open exit points. However, one can show that if two trajectories that exit Z_1 enter the same zone Z_2, then the two exit points must be of the same type, both closed or both open. In particular, this means that given disjoint zones B, C, all exit points of the set $B \cap t^{-1}(C)$ are of the same type. In the algorithm, when we need to compute $B \cap t^{-1}(C)$, we know already a trajectory that passes from B to C, namely the trajectory of the marked configuration of B, and hence we know the type of the exit points from B to C.

The set $B \cap t^{-1}(C)$ for the immediate time action t can be computed by first forming the boundary $\partial(B, C)$ between the two zones where a time trajectory may pass from B to C, then computing its inverse *time* image $\mathit{time}^{-1}(\partial(B, C))$, and finally intersecting the result with B. We explain how to compute $\partial(B, C)$ and $\mathit{time}^{-1}(\partial(B, C))$. The boundary $\partial(B, C)$ can be formed by putting together the constraints of B and C as in the intersection operator, except that in the case of open exit from B, all the upper bound inequalities on the clocks of B are made weak, and in the case of closed exit, all the lower bounds of C are made weak. Given any zone D (e.g., $\partial(B, C)$) represented by its tight matrix, the set $\mathit{time}^{-1}(D)$ (i.e., the set of configurations whose time trajectory meets D) can be formed by simply dropping the lower bound constraints of D (note: it is important for the matrix to be tight for this to work correctly).

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, d_i an integer between 0 and k, and ϵ should be thought of as a symbol of a small number; in explicit computations we can use a sufficiently small value for ϵ, for example $\epsilon = 1/(k+1)$ will do. 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. The configuration p is simply a convenient symbolic representative of a reachable region contained in the block B; such a concrete point p with regularly spaced fractional parts may in fact not be itself reachable for any concrete value of ϵ. The marked configuration p of B 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 as follows. If the time trajectory from p does not stay within B (i.e., B has upper bound constraints for some clocks), compute the point where the trajectory hits the boundary of B;. this will happen when the trajectory hits an upper bound $x_j < b_j$ or $x_j \leq b_j$ for the first time. Note that since the bounds are integers, the index j does not depend on the exact values of the coordinates of p but only on the integral parts and the relative order of the fractional parts. The trajectory may hit the upper bound at the same time for a set of more than one clocks x_j; note that, since the bounds are integral, the configuration p has equal fractional parts for these clocks x_j, i.e., equal d_j's. If the exit point is open, i.e., the upper bound inequality $x_j < b_j$ is strict, then $t(p)$ is computed in the straightforward way by adding $b_j - (c_j + d_j\epsilon)$ to all coordinates of p, interpreting ϵ as $1/(k+1)$; the integer parts are increased by the appropriate amounts and the d_i's are shifted cyclically mod$(k+1)$ so that the new value of d_j becomes 0. If the exit point is closed, i.e., the upper bound inequality $x_j \leq b_j$ is weak, there is a small subtlety. We would like to move above the halfspace $x_j \leq b_j$, but only barely, not far enough to have another clock become integer. We can accomplish this symbolically as follows. Compute as above the point $p' = (s, \langle c_i' + d_i'\epsilon \rangle)$ where the trajectory meets $x_j \leq b_j$. Since there are k clocks, $k+1$ possible values for the d_i''s and $d_j' = 0$, it follows that there is a value l between 0 and k that is not equal to any of the d_i''s. Let $t(p)$ be obtained from p' by increasing by 1 all the d_i''s with $0 \leq d_i' < l$. Clearly, this reflects correctly the relative values of the fractional parts of a point right above $x_j \leq b_j$.

Operation (1), i.e., computing $a(p)$ for a configuration p and transition a, takes $O(k)$ time. Operation (2), i.e., testing membership of a configuration in a block, takes $O(k^2)$ time. Operation (3) takes no more than $O(k^3)$ time (the cubic time is due to the all-pairs shortest path computation). 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 $O(dk^2 + k^3)$: first combine in $O(dk^2)$ time the bounds of B_u and $a^{-1}(C)$ for all ordinary transitions a and appropriate blocks C, and then incorporate in $O(k^3)$ time the set $t^{-1}(C)$, tighten the resulting matrix for B_u, and compare with the original matrix. We have then:

Theorem 5.1. *Suppose that the minimal reachable graph has N nodes and M arcs. After $O(k^3N^2 + k^2N(M + |\pi|))$ time, the marked graph is equal to the minimal reachable graph.*

In terms of space, we maintain the following information:

1. The marked graph; clearly, this 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, we have its matrix ($O(k^2)$ space) and its marked configuration ($O(k)$ space). 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.

