A Boolean Approach to Unbounded, Fully Symbolic Model Checking of Timed Automata

Sanjit A. Seshia Randal E. Bryant

March 2003
CMU-CS-03-117

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A shorter version of this paper will appear at CAV '03.

Abstract

We present a new approach to unbounded, fully symbolic model checking of timed automata that is based on an efficient translation of quantified separation logic to quantified Boolean logic. Our technique preserves the interpretation of clocks over the reals and can check any property expressed in the timed μ calculus. The core operations of eliminating quantifiers over real variables and deciding separation logic are respectively translated to eliminating quantifiers on Boolean variables and checking Boolean satisfiability (SAT). We can thus leverage well-known techniques for Boolean formulas, including Binary Decision Diagrams (BDDs) and recent advances in SAT and SAT-based quantifier elimination. We present preliminary empirical results for a BDD-based implementation of our method.

This research was supported in part by a National Defense Science and Engineering Graduate Fellowship and by ARO grant DAAD19-01-1-0485.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained in this document are those of the authors, and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Defense or the U.S. Government.

Keywords: Timed automata, Model checking, Boolean logic, Separation logic, Quantifier elimination

1 Introduction

Timed automata [2] have proved to be a useful formalism for modeling real-time systems. A timed automaton is a generalization of a finite automaton with a set of real-valued clock variables. The state space of a timed automaton thus has a finite component (over Boolean state variables) and an infinite component (over clock variables). Several model checking techniques for timed automata have been proposed over the past decade. These can be classified, on the one hand, as being either *symbolic* or *fully symbolic*, and on the other, as being *bounded* or *unbounded*. Symbolic techniques use a symbolic representation for the infinite component of the state space, and either symbolic or explicit representations for the finite component. In contrast, fully symbolic methods employ a single symbolic representation for both finite and infinite components of the state space. Bounded model checking techniques work by unfolding the transition relation d times, finding counterexamples of length up to d, if they exist. As in the untimed case, these methods suffer from the limitation that, unless a bound on the length of counterexamples is known, they cannot verify the property of interest. Unbounded methods, on the other hand, can produce a guarantee of correctness.

The theoretical foundation for unbounded, fully symbolic model checking of timed automata was laid by Henzinger et al. [11]. The characteristic function of a set of states is a formula in *separation logic*, a quantifier-free fragment of first-order logic. Formulas in Separation Logic (SL) are Boolean combinations of Boolean variables and predicates of the form $x_i \bowtie x_j + c$ where $\bowtie \in \{>, \geq\}$, x_i and x_j are real-valued variables, and c is a constant. *Quantified Separation Logic* (QSL) is an extension of SL with quantifiers over real and Boolean variables. The most important model checking operations involve deciding SL formulas and eliminating quantifiers on real variables from QSL formulas.

In this paper, we present the first approach to unbounded, fully symbolic model checking of timed automata that is based on a Boolean encoding of SL formulas and that preserves the interpretation of clocks over the reals. Unlike many other fully symbolic techniques, our method can be used to model check any property in the timed μ calculus or Timed Computation Tree Logic (TCTL) [3]. The main theoretical contribution of this paper is a new technique for transforming the problem of eliminating quantifiers on real variables to one of eliminating quantifiers on Boolean variables. In some cases, we can avoid introducing Boolean quantification altogether. These techniques, in conjunction with previous work on deciding SL formulas via a translation to Boolean satisfiability (SAT) [17], allow us to leverage well-known techniques for manipulating quantified Boolean formulas, including Binary Decision Diagrams (BDDs) and recent work on SAT and SAT-based quantifier elimination [13].

Related Work. The work that is most closely related to ours is the approach based on representing SL formulas using Difference Decision Diagrams (DDDs) [14]. A DDD is a BDD-like data structure, where the node labels are generalized to be separation predicates rather than just Boolean variables, with the ordering of predicates induced by an ordering of clock variables. This predicate ordering permits the use of local reduction operations, such as eliminating inconsistent combinations of two predicates that involve the same pair of clock variables. Deciding a SL formula represented as a DDD is done by eliminating all inconsistent paths in the DDD. This is done by enumerating all paths in the DDD and checking the satisfiability of the conjunction of predicates on each path using a constraint solver based on the Bellman-Ford shortest path algorithm. Note that each path can be viewed as a disjunct in the Disjunctive Normal Form (DNF) representation of the DDD, and in the worst case there can be exponentially many calls to the constraint solver. Quantifier elimination is performed by the Fourier-Motzkin technique [10], which also requires enumerating all possible paths. In contrast, our Boolean encoding method is general in that any representation of Boolean functions may be used. Our decision procedure and quantifier elimination scheme use a direct translation to SAT and Boolean quantification, respectively, avoiding the need to explicitly enumerate each DNF term. In theory, the use of DDDs permits unbounded, fully symbolic model checking of TCTL; however, the DDD-based model checker [14] can only check reachability properties (these can express safety and bounded-liveness properties [1]).

UPPAAL2K and KRONOS are unbounded, symbolic model checkers that explicitly enumerate the discrete component of the state space. KRONOS uses Difference Bound Matrices (DBMs) as the symbolic representation [19] of the infinite component. UPPAAL2K uses, in addition, Clock Difference Diagrams (CDDs) to symbolically represent unions of convex clock regions [6]. In a CDD, a node is labeled by the difference of a pair of clock variables, and each outgoing edge from a node is labeled with an interval bounding that difference. Note that while KRONOS can check arbitrary TCTL formulas, UPPAAL2K is limited to checking reachability properties and very restricted liveness properties such as $\mathbf{AF}p$.

RED is an unbounded, fully symbolic model checker based on a data structure called the Clock Restriction Diagram (CRD) [18]. The CRD is similar to a CDD, labeling each node with the difference between two clock variables. However, each outgoing edge from a node is labeled with an upper bound, instead of an interval. RED represents separation formulas by a combined BDD-CRD structure, and can model check TCTL formulas.

A fully symbolic version of KRONOS using BDDs has been developed by interpreting clock variables over integers [8]; however, this approach is restricted to checking reachability for the subclass of closed timed automata[1], and the encoding blows up with the size of the integer constants. Rabbit [7] is a tool based on this approach that additionally exploits compositional methods to find good BDD variable orderings. In comparison, our technique applies to all timed automata and its efficiency is far less sensitive to the size of constants. Also, the variable ordering methods used in Rabbit could be used in a BDD-based implementation of our technique.

Many fully symbolic, but bounded model checking methods based on SAT have been developed recently (e.g., [5, 15]). These algorithms cannot be directly extended to perform unbounded model checking.

The rest of the paper is organized as follows. We define notation and present background material in Sections 2 and 3. We describe our new contributions in Sections 4 and 5. We conclude in Section 6 with experimental results and ongoing work.

2 Background

We begin with a brief presentation of background material, based on papers by Alur [2] and Henzinger et al. [11]. We refer the reader to these papers for details.

2.1 Separation Logic

Separation logic (SL), also known as *difference logic*, is a quantifier-free fragment of first-order logic. A formula ϕ in separation logic is a Boolean combination of Boolean variables and *separation predicates* (also known as *difference bound constraints*) involving real-valued variables, as given by the following grammar:

$$\phi \quad ::= \quad \mathbf{true} \mid \mathbf{false} \mid b \mid \neg\phi \mid \phi \wedge \phi \mid x_i \geq x_j + c \mid x_i > x_j + c$$

[1] Clock constraints in a closed timed automaton do not contain strict inequalities.

We use a special variable x_0 to denote the constant 0; this allows us to express bounds of the form $x \geq c$. We will however use both $x \bowtie c$ and $x \bowtie x_0 + c$, where $\bowtie \in \{>, \geq\}$, as suits the context. We will denote Boolean variables by $b, b_1, b_2, \ldots$, real variables by $x, x_1, x_2, \ldots$, and SL formulas by $\phi, \phi_1, \phi_2, \ldots$. Note that the relations $>$ and $\geq$ suffice to represent equalities and other inequalities.

Characteristic functions of sets of states of timed automata are SL formulas. Deciding the satisfiability of a SL formula is NP-complete [11].

Quantified Separation Logic. Separation logic can be generalized by the addition of quantifiers over both Boolean and real variables. This yields *quantified separation logic* (QSL). The satisfiability problem for QSL is PSPACE-complete [12]. We will denote QSL formulas by $\omega, \omega_1, \ldots$.

2.2 Timed Automata

A *timed automaton* $\mathcal{T}$ is a tuple $\langle \mathcal{L}, \mathcal{L}_0, \Sigma, \mathcal{X}, \mathcal{I}, \mathcal{E} \rangle$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{L}_0 \subseteq \mathcal{L}$ is a finite set of initial locations, Σ is a finite set of labels used for product construction, $\mathcal{X}$ is a finite set of non-negative real-valued clock variables, $\mathcal{I}$ is a function mapping a location to a SL formula (called a *location invariant*), and $\mathcal{E}$ is the transition relation, a subset of $\mathcal{L} \times \Psi \times \mathcal{R} \times \Sigma \times \mathcal{L}$, where Ψ is a set of SL formulas that form enabling *guard* conditions for each transition, and $\mathcal{R}$ is a set of *clock reset assignments*. A location invariant is the condition under which the system can stay in that location. A clock reset assignment is of the form $x_i := x_0 + c$ or $x_i := x_j$, where $x_i, x_j \in \mathcal{X}$ and c is an integer constant,[2] and indicates that the clock variable on the left-hand side of the assignment is reset to the value of the expression on the right-hand side. We will denote guards by $\psi, \psi_1, \ldots$.

Two timed automata are composed by synchronizing over common labels. We refer the reader to Alur's paper [2] for a formal definition of product construction. Note that in contrast to the definition of timed automata given by Alur [2], we allow location invariants and guards to be arbitrary SL formulas, rather than simply conjunctions over separation predicates involving clock variables.

The invariant $\mathcal{I}_{\mathcal{T}}$ for the timed automaton $\mathcal{T}$ is defined as $\mathcal{I}_{\mathcal{T}} = \bigwedge_{l \in \mathcal{L}} [enc(l) \implies \mathcal{I}(l)]$, where $enc(l)$ denotes the Boolean encoding of location l. We will also denote a transition $t \in \mathcal{E}$ as $\psi \implies A$, where ψ is a guard condition over both Boolean state variables (used to encode locations) and clock variables of the system, and A is a set of assignments to clock and Boolean state variables.

2.3 Timed μ Calculus and TCTL

We express properties of timed automata in a generalization of the μ calculus called the *timed μ* ($\mathbf{T}\mu$) calculus. A formula φ of the $\mathbf{T}\mu$ calculus is generated by the following grammar:

$$\varphi ::= X \mid \phi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \triangleright \varphi_2 \mid z.\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

z is a *specification clock variable* (i.e., $z \notin \mathcal{X}$) and X is a *formula variable* used in fixpoint computation. The formula $\varphi_1 \triangleright \varphi_2$ means that the formula φ_1 is true at the present state, and remains true (as time elapses) until some transition is taken, at which time formula φ_2 becomes true; thus "$\triangleright$" is essentially a next-state operator. The formula $z.\varphi$ is true in a state where φ is true after setting specification clock variable z to zero. The expression $\mu X.\varphi$ stands for the least fixpoint of φ, where X is a formula variable bound inside φ; ν denotes the greatest fixpoint operator.

Henzinger et al. [11] show that the $\mathbf{T}\mu$ calculus can express the dense-real-time version of Computation Tree Logic (CTL), Timed CTL (TCTL) [3]. TCTL generalizes CTL by allowing

[2] The assignment $x_i := c$ is represented as $x_i := x_0 + c$. Wherever we use x_i to denote a clock variable, $i > 0$.

atomic propositions to be any SL formula, and in addition contains formulas of the form $z.\varphi$ where z is a specification clock variable and φ is a TCTL formula in which z appears free; the latter class enables one to write time-bounded properties. We omit the details for brevity.

Several model checkers are specialized to check reachability properties. Using the notation of the $\mathbf{T}\mu$ calculus, a reachability property is a formula of the form

$$\phi_{init} \implies \neg\mu X.[\phi_{err} \vee (\mathbf{true} \triangleright X)]$$

where ϕ_{init} is the initial set of states, and ϕ_{err} characterizes the bad states; the formula evaluates to **true** if no error state is reachable from any initial state.

3 Fully Symbolic Model Checking

We use a model checking algorithm given by Henzinger et al. [11]. This algorithm checks that a timed automaton $\mathcal{T}$ satisfies a specification given as a $\mathbf{T}\mu$ formula φ. The algorithm always terminates, and generates a SL formula $|\varphi|$, such that, if $\mathcal{T}$ is *non-zeno* (i.e., time can diverge from any state), then $|\varphi|$ is equivalent to $\mathcal{I}_{\mathcal{T}}$.

The algorithm is fully symbolic since it avoids the need to enumerate locations by representing sets of values of both Boolean state variables and clock variables as SL formulas. It performs backward exploration of the state space and uses the following three special operators over SL formulas:

1. **Time Elapse:** $\phi_1 \rightsquigarrow \phi_2$ denotes the set of all states that can reach the state set ϕ_2 by allowing time to elapse, while staying in state set ϕ_1 at all times in between. Formally,

$$\phi_1 \rightsquigarrow \phi_2 \doteq \exists\delta\{\delta \geq x_0 \wedge \phi_2 + \delta \wedge \forall\epsilon[x_0 \leq \epsilon \leq \delta \implies \phi_1 + \epsilon]\} \tag{1}$$

   where $\phi + \delta$ denotes the formula obtained by adding δ to all clock variables occurring in ϕ, computed as $\phi[x_i + \delta/x_i, 1 \leq i \leq n]$, where $x_1, x_2, \ldots, x_n$ are the clock variables in ϕ_i (i.e., not including the zero variable x_0).