Typically, the given timed graph is much smaller than the minimal reachable graph. That is, the number k of clocks and the size $|\pi|$ of the initial partition are much smaller than N and M, the number of nodes and arcs of the marked graph. Recall that the region graph has $k!n2^{wk}$ nodes (regions) where n is the number of nodes of the timed graph and w is the number of bits of the constants on the enabling conditions. Thus, in the worst case N and M could potentially be exponential in k, or even worse if the constants on the enabling conditions are very large. We note also that the output graph is typically sparse: if the maximum degree of the input timed graph is d then $M \leq (d+1)N$, and $d \ll N$. Thus, the most significant term in the running time bound of Theorem 5.1 is the quadratic dependence on the output size. Note nevertheless, that the minimal reachable graph can be much smaller (e.g., exponentially smaller) than the region graph. If minimization decreases nontrivially the size of the region graph, even just to a milder exponential (for example, if the reachable region graph has size 2^{wk}, and the marked reachable graph has size $2^{wk/3}$), then doing the minimization will give an improvement.

6. Termination

We shall show that we do not need to worry explicitly about termination: our algorithm left on its own will terminate in polynomial time with the stable minimal reachable system.

When the marked graph H becomes equal to G_m, the forest has depth $l \leq N$. The algorithm will first split the marked leaves in order of level until they are all at the same level l. The time until this point is bounded by the expression $O(k^3N^2 + k^2N(M + |\pi|))$ of Theorem 5.1.[4] From that point on, the marked blocks will be split in a round robin fashion, increasing the level by 1 in each round. We will argue that the algorithm will converge after no more than $O(k^2N)$ rounds. (Recall incidentally that $k \ll N$.)

The outline of the argument is as follows. We will show that every round of splits fixes at least one entry of the difference-bound matrix of one zone to the correct number, i.e., the value of the entry in the minimal reachable system, except that the type may be incorrect (weak instead of strict). Thus, after at most $(k+1)^2N$ rounds, all entries of all matrices will have the correct number. After a set of at most another $(k+1)^2N$ rounds, the types of all entries of all matrices will be also correct, and the system will have converged to the final stable minimal reachable system.

We proceed with the details of the proof. Consider a round of splits. The marked graph H is equal to G_m at the beginning and end of the split. There is a one-to-one correspondence between the marked configurations and blocks on the one hand and the blocks of G_m on the other hand. Let B_r^*, $r = 1, \ldots, N$, be the blocks of the minimal reachable system, and let B_r, B_r' be the corresponding blocks of H at the beginning and the end of the round respectively. Let A_r^*, A_r, and A_r' $r = 1, \ldots, N$, be the tight matrices representing blocks B_r^*, B_r, and B_r' respectively. Clearly, $B_r^* \subseteq B_r' \subseteq B_r$, and thus $A_r^* \leq A_r' \leq A_r$, i.e.,

$A_r^*[i, j] \le A_r'[i, j] \le A_r[i, j]$ for all $i, j = 0, \dots, k$. Recall that the entries of the matrices are typed numbers, and they are compared as such.

Let R be the set of $(k+1)^2 N$ indices $(r, i, j), r = 1, \dots, N; i, j = 0, \dots, k$, of the entries of the matrices, let Q (resp. Q') be the set of indices (r, i, j) such that $\#A_r^*[i, j] = \#A_r[i, j]$ (resp. $\#A_r^*[i, j] = \#A_r'[i, j]$), i.e., the entries are equal except possibly for the strictness type. Clearly, $Q \subseteq Q' \subseteq R$. We shall show that every round adds at least one more index to Q.

Lemma 6.1. *If $Q \ne R$, then Q' is a proper superset of Q.*

Proof: Assume to the contrary that $Q = Q' \subset R$. Consider the blocks $\hat{B}_r$ whose matrices $\hat{A}_r$ are obtained from A_r^* by adding 1/2 to all the entries with indices in $R - Q$. Note that $B_r^* \subseteq \hat{B}_r \subseteq B_r'$. We will argue that (1) the matrices $\hat{A}_r$ are tight, and (2) the blocks $\hat{B}_r$ are stable, contradicting the fact that the B_r^*'s are the blocks of the minimal reachable system.

For (1), suppose that $\hat{A}_r$ is not tight. Then the triangle inequality $\hat{A}_r[i, j] \le \hat{A}_r[i, l] + \hat{A}_r[l, j]$ is violated for some triple i, j, l. Since A_r^* is tight, and thus satisfies the triangle inequality, it must be the case that (r, i, l) and (r, l, j) belong to Q, whereas (r, i, j) does not. The first two facts imply $\#A_r'[i, l] = \#A_r^*[i, l] = \#\hat{A}_r[i, l]$ and $\#A_r'[l, j] = \#A_r^*[l, j] = \#\hat{A}_r[l, j]$, and the third fact implies $\#A_r'[i, j] \ge \#A_r^*[i, j] + 1$, hence $\#A_r'[i, j] \ge \#\hat{A}_r[l, j] + 1/2$. Since A_r' satisfies the triangle inequality, it follows that $\hat{A}_r$ satisfies it too.

For (2), suppose that block $\hat{B}_r$ is not stable for some transition a, i.e., $a(p_r) \in \hat{B}_q$ for the marked configuration p_r of the block, but there is another configuration $f \in \hat{B}_r$ such that $a(f)$ is not in $\hat{B}_q$.

Assume first that a is an ordinary transition, and let C be the set of clocks that it resets. For notational convenience, for each clock index $i = 1, \dots, k$ we let $i' = i$ if $i \notin C$ and $i' = 0$ if $i \in C$; note that with this notation, the new value of each clock x_i after the transition is the old $x_{i'}$. Let v be the vector of clock values of f and w for $a(f)$. Suppose that $a(f)$ violates the ij constraint of $\hat{B}_q$, i.e., $w_i - w_j > \hat{A}_q[i, j]$. Observe that $w_i - w_j = v_{i'} - v_{j'} \le \hat{A}_r[i', j']$.

If $(r, i', j') \in Q$, then $A_r^*[i', j'] = \hat{A}_r[i', j']$, and since the matrix A_r^* is tight, block B_r^* has a point f^* with the same or larger difference $v_{i'} - v_{j'}$. The image $a(f^*)$ of this point has the same or larger value for $w_i - w_j$, and since $a(f^*)$ is in B_q^*, it follows that $w_i - w_j$ satisfies the ij constraint for B_q^* and hence also for $\hat{B}_q$.

Assume then that $(r, i', j') \notin Q$, and consider a point $f^* = (s, v^*) \in B_r^*$ such that the difference $v_{i'}^* - v_{j'}^*$ is equal to $\#A_r^*[i', j']$, or is arbitrarily close to it if the corresponding inequality is strict (this is possible because the matrix A_r^* is tight). We have then $w_i - w_j \le v_{i'}^* - v_{j'}^* + 1/2 \le A_q^*[i, j] + 1/2$, because $a(f^*) \in B_q^*$. Therefore, $\hat{A}_q[i, j] < A_q^*[i, j] + 1/2$, which implies that $(q, i, j) \in Q$, thus $\hat{A}_q[i, j] = A_q^*[i, j]$, and furthermore we must also have $\#A_r^*[i', j'] = \#A_q^*[i, j] = \#A_q[i, j]$. Since $(r, i', j') \notin Q$, we have $\#A_r'[i', j'] \ge \#A_r^*[i', j'] + 1$, and thus there is a point $f' = (s, v') \in B_r'$ such that $v_{i'}' - v_{j'}' > A_r^*[i', j'] + 1/4 > A_q[i, j]$. The image $a(f')$ of this point violates the ij inequality of B_q, contradicting the fact that $a(B_r') \subseteq B_q$.