2. **Assignment:** $\phi[A]$, where A is a set of assignments, denotes the formula obtained by simultaneously substituting in ϕ the right hand side of each assignment in A for the left hand side. Formally, if A is the list $b_1 := \phi_1, \ldots, b_k := \phi_k, x_1 := x_{j_1} + c_1, \ldots, x_n := x_{j_n} + c_n$, where each b_i is a Boolean variable, each x_j is a clock variable, and for each x_{j_l}, $j_l = 0$ or $c_l = 0$, then

$$\phi[A] = \phi[\phi_1/b_1, \ldots, \phi_k/b_k, x_{j_1} + c_1/x_1, \ldots, x_{j_n} + c_n/x_n]$$

   Assignments are thus performed via substitutions of variables.

3. **Weakest Pre-condition:** $pre_{\mathcal{T}}\phi$ denotes the weakest precondition of ϕ with respect to the timed automaton $\mathcal{T}$. Formally,

$$pre_{\mathcal{T}}\phi = \mathcal{I}_{\mathcal{T}} \wedge (\phi \vee \bigvee_{t \in \mathcal{E}} pre_t(\mathcal{I}_{\mathcal{T}} \wedge \phi))$$

   where for a transition $t = \psi \implies A$

$$pre_t(\phi) = \psi \wedge \phi[A]$$

   Note that $pre_{\mathcal{T}}$ is defined using assignments and Boolean operations.

The model checking algorithm is defined inductively on the structure of $\mathbf{T}\mu$ formulas:

- $|\phi| := \mathcal{I}_{\mathcal{T}} \wedge \phi$
- $|\neg\varphi| := \mathcal{I}_{\mathcal{T}} \wedge \neg|\varphi|$
- $|\varphi_1 \vee \varphi_2| := |\varphi_1| \vee |\varphi_2|$
- $|\varphi_1 \triangleright \varphi_2| := |(|\varphi_1| \vee |\varphi_2|) \rightsquigarrow pre_{\mathcal{T}}(|\varphi_2|)|$
- $|z.\varphi| := |\varphi|[z := 0]$
- $|\mu X.\varphi|$ is the result of the following iteration:

    $\phi_{new} :=$ **false**;
    repeat
        $\phi_{old} := \phi_{new}$;
        $\phi_{new} := |\varphi[X := \phi_{old}]|$;
    until($\phi_{new} \implies \phi_{old}$);
    returnϕ_{old};

As can be seen from the algorithm description above, apart from Boolean operators, the main components of the algorithm are: quantifier elimination in the time elapse operation, substitution of state variables in an assignment, and the decision procedure used to check containment in fixpoint computation. For a fully symbolic model checker that represents state sets as SL formulas, these model checking operators can be defined as operations in QSL. We elaborate below.

Time Elapse. Consider the formula on the right hand side of Equation 1, the definition of the time elapse operator. This formula is not in QSL, since it includes expressions that are the sum of two real variables (e.g., $x + \delta$). However, it can be transformed to a QSL formula, by using instead of δ and ϵ, variables $\overline{\delta}$ and $\overline{\epsilon}$ that represent their negations:

$$\exists\overline{\delta}\{\overline{\delta} \leq x_0 \wedge \phi_2 + (-\overline{\delta}) \wedge \forall\overline{\epsilon}[\overline{\delta} \leq \overline{\epsilon} \leq x_0 \implies \phi_1 + (-\overline{\epsilon})]\} \tag{2}$$

Formula 2 is expressible in QSL, since the substitution $\phi[x_i + (-\overline{\delta})/x_i, 1 \leq i \leq n]$ can be computed as $\phi[\overline{\delta}/x_0]$.[3] This yields,

$$\exists\overline{\delta}\{\overline{\delta} \leq x_0 \wedge \phi_2[\overline{\delta}/x_0] \wedge \forall\overline{\epsilon}(\overline{\delta} \leq \overline{\epsilon} \leq x_0 \implies \phi_1[\overline{\epsilon}/x_0])\} \tag{3}$$

Finally, we can rewrite Formula 3 purely in terms of existential quantifiers:

$$\exists\overline{\delta}\{\overline{\delta} \leq x_0 \wedge \phi_2[\overline{\delta}/x_0] \wedge \neg\exists\overline{\epsilon}(\overline{\epsilon} \leq x_0 \wedge \overline{\delta} \leq \overline{\epsilon} \wedge \neg\phi_1[\overline{\epsilon}/x_0])\} \tag{4}$$

A procedure for performing the time elapse operation therefore requires one for eliminating (existential) quantifiers over real variables from a SL formula.

Checking Containment. Containment of one set of states, ϕ_{new}, in another, ϕ_{old}, is checked by deciding the validity of the SL formula $\phi = \phi_{new} \implies \phi_{old}$ (or equivalently, the satisfiability of $\neg\phi$). There are several procedures that can decide separation formulas (e.g., [17, 4, 15]).

[3] Note that substituting x_0 by $\overline{\delta}$ or $\overline{\epsilon}$ can be viewed as shifting the zero reference point to a more negative value, thus increasing the value of any clock variable relative to zero (e.g., [5, 14]).

4 Model Checking Operations using Boolean Encoding

We now show how to implement the fundamental model checking operations using a Boolean encoding of separation predicates. We first describe how our encoding allows us to replace quantification of real variables by quantification of Boolean variables. This builds on previous work on deciding a SL formula by transformation to a Boolean formula [17]. We then show how we represent SL formulas as Boolean formulas, allowing the model checking operations to be implemented as operations in Quantified Boolean Logic (QBL), and enabling the use of QBL packages, e.g., a BDD package.

In the remainder of this section, we will use ϕ to denote a SL formula over real variables $x_1, x_2, \ldots, x_n$, and Boolean variables $b_1, b_2, \ldots, b_k$. Also, let $\bowtie, \bowtie_1, \bowtie_2 \in \{>, \geq\}$.

4.1 From Real Quantification to Boolean Quantification

Consider the QSL formula $\omega_a \doteq \exists x_a.\phi$, where $a \in [1..n]$.

We transform ω_a to an equivalent QSL formula ω_{bool} with quantifiers over only Boolean variables in the following three steps:

1. *Encode separation predicates:*

   Consider each separation predicate in ϕ of the form $x_i \bowtie x_j + c$ where either $i = a$ or $j = a$. For each such predicate, we generate a corresponding Boolean variable $e^{\bowtie,c}_{i,j}$. Separation predicates that are negations of each other are represented by Boolean literals (true or complemented variables) that are negations of each other; however, for ease of presentation, we will extend the naming convention for Boolean variables to Boolean literals, writing $e^{>,-c}_{j,i}$ for the negation of $e^{\geq,c}_{i,j}$.

   Let the added Boolean variables be $e^{\bowtie_{i_1},c_{i_1}}_{i_1,a}, e^{\bowtie_{i_2},c_{i_2}}_{i_2,a}, \ldots, e^{\bowtie_{i_m},c_{i_m}}_{i_m,a}$ for the upper bounds on x_a, and $e^{\bowtie_{j_1},c_{j_1}}_{a,j_1}, e^{\bowtie_{j_2},c_{j_2}}_{a,j_2}, \ldots, e^{\bowtie_{j_{m'}},c_{j_{m'}}}_{a,j_{m'}}$ for the lower bounds on it.