Suppose now that the unstable transition a of $\hat{B}_r$ is the t transition. Consider the point at which the trajectory from the point f hits the boundary and exits $\hat{B}_r$. In the case of a closed exit point, we can assume without loss of generality that f itself is the exit point and let $t(f)$ be a point that is further along on the trajectory by an arbitrarily small amount $\epsilon > 0$. In the case of an open exit point, we let $t(f)$ be the exit point, and assume that f is within ϵ of the exit. Thus, $f = (s, v)$ and $t(f) = (s, w)$, where $w = v + \epsilon$. Suppose that $t(f)$ violates the ij constraint of $\hat{B}_q$.

Assume first that $i, j = 1, \ldots, k$. We show first that the containment $t(B_r^*) \subseteq B_q^*$ implies that $A_r^*[i, j] \leq A_q^*[i, j]$. To see this, consider a point of B_r^* with as large a value as possible for the clock difference $x_i - x_j$, i.e., $x_i - x_j$ is equal to or arbitrarily close to $\#A_r^*[i, j]$, depending on whether the bound is weak or strict. As time increases, the trajectory leaves $t(B_r^*)$ and enters B_q^*. Note that the difference $x_i - x_j$ remains the same along the trajectory, hence the claim. Similarly, the containment $t(B_r') \subseteq B_q$ implies that $A_r'[i, j] \leq A_q[i, j]$. Since $t(f) \notin \hat{B}_q$, i.e., $w_i - w_j = v_i - v_j > \hat{A}_q[i, j]$, and $v_i - v_j \leq \hat{A}_r[i, j]$ (because $f \in \hat{B}_r$), we have $A_r^*[i, j] \leq A_q^*[i, j] \leq \hat{A}_q[i, j] < \hat{A}_r[i, j]$. Therefore, it must be the case that $(r, i, j) \notin Q$, $(q, i, j) \in Q$, and $\#A_r^*[i, j] = \#A_q^*[i, j] = \#A_q[i, j]$. Since $t(B_r') \subseteq B_q$, we have $A_r'[i, j] \leq A_q[i, j]$, and thus, $\#A_r'[i, j] \leq \#A_r^*[i, j]$, contradicting the fact that $(r, i, j) \notin Q$.

Assume from now on that $t(f)$ satisfies all the ij constraints, for $i, j \neq 0$. Suppose that it violates the $i0$ constraint of $\hat{B}_q$, i.e., $w_i > \hat{A}_q[i, 0]$. Since $f \in \hat{B}_r$ and hence $f \notin \hat{B}_q$, point f must violate some constraint of $\hat{B}_q$. We will argue that it must be some $0j$ constraint (i.e., lower bound constraint on clock x_j). First, note that since $t(f)$ satisfies all the clock difference constraints $i, j \neq 0$ of $\hat{B}_q$, so does f. Second, we will show that $\hat{A}_r[j, 0] \leq \hat{A}_q[j, 0]$ for all j, and hence f satisfies all the upper bound constraints on the clocks. By considering a point of B_r^* with x_j equal to or close to $A_r^*[j, 0]$ and noting that its trajectory then enters B_q^* (because $t(B_r^*) \subseteq B_q^*$), we deduce that $A_r^*[j, 0] \leq A_q^*[j, 0]$ for all j. Similarly, $t(B_r') \subseteq B_q$ implies $A_r'[j, 0] \leq A_q[j, 0]$. The inequality $A_r^*[j, 0] \leq A_q^*[j, 0]$ implies $\hat{A}_r[j, 0] \leq \hat{A}_q[j, 0]$ unless $(r, j, 0) \notin Q$ and $(q, j, 0) \in Q$. But then $\hat{A}_r[j, 0] = A_r^*[j, 0] + 1/2 < A_r'[j, 0]$ (because $(r, j, 0) \notin Q$ and $A_r^*[j, 0], A_r'[j, 0]$ are integers), which is at most $\#A_q[j, 0] = \#\hat{A}_q[j, 0]$ (because $(q, j, 0) \in Q$); thus, even in this case we have $\hat{A}_r[j, 0] \leq \hat{A}_q[j, 0]$. We conclude that, f violates a $0j$ constraint of $\hat{B}_q$, i.e., $-v_j > \hat{A}_q[0, j]$ for some j.