   We replace each predicate $x_a \bowtie x_j + c$ (or $x_i \bowtie x_a + c$) in ϕ by the corresponding Boolean variable $e^{\bowtie,c}_{a,j}$ (or $e^{\bowtie,c}_{i,a}$). Let the resulting SL formula be ϕ^a_{bool}.

2. *Add transitivity constraints:*

   Notice that there can be assignments to the $e^{\bowtie,c}_{i,a}$ and $e^{\bowtie,c}_{a,j}$ variables that have no corresponding assignment to the real valued variables. To disallow such assignments, we place constraints on these added Boolean variables. Each constraint is generated from two Boolean literals that encode predicates containing x_a. We will refer to these constraints as *transitivity constraints for x_a*.

   A transitivity constraint for x_a has one of the following types:

   (a) $e^{\bowtie_1,c_1}_{i,a} \wedge e^{\bowtie_2,c_2}_{a,j} \implies (x_i \bowtie x_j + c_1 + c_2)$,
   where if $\bowtie_1 = \bowtie_2$, then $\bowtie = \bowtie_1$, otherwise, we must duplicate this constraint for both $\bowtie = \bowtie_1$ and for $\bowtie = \bowtie_2$.

   (b) $e^{\bowtie_1,c_1}_{i,j} \implies e^{\bowtie_2,c_2}_{i,j}$, where $c_1 > c_2$ and either $i = a$ or $j = a$.

   (c) $e^{>,c}_{i,j} \implies e^{\geq,c}_{i,j}$, where either $i = a$ or $j = a$.

Note that a constraint of type (a) involves a separation predicate ($x_i \bowtie x_j + c_1 + c_2$). This predicate might not be present in the original formula ϕ.[4]

After generating all transitivity constraints for x_a, we conjoin them to get the SL formula ϕ^a_{cons}.

3. Finally, generate the QSL formula ω_{bool} given below:

$$\exists e^{\bowtie_{i_1},c_{i_1}}_{i_1,a}, e^{\bowtie_{i_2},c_{i_2}}_{i_2,a}, \ldots, e^{\bowtie_{i_m},c_{i_m}}_{i_m,a}.\exists e^{\bowtie_{j_1},c_{j_1}}_{a,j_1}, e^{\bowtie_{j_2},c_{j_2}}_{a,j_2}, \ldots, e^{\bowtie_{j_{m'}},c_{j_{m'}}}_{a,j_{m'}}.[\phi^a_{cons} \wedge \phi^a_{bool}]$$

We formalize the correctness of this transformation in the following theorem.

Theorem 1 *ω_a and ω_{bool} are equivalent.*

Proof: To show that ω_a and ω_{bool} are equivalent, we show that $\omega_a \implies \omega_{bool}$ and $\omega_{bool} \implies \omega_a$.

Denote the formula $\omega_a \implies \omega_{bool}$ by ω^1 and $\omega_{bool} \implies \omega_a$ by ω^2. Note first that the free variables in both implications are the real-valued variables $x_1, x_2, \ldots, x_{a-1}, x_{a+1}, \ldots, x_n$ and the Boolean variables $b_1, b_2, \ldots, b_k$. For all i and j, the values assigned to x_i and b_j by an assignment A are denoted by $A[x_i]$ and $A[b_j]$ respectively.

1. We first show that ω^1 is valid.

   Let A denote an arbitrary assignment to all free variables and to the bound real variable x_a in ω_a such that $A[\omega_a] = \mathbf{true}$. We extend A with an assignment to the Boolean variables $e^{\bowtie_{i_1},c_{i_1}}_{i_1,a}, e^{\bowtie_{i_2},c_{i_2}}_{i_2,a}, \ldots, e^{\bowtie_{i_m},c_{i_m}}_{i_m,a}$ and $e^{\bowtie_{j_1},c_{j_1}}_{a,j_1}, e^{\bowtie_{j_2},c_{j_2}}_{a,j_2}, \ldots, e^{\bowtie_{j_{m'}},c_{j_{m'}}}_{a,j_{m'}}$, such that $A[\omega_{bool}] = \mathbf{true}$ and hence $A[\omega^1] = \mathbf{true}$.

   Define an evaluation of the newly added Boolean variables according to the following rules:

$$A[e^{c,\bowtie}_{a,j}] = A[x_a \bowtie x_j + c] \qquad \forall j \neq a, \text{ for all constants } c \text{ and relations } \bowtie \tag{5}$$
$$A[e^{c,\bowtie}_{i,a}] = A[x_i \bowtie x_a + c] \qquad \forall i \neq a, \text{ for all constants } c \text{ and relations } \bowtie \tag{6}$$

   Since $A[\omega_a] = \mathbf{true}$, $A[\phi] = \mathbf{true}$. Further, using Equations 5 and 6, we can conclude that $A[\phi^a_{bool}] = A[\phi]$ because ϕ^a_{bool} is obtained from ϕ by replacing predicates $(x_a \bowtie x_j + c)$ and $(x_i \bowtie x_a + c')$ (for all i, j and for all constants c, c') with Boolean variables $e^{c,\bowtie}_{a,j}$ and $e^{c',\bowtie}_{i,a}$. Therefore, $A[\phi^a_{bool}] = \mathbf{true}$.

   To show that $A[\omega_{bool}] = \mathbf{true}$, we need to additionally show that $A[\phi^a_{cons}] = \mathbf{true}$. We consider an arbitrary transitivity constraint of each type:

   (a) $e^{\bowtie_1,c_1}_{i,a} \wedge e^{\bowtie_2,c_2}_{a,j} \implies (x_i \bowtie x_j + c_1 + c_2)$.
   Suppose $A[e^{\bowtie_1,c_1}_{i,a}] = A[e^{\bowtie_2,c_2}_{a,j}] = \mathbf{true}$. Then, by Equations 5 and 6, we conclude that $A[x_i] \bowtie_1 A[x_a] + c_1$ and $A[x_a] \bowtie_2 A[x_j] + c_2$. If $\bowtie_1 = \bowtie_2 = \bowtie$, we can infer $A[x_i] \bowtie A[x_j] + c_1 + c_2$, and thus $A[x_i \bowtie x_j + c_1 + c_2] = \mathbf{true}$. If $\bowtie_1 \neq \bowtie_2$, then we can infer $A[x_i \bowtie_1 x_j + c_1 + c_2] = A[x_i \bowtie_2 x_j + c_1 + c_2] = \mathbf{true}$.

   (b) $e^{\bowtie_1,c_1}_{i,j} \implies e^{\bowtie_2,c_2}_{i,j}$, where $c_1 > c_2$ and either $i = a$ or $j = a$.
   Suppose $A[e^{\bowtie_1,c_1}_{i,j}] = \mathbf{true}$. Then, by Equations 5 and 6, $A[x_i \bowtie_1 x_j + c_1] = \mathbf{true}$. Since $c_1 > c_2$, $A[x_i \bowtie_2 x_j + c_2] = \mathbf{true}$, and hence $A[e^{\bowtie_2,c_2}_{i,j}] = \mathbf{true}$.

[4] This addition is analogous to the "tightening" step performed in difference-bound matrix based algorithms

(c) $e^{>,c}_{i,j} \implies e^{\geq,c}_{i,j}$, where either $i = a$ or $j = a$.

Exactly as for type (b) constraints, $A[e^{>,c}_{i,j}] = A[x_i > x_j + c] = \textbf{true}$. Therefore, $A[x_i \geq x_j + c] = \textbf{true}$ and hence $A[e^{\geq,c}_{i,j}] = \textbf{true}$.

Thus, A satisfies all transitivity constraints, and hence $A[\phi^a_{cons}] = \textbf{true}$, completing the proof for the first part.

2. We now show that ω^2 is valid.

Let A denote an arbitrary assignment to all free variables and to the bound Boolean variables in ω_{bool} such that $A[\omega_{bool}] = \textbf{true}$. We extend A with an evaluation of x_a such that $A[\omega_a] = \textbf{true}$ and hence $A[\omega^2] = \textbf{true}$.

Since $A[\omega_{bool}] = \textbf{true}$, we know that $A[\phi^a_{cons}] = \textbf{true}$ (i.e., the transitivity constraints are satisfied by A) and $A[\phi^a_{bool}] = \textbf{true}$.

Suppose we can find a value $A[x_a]$ that satisfies the following equations:

$$A[x_a \bowtie x_j + c] = A[e^{c,\bowtie}_{a,j}] \qquad \forall j \neq a, \forall \text{ constants } c \tag{7}$$

$$A[x_i \bowtie x_a + c] = A[e^{c,\bowtie}_{i,a}] \qquad \forall i \neq a, \forall \text{ constants } c \tag{8}$$

Then, $A[\phi^a_{bool}] = A[\phi]$ because ϕ^a_{bool} is obtained from ϕ by replacing predicates $(x_a \bowtie x_j + c)$ and $(x_i \bowtie x_a + c')$ (for all i, j and for all constants c, c') with Boolean variables $e^{c,\bowtie}_{a,j}$ and $e^{c',\bowtie}_{i,a}$. Since $A[\phi^a_{bool}] = \textbf{true}$, $A[\phi] = \textbf{true}$, and hence $A[\omega_a] = \textbf{true}$.

A value $A[x_a]$ that satisfies Equations 7 and 8 exists if:

$$A[x_a] \geq A[x_j] + c \qquad \text{if } A[e^{c,\geq}_{a,j}] = \textbf{true} \tag{9}$$

$$A[x_a] < A[x_j] + c \qquad \text{if } A[e^{c,\geq}_{a,j}] = \textbf{false} \tag{10}$$

$$A[x_a] > A[x_j] + c \qquad \text{if } A[e^{c,>}_{a,j}] = \textbf{true} \tag{11}$$

$$A[x_a] \leq A[x_j] + c \qquad \text{if } A[e^{c,>}_{a,j}] = \textbf{false} \tag{12}$$

In the above equations, w.l.o.g., we use literals encoding lower bounds on x_a (e.g., $e^{c,\geq}_{a,j}$) in place of those encoding upper bounds (e.g., $e^{-c,>}_{j,a}$).

Let

$$U_a = \min_{j,c \text{ s.t. } e^{c,\bowtie}_{a,j} = \textbf{false}} (A[x_j] + c)$$

and

$$L_a = \max_{j,c \text{ s.t. } e^{c,\bowtie}_{a,j} = \textbf{true}} (A[x_j] + c)$$

U_a and L_a are respectively the tightest upper and lower bounds on $A[x_a]$.

Define the ordering relation $\diamond$ as follows

$$\diamond = \begin{cases} \geq & \text{if the tightest bounds are non-strict, i.e., } A[x_a] \leq U_a \text{ and } A[x_a] \geq L_a \\ > & \text{otherwise} \end{cases} \tag{13}$$

Then, the inequalities 9 to 12 can be satisfied if:

$$U_a \diamond L_a \tag{14}$$

In other words, if the minimum upper bound on $A[x_a]$ is greater (or greater than or equal to) the maximum lower bound on $A[x_a]$.

To show that the above is true, it is enough to show that for any pair of upper and lower bounds on $A[x_a]$, the relation $\diamond$ holds, and so it holds in particular for the minimum upper bound and the maximum lower bound. For example, for the two inequalities $A[x_a] < A[x_j]+c_1$ and $A[x_a] \geq A[x_k] + c_2$ to be true we need that $A[x_j] + c_1 > A[x_k] + c_2$.

Therefore, consider two arbitrary indices j and k different from a. We need to consider four cases based on evaluations of the Boolean literals $e_{a,j}^{c_1,\bowtie}$ and $e_{a,k}^{c_2,\bowtie}$. Note that cases in which both literals evaluate to **true** or both to **false** only give rise to two lower bounds or to two upper bounds. By the transitivity constraints of types (b) and (c), if the minimum upper bound (or maximum lower bound) is satisfied, then every other upper bound (or lower bound) will be satisfied.

The four cases are enumerated below:

(a) $e_{a,j}^{c_1,>} =$ **false**, $e_{a,k}^{c_2,\geq} =$ **true**.
This implies that

$$A[x_j] \geq A[x_a] - c_1 \text{ and } A[x_a] \geq A[x_k] + c_2$$

We need to show that

$$A[x_j] + c_1 \geq A[x_k] + c_2$$

Or

$$A[x_j] \geq A[x_k] + (c_2 - c_1)$$

The last inequality is true, since A satisfies the transitivity constraint $e_{j,a}^{-c_1,\geq} \wedge e_{a,k}^{c_2,\geq} \implies (x_j \geq x_k + c_2 - c_1)$.

(b) $e_{a,j}^{c_1,\geq} =$ **false**, $e_{a,k}^{c_2,>} =$ **true**.
This case is identical to the one above, with $\geq$ and $>$ interchanged.

(c) $e_{a,j}^{c_1,>} =$ **false**, $e_{a,k}^{c_2,>} =$ **true**.
This implies that

$$A[x_j] \geq A[x_a] - c_1 \text{ and } A[x_a] > A[x_k] + c_2$$

We need to show that

$$A[x_j] + c_1 > A[x_k] + c_2$$

Or

$$A[x_j] > A[x_k] + (c_2 - c_1)$$

The last inequality is true, since A satisfies the transitivity constraint $e_{j,a}^{-c_1,\geq} \wedge e_{a,k}^{c_2,>} \implies (x_j > x_k + c_2 - c_1)$.

(d) $e_{a,j}^{c_1,\geq} =$ **false**, $e_{a,k}^{c_2,\geq} =$ **true**.
This case is identical to the one above, with $\geq$ and $>$ interchanged.

Thus, we can conclude that Equation 14 is satisfied, and that completes the proof of this part.