We distinguish two cases depending on whether the exit point is closed (i.e., it is f) or it is open ($t(f)$). Suppose that the exit point is f. Since $v_i + \epsilon > \hat{A}_q[i, 0]$ for all $\epsilon > 0$, we have $v_i \geq \hat{A}_q[i, 0]$ and therefore, $w_i - w_j = v_i - v_j > \hat{A}_q[i, 0] + \hat{A}_q[0, j] \geq \hat{A}_q[i, j]$. Suppose that the exit point is $t(f)$. Since $-w_j + \epsilon > \hat{A}_q[0, j]$ for all $\epsilon > 0$, we have $-w_j \geq \hat{A}_q[0, j]$, and therefore again $w_i - w_j > \hat{A}_q[i, 0] + \hat{A}_q[0, j] \geq \hat{A}_q[i, j]$. Thus, in either case $t(f)$ violates the ij constraint of $\hat{B}_q$, with $i, j \neq 0$, a contradiction.

Finally, suppose that $t(f)$ violates the $0j$ constraint of $\hat{B}_q$, i.e., $-w_j > \hat{A}_q[0, j]$. We will show first that the containment $t(B_r^*) \subseteq B_q^*$ implies $\#A_r^*[i, j] \leq \#A_r^*[i, 0] + \#A_q^*[0, j]$ for all $i, j \neq 0$. To see this, consider the trajectory from a point p of B_r^* with $x_i - x_j$ equal to $A_r^*[i, j]$ if it is weak, or arbitrarily close to it if it is strict, and the exit of this trajectory from B_r^*; say $(s, z - \epsilon) \in B_r^*$ and $(s, z) \in B_q^*$. We have $z_i - \epsilon \leq A_r^*[i, 0]$ and $-z_j \leq A_q^*[0, j]$, thus $A_r^*[i, j] - 2\epsilon \leq z_i - z_j - \epsilon \leq A_r^*[i, 0] + A_q^*[0, j]$ for all $\epsilon > 0$.

Therefore $\#A_r^*[i, j] \leq \#A_r^*[i, 0] + \#A_q^*[0, j]$. Note furthermore that in the case of equality, if $A_r^*[i, j]$ is weak (thus, equal to $z_i - z_j$) and $A_r^*[i, 0]$ is strict (hence $z_i \leq \#A_r^*[i, 0]$), then $-z_j \geq \#A_r^*[i, j] - \#A_r^*[i, 0] = \#A_q^*[0, j]$, and therefore, $A_q^*[0, j]$ must be weak.

We show now the analogous inequality for $\hat{A}_r$, $\hat{A}_q$. This is implied immediately by the inequality for A_r^*, A_r^*, unless $(r, i, j) \notin Q$, and both $(r, i, 0), (q, 0, j) \in Q$. In this case we have $\#\hat{A}_r[i, j] \leq \#A_r'[i, j]$, which by the analogous inequality due to $t(B_r') \subseteq B_q$ is bounded from above by $\#A_r[i, 0] + \#A_q[0, j] = \#\hat{A}_r[i, 0] + \#\hat{A}_q[0, j]$. Thus, in all cases we have $\#\hat{A}_r[i, j] \leq \#\hat{A}_r[i, 0] + \#\hat{A}_q[0, j]$.

Consider the trajectory of f and let i be the index of an upper bound hyperplane that it crosses to exit $\hat{B}_r$. That is, either the exit point is closed, $\hat{A}_r[i, 0]$ is weak and $w_i > \#\hat{A}_r[i, 0]$, or the exit point is open, $\hat{A}_r[i, 0]$ is strict and $w_i = \#\hat{A}_r[i, 0]$. Since $w_i - w_j \leq \hat{A}_r[i, j]$, we conclude that $-w_j \leq \#\hat{A}_r[i, j] - \#\hat{A}_r[i, 0] \leq \#\hat{A}_q[0, j]$ and the first inequality is strict unless $w_i - w_j = \#\hat{A}_r[i, j]$ (hence $\hat{A}_r[i, j]$ is weak) and $w_i = \#\hat{A}_r[i, 0]$ (hence $\hat{A}_r[i, 0]$ is strict). Since the entries of corresponding $\hat{A}$ and A^* matrices have the same type, we conclude that $-w_j \leq \#\hat{A}_q[0, j]$ and moreover in case of equality, $\hat{A}_q[0, j]$ is weak. Thus, in any case $-w_j \leq \hat{A}_q[0, j]$. □