$\square$

Example 1 *Let* $\omega_a = \exists x_a.\phi$ *where* $\phi = x_a \leq x_0 \wedge x_1 \geq x_a \wedge x_2 \leq x_a$. *Then,* $\phi_{bool}^a = e_{0,a}^{\geq,0} \wedge e_{1,a}^{\geq,0} \wedge e_{a,2}^{\geq,0}$. ϕ_{cons}^a *is the conjunction of the following constraints:*

1. $e^{\geq,0}_{0,a} \wedge e^{\geq,0}_{a,2} \implies x_0 \geq x_2$

2. $e^{\geq,0}_{1,a} \wedge e^{\geq,0}_{a,2} \implies x_1 \geq x_2$

Then, $\omega_{bool} = \exists e^{\geq,0}_{0,a}, e^{\geq,0}_{1,a}, e^{\geq,0}_{a,2}.[\phi^a_{cons} \wedge \phi^a_{bool}]$ *evaluates to* $x_0 \geq x_2 \wedge x_1 \geq x_2$. □

The quantifier transformation procedure described here works even when ϕ is replaced by a QSL formula with quantifiers only over Boolean variables.

Note also that the transformation procedure we present here differs from the one presented by Strichman et al. [17] in that the latter is concerned with preserving satisfiability only, whereas the former must produce an equivalent formula that preserves all satisfying assignments to the free variables.

4.2 Deciding SL formulas

Suppose we want to decide the satisfiability of ϕ. Note that ϕ is satisfiable iff the QSL formula $\omega_{1..n} = \exists x_1, x_2, \ldots, x_n.\phi$ is.

Using Theorem 1, we can transform $\omega_{1..n}$ to an equivalent QSL formula ω_{bool} with existential quantifiers only over Boolean variables encoding all separation predicates. As ω_{bool} is a QBL formula with only existential quantifiers, we can simply discard the quantifiers and use a Boolean satisfiability checker to decide the resulting Boolean formula.

Note that the procedure described above can be viewed as one way to implement the procedure of Strichman et al. [17].

4.3 Representing SL Formulas as Boolean Formulas

In our presentation up to this point, we have not used any specific representation of SL formulas. In practice, we encode a SL formula ϕ as a Boolean formula β. The encoding is performed as follows. Consider each separation predicate $x_i \bowtie x_j + c$ in ϕ. As in Section 4.1 earlier, we introduce a Boolean variable $e^{\bowtie,c}_{i,j}$ for $x_i \bowtie x_j + c$, only this time we do it for every single separation predicate. Also as before, separation predicates that are negations of each other are represented by Boolean literals that are negations of each other. We then replace each separation predicate in ϕ by its corresponding Boolean literal. The resulting Boolean formula is β.

Clearly, β, by itself, stores insufficient information for generating transitivity constraints. Therefore, we also store the 1-1 mapping of separation predicates to the Boolean literals that encode them. However, this mapping is used only lazily, i.e., when generating transitivity constraints during quantification and in deciding SL formulas.

4.3.1 Substitution.

Given the representation described above, we can implement substitution of a clock variable as follows. For a clock variable x_i, we perform the substitution $[x_i \leftarrow x_k + d]$ (where $k = 0$ or $d = 0$), by replacing all Boolean variables of the form $e^{\bowtie,c}_{i,j}$ and $e^{\bowtie',c'}_{j,i}$, for all j, by variables $e^{\bowtie,c-d}_{k,j}$ and $e^{\bowtie',c'+d}_{j,k}$ respectively, creating fresh replacement variables if necessary. Substitution of a Boolean state variable by the Boolean encoding of a separation formula is done by Boolean function composition.

5 Optimizations

The methods presented in Section 4 can be optimized in a few ways. First, we can be more selective in deciding when to add transitivity constraints. Second, we can compute the time elapse operator more efficiently by a method that does not explicitly introduce the bound real variable $\overline{\epsilon}$, and hence does not introduce new quantifiers over Boolean variables. The third optimization concerns eliminating paths in a BDD representation that violate transitivity constraints. As is well-known, the size of a BDD is very sensitive to the number and ordering of BDD variables. In the case of model checking timed automata, new Boolean variables are created as the model checking proceeds, while generating transitivity constraints, and while performing substitutions of clock variables. This has the potential to add several BDD variables on each iteration. Finally, we can use an over-approximation technique to to reduce the number of BDD variables added on each model checking iteration. While all four techniques presented in this section help in reducing the number of BDD variables, only the last two are specialized for BDDs.

5.1 Determining if Bounds are Conjoined

Suppose ϕ is a SL formula with Boolean encoding β, and we wish to eliminate the quantifier in $\exists x_a.\phi$. As described in Section 4.1, a transitivity constraint for x_a involves two Boolean literals that encode separation predicates involving x_a. For a syntactic representation of β, as the number of constraints grows, so does the size of $[\beta^a_{cons} \wedge \beta^a_{bool}]$, the Boolean encoding of $[\phi^a_{cons} \wedge \phi^a_{bool}]$. Further, new separation predicates can be added when a transitivity constraint is generated from an upper bound and a lower bound on x_a. For a BDD-based implementation, this corresponds to the addition of a new BDD variable. We would therefore like to avoid adding transitivity constraints wherever possible.

In fact, we only need to add a constraint involving an upper bound literal and a lower bound literal if they are conjoined in a minimized DNF representation of β.[5] From a geometric viewpoint, this means that we check that the predicates corresponding to the two literals are bounds for the same convex clock region. This check can be posed as a Boolean satisfiability problem, which is easily solved using a BDD representation of β. Let the literals be e_1 and e_2. Then, we use cofactoring and Boolean operations to compute the following Boolean formula:

$$e_1 \wedge e_2 \wedge [\beta|_{e_1=\mathbf{true}} \wedge \neg(\beta|_{e_1=\mathbf{false}})] \wedge [\beta|_{e_2=\mathbf{true}} \wedge \neg(\beta|_{e_2=\mathbf{false}})] \tag{15}$$

Consider the subformula $e_i \wedge [\beta|_{e_i=\mathbf{true}} \wedge \neg(\beta|_{e_i=\mathbf{false}})]$ for $i = 1, 2$. This formula represents the set of input combinations $\overline{e}$ in which e_i must be set to **true** in order for $\beta(\overline{e})$ to evaluate to **true**. Thus, the conjunction of the subformulas for $i = 1$ and $i = 2$ is satisfiable only if there exists a non-empty set of input combinations $\overline{e}$ in which both e_1 and e_2 must be set to **true** for $\beta(\overline{e})$ to evaluate to **true**. Viewed alternately, Formula 15 expresses the Boolean function corresponding to the disjunction of all terms in the minimized DNF representation of β that contain both e_1 and e_2 in true form. Therefore, if Formula 15 is satisfiable, it means that e_1 and e_2 are conjoined, and we must add a transitivity constraint involving them both.

Note however, that since β does not, by itself, represent the original SL formula ϕ, finding that e_1 and e_2 are conjoined in β does not imply that they are bounds in the same convex region of ϕ. However, the converse is true, so our method is sound.

[5] A conservative, syntactic variant of this idea has been proposed earlier by Strichman [16].

5.2 Quantifier Elimination by Eliminating Upper Bounds on x_0

The definition of the time elapse operation introduces two quantified non-clock real variables: $\overline{\delta}$ and $\overline{\epsilon}$. We can exploit the special structure of the QSL formula for the time elapse operation so as to avoid introducing $\overline{\epsilon}$ altogether. Thus, we can avoid adding new quantified Boolean variables encoding predicates involving $\overline{\epsilon}$.

Consider the inner existentially quantified SL formula in Formula 4 in Section 3, reproduced here:

$$\exists\overline{\epsilon}(\overline{\epsilon} \leq x_0 \wedge \overline{\delta} \leq \overline{\epsilon} \wedge \neg\phi_1[\overline{\epsilon}/x_0])$$

Grouping the inequality $\overline{\delta} \leq \overline{\epsilon}$ with the formula $\neg\phi_1[\overline{\epsilon}/x_0]$, we get:

$$\exists\overline{\epsilon}\{\overline{\epsilon} \leq x_0 \wedge (\overline{\delta} \leq x_0 \wedge \neg\phi_1)[\overline{\epsilon}/x_0]\} \tag{16}$$

Finally, treating $\overline{\delta}$ as a clock variable, we can revert back to ϵ from $\overline{\epsilon}$, transforming Formula 16 to the following form:

$$\exists\epsilon[\epsilon \geq x_0 \wedge (\overline{\delta} \leq x_0 \wedge \neg\phi_1) + \epsilon] \tag{17}$$

Formula 17 is a special case of the formula ω_ϵ given by

$$\omega_\epsilon = \exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon$$

for some SL formula ϕ. From a geometric viewpoint, ϕ is a region in $\mathbf{R^n}$ and ω_ϵ is the shadow of ϕ for a light source at ∞^n. Examples of ϕ and the corresponding ω_ϵ are shown in Figures 1(a) and 1(c) respectively.

We can transform ω_ϵ to an equivalent SL formula ϕ_{ub} by eliminating upper bounds on x_0, i.e., Boolean variables of the form $e_{i,0}^{\bowtie,c}$. The transformation is performed iteratively in the following steps:

1. Let $\phi_0 = \phi$. Let $e_{i_1,0}^{\bowtie_1,c_1}, e_{i_2,0}^{\bowtie_2,c_2}, \ldots, e_{i_m,0}^{\bowtie_m,c_m}$ be Boolean literals encoding all upper bounds on x_0 that occur in ϕ.

   Note that an upper bound literal $e_{i_j,0}^{\bowtie_j,c_j}$ occurs in ϕ, if it appears in some term in the minimized DNF representation of ϕ. This can be checked by evaluating the Boolean function $[\beta|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{true}} \wedge \neg(\beta|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{false}})]$, where β is the Boolean encoding of ϕ, and checking that it is not **false**.

2. For $j = 1, 2, \ldots, m$, we construct ϕ_j as follows:

   (a) Replace all occurrences of $x_{i_j} \bowtie_j x_0 + c_j$ in ϕ_{j-1} with $e_{i_j,0}^{\bowtie_j,c_j}$ to get $\phi_{bool}^{0,j-1}$.

   (b) Construct $\phi_{cons}^{0,j-1}$, the conjunction of all transitivity constraints[6] for x_0 involving $e_{i_j,0}^{\bowtie_j,c_j}$ and clock variables in $\phi_{bool}^{0,j-1}$.

   (c) Construct the formula ϕ_j, a disjunction of two terms:

   $$\phi_j = \{(\phi_{bool}^{0,j-1} \wedge \phi_{cons}^{0,j-1})|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{true}}\} \vee \{[\neg(x_{i_j} \bowtie_j x_0 + c_j)] \wedge [\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{false}}]\}$$

   The first disjunct is the region obtained by dropping the bound $x_{i_j} \bowtie_j x_0 + c_j$ from convex sub-regions of ϕ_{j-1} where it is a lower bound on x_{i_j}, while letting time elapse backward. The second disjunct corresponds to sub-regions where $\neg(x_{i_j} \bowtie_j x_0 + c_j)$ is an upper bound; these regions are left unchanged.

[6] We can use the optimization technique of Section 5.1 in this step.

The output of the above transformation, ϕ_{ub}, is given by $\phi_{ub} = \phi_m$. The correctness of this procedure is formalized in the following theorem.

Theorem 2 *ω_ϵ and ϕ_{ub} are equivalent.*

Proof: We make use of the following lemmas.

Lemma 1 *For all $j = 1, \ldots, m$, $\exists \epsilon . \epsilon \geq x_0 \wedge \phi_{j-1} + \epsilon$ is equivalent to $\exists \epsilon . \epsilon \geq x_0 \wedge \phi_j + \epsilon$.*

Proof:(Lemma 1)
We give the proof for an arbitrary j satisfying $1 \leq j \leq m$. Let ω_{j-1} and ω_j respectively denote $\exists \epsilon_{j-1} . \epsilon_{j-1} \geq x_0 \wedge \phi_{j-1} + \epsilon_{j-1}$ and $\exists \epsilon_j . \epsilon_j \geq x_0 \wedge \phi_j + \epsilon_j$. Notice that we have renamed the bound variable ϵ.

1. First, we show that $\omega_{j-1} \implies \omega_j$. Let A be an assignment to the free and bound variables in ω_{j-1} such that $A[\omega_{j-1}] = \mathbf{true}$. This means that $A[\phi_{j-1} + \epsilon_{j-1}] = \mathbf{true}$. Extend A so that $A[\epsilon_j] = A[\epsilon_{j-1}]$. Thus, $A[\epsilon_{j-1} \geq x_0] = A[\epsilon_j \geq x_0] = \mathbf{true}$.

   We consider two cases.

   (a) *Case 1:* $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \mathbf{true}$.
   Note that by construction,

   $$\phi_{bool}^{0,j-1} = \phi_{j-1}[e_{i_j,0}^{\bowtie_j,c_j} / (x_{i_j} \bowtie_j x_0 + c_j)]$$

   From the two equalities above, and since $A[\epsilon_j] = A[\epsilon_{j-1}]$, we get

   $$A[\phi_{j-1} + \epsilon_{j-1}] = A[\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j} = \mathbf{true}} + \epsilon_j]$$

   In addition, the transitivity constraints are satisfied, i.e.,

   $$A[\phi_{cons}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j} = \mathbf{true}} + \epsilon_j] = \mathbf{true}$$

   because $\phi_{cons}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j} = \mathbf{true}} + \epsilon_j$ only involves real-valued variables. Therefore,

   $$A[\phi_{j-1} + \epsilon_{j-1}] = A[(\phi_{bool}^{0,j-1} \wedge \phi_{cons}^{0,j-1})|_{e_{i_j,0}^{\bowtie_j,c_j} = \mathbf{true}} + \epsilon_j]$$

   Thus, we conclude that

   $$A[\phi_{j-1} + \epsilon_{j-1}] = A[\phi_j + \epsilon_j] = \mathbf{true}$$

   which in turn implies that

   $$A[\epsilon_{j-1} \geq x_0 \wedge \phi_{j-1} + \epsilon_{j-1}] = A[\epsilon_j \geq x_0 \wedge \phi_j + \epsilon_j] = \mathbf{true}$$

   and so

   $$A[\omega_{j-1}] = A[\omega_j] = \mathbf{true}$$

   This concludes the first case.

(b) *Case 2:* $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \textbf{false}$.