Thus, after $O(k^2 N)$ rounds, we will have $Q = R$, i.e., all the entries of the matrices have the correct value except possibly for the type. From this point on, every subsequent round will change at least one entry from weak to strict (thus, there are at most $k^2 N$ more rounds), until a round does not produce any change, at which point the (reachable) partition is stable and we have the minimal reachable system. We remark that the $O(k^2 N)$ worst-case upper bound on the number of rounds for termination, is only what we were able to prove. We do not know in fact if it is tight, i.e., it is possible that the process terminates always faster.

Summarizing, we have:

Theorem 6.2. *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, our algorithm constructs the minimal reachable system in time polynomial in the input and the output (the number of reachable blocks). We obtained our algorithm by adapting the general method of [19] to the class of timed systems. This required the efficient implementation of the block operations, and the addressing of the termination problem. A critical factor in achieving efficiency was to stay with a representation of the blocks as zones, i.e., *conjunctions* of constraints, carefully avoiding the explicit application of the difference operator that would turn conjunctions into disjunctions. This is generally possible for deterministic transition systems, and although timed systems are not completely deterministic because of the time transition, they are almost deterministic. As for termination, we proved that no explicit measure was needed because our round-robin method of splitting blocks leads automatically to polynomial-time termination in this case.

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, for example, if the initial partition contains differences of zones, then we cannot guarantee polynomial time in the number of reachable blocks; in fact we can show an NP-hardness result in this regard. Specifically, the following problem is NP-hard: given a timed graph G with n nodes, an initial configuration p_0 and an initial partition π where all the blocks are differences of zones, determine whether the minimal reachable system has more than n blocks. NP-hardness holds even in quite restricted cases, for example, for acyclic timed graphs with all constants 0 or 1, and all blocks of the initial partition being zones except for one block that is a difference $Z_1 - Z_2$ with Z_2 a singleton. This implies that if $\mathrm{P} \neq \mathrm{NP}$, then it is impossible to construct the minimal reachable graph in polynomial time in its number of nodes and edges. The problem in this case is that having nonconvex sets (differences of zones) leads to an exponential increase in the size of the representation of the blocks. That is, even if there is a small number of reachable blocks, their descriptive complexity may be exponentially large.

Notes

1. The same results apply if we allow transitions to reset clocks to arbitrary integer values; we stay with resets to 0 for simplicity.
2. 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, and they are uniquely determined by the initial partition.
3. [2] implemented also an algorithm that uses some ideas from [19], but which lacks the minimality and the performance guarantees of [19]: both the output of the algorithm and its time complexity depend on nondeterministic choices made in the calls to the difference routine, thus, the algorithm may compute a nonminimal system and it may run in exponential time. 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].
4. Alternatively, we can first apply Step 2a on all the unprocessed marked leaves (in level order) to make them into zones, and then, regard all the marked leaves as being at the same, maximum, level, and split them in round robin order from that point on.

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*, Vol. LNCS 630, pp. 341–354, Springer-Verlag, 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*, Vol. LNCS 443, pp. 322–335, Springer-Verlag, 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, pp. 85–91, ACM-AMS, 1990.
7. A. Bouajjani, J. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel, “Minimal state graph generation,” in *Science of Computer Programming*, Vol. 18, No. 3, 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*, Vol. LNCS, Springer-Verlag, 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, pp. 207–218, ACM-AMS, 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*, Vol. LNCS, Springer-Verlag, 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.), Vol. LNCS 407, Springer Verlag, 1989.
15. T. Henzinger, X. Nicollin, J. Sifakis, 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*, pp. 264–274, 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. Milner, *Communication and Concurrency*, Prentice-Hall Intl., London, UK, 1989.
22. R. Paige and R. Tarjan, “Three partition refinement algorithms,” in *SIAM J. on Computing*, Vol. 16, pp. 973–989, 1987.
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-Toi, personal communication, 1992.