Since

$$\phi_{bool}^{0,j-1} = \phi_{j-1}[e_{i_j,0}^{\bowtie_j,c_j}/(x_{i_j} \bowtie_j x_0 + c_j)]$$

and, in addition, $A[\epsilon_j] = A[\epsilon_{j-1}]$, we have

$$A[\phi_{j-1} + \epsilon_{j-1}] = A[\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{false}} + \epsilon_j]$$

Now, since $A[\phi_{j-1} + \epsilon_{j-1}] = \textbf{true}$, we get

$$A[\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{false}} + \epsilon_j] = \textbf{true}$$

and

$$A[[\neg(x_{i_j} \bowtie_j x_0 + c_j) \wedge \phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{false}}] + \epsilon_j] = \textbf{true}$$

and so, we conclude that

$$A[\phi_j + \epsilon_j] = A[\epsilon_j \geq x_0 \wedge \phi_j + \epsilon_j] = A[\omega_j] = \textbf{true}$$

which concludes case 2.

Thus, $\omega_{j-1} \implies \omega_j$.

2. We next show that $\omega_j \implies \omega_{j-1}$.

Let A be an assignment to the free and bound variables in ω_j such that $A[\omega_j] = \textbf{true}$. This means that $A[\phi_j + \epsilon_j] = \textbf{true}$. We wish to extend A by an assignment to ϵ_{j-1} so that $A[\phi_{j-1} + \epsilon_{j-1}] = \textbf{true}$ and $A[\epsilon_{j-1} \geq x_0] = \textbf{true}$.

We consider two cases.

(a) *Case 1:* $A[(\phi_{bool}^{0,j-1} \wedge \phi_{cons}^{0,j-1})|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{true}} + \epsilon_j] = \textbf{true}$.

Therefore,

$$A[\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{true}} + \epsilon_j] = \textbf{true} \tag{18}$$

and

$$A[\phi_{cons}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\textbf{true}} + \epsilon_j] = \textbf{true}$$

If $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j] = \textbf{true}$, then using the equality

$$\phi_{bool}^{0,j-1} = \phi_{j-1}[e_{i_j,0}^{\bowtie_j,c_j}/(x_{i_j} \bowtie_j x_0 + c_j)] \tag{19}$$

we can set $A[\epsilon_{j-1}] = A[\epsilon_j]$, which yields $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \textbf{true}$, and so using Equations 18 and 19, we get

$$A[\phi_{j-1} + \epsilon_{j-1}] = A[\phi_j + \epsilon_j] = \textbf{true} \tag{20}$$

However, if $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j] = \textbf{false}$, then we must find an alternate assignment to ϵ_{j-1}, such that $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \textbf{true}$. Then, we can conclude, as above, that Equation 20 holds.

Consider, w.r.t. the assignment A, all lower bounds on x_0 that occur in $\phi_{j-1} + \epsilon_j$ (and hence in $\phi_{bool}^{0,j-1} + \epsilon_j$); more precisely, a lower bound on x_0 is a predicate $(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j$ such that $A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j] = \mathbf{true}$.
If no such lower bound on x_0 exists, then we can set ϵ_{j-1} to any value that results in $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \mathbf{true}$, because there is no lower bound to be violated by increasing the value of a clock variable.
So suppose at least one lower bound on x_0 exists in ϕ_{j-1}. Define the value v_s as

$$v_s = \min_{k \text{ s.t. } A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j] = \mathbf{true}} (-c_k - A[x_{i_k} + \epsilon_j]) \tag{21}$$

Note that $v_s \geq 0$ since $A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j] = \mathbf{true}$ for all k in Equation 21.
Let l be the k for which the minimum on the right-hand side of Equation 21 is attained. If there are many such k, say $k_1, k_2, \ldots, k_d$, set l according to the following rules:

i. If there exists k_i for which $\bowtie_{k_i} = >$, set l to any one such k_i.
ii. Otherwise select l to be any one of $k_1, k_2, \ldots, k_d$.

Thus,

$$v_s = -c_l - A[x_{i_l} + \epsilon_j] \tag{22}$$

Next, we define a positive real number χ as follows:

$$\chi = \begin{cases} \chi_0 & \text{if } \bowtie_l = >\text{, and where } \chi_0 \in (0, A[x_{i_j} - x_{i_l} - c_j - c_l]) \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

Note that $A[x_{i_j} - x_{i_l} - c_j - c_l]$ is non-negative and is strictly positive when $\bowtie_l = >$. This is because there exists a transitivity constraint in $\phi_{cons}^{0,j-1}$ of the form

$$(e_{i_j,0}^{\bowtie_j, c_j} \wedge x_0 \bowtie_l x_{i_l} + c_l) \implies (x_{i_j} \bowtie_j x_{i_l} + c_j + c_l)$$

which occurs in $\phi_{cons}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j, c_j} = \mathbf{true}}$ as

$$(x_0 \bowtie_l x_{i_l} + c_l) \implies (x_{i_j} \bowtie_j x_{i_l} + c_j + c_l)$$

If $\bowtie_j \neq \bowtie_l$, the following constraint also holds:

$$(x_0 \bowtie_l x_{i_l} + c_l) \implies (x_{i_j} \bowtie_l x_{i_l} + c_j + c_l)$$

Since $A[(x_0 \bowtie_l x_{i_l} + c_l) + \epsilon_j] = \mathbf{true}$, the following equalities hold:

$$A[(x_{i_j} \bowtie_j x_{i_l} + c_j + c_l) + \epsilon_j] = A[x_{i_j} \bowtie_j x_{i_l} + c_j + c_l] = \mathbf{true} \tag{24}$$

$$A[(x_{i_j} \bowtie_l x_{i_l} + c_j + c_l) + \epsilon_j] = A[x_{i_j} \bowtie_l x_{i_l} + c_j + c_l] = \mathbf{true} \tag{25}$$

Thus, $A[x_{i_j} - x_{i_l} - c_j - c_l]$ is non-negative and is strictly positive when $\bowtie_l = >$.
We now show that $v_s - \chi \geq 0$. If $\chi = 0$, clearly $v_s - \chi \geq 0$. So, assume that $\bowtie_l = >$, and thus $\chi \in (0, A[x_{i_j} - x_{i_l} - c_j - c_l])$. Then we can conclude the following:

$$\begin{aligned} v_s - \chi &= -c_l - A[x_{i_l}] - A[\epsilon_j] - \chi \\ &> -c_l - A[x_{i_l}] - A[\epsilon_j] - A[x_{i_j} - x_{i_l} - c_j - c_l] \\ &= -c_l - A[x_{i_l}] - A[\epsilon_j] - A[x_{i_j}] + A[x_{i_l}] + c_j + c_l \\ &= c_j - A[x_{i_j}] - A[\epsilon_j] \\ &\geq 0 \quad (\text{since } A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j] = \mathbf{false}) \end{aligned}$$

Intuitively, $v_s - \chi$ is a non-negative real number we can add to all clock variables without violating lower bounds on x_0 in $\phi_{j-1} + \epsilon_j$.

Now, define $A[\epsilon_{j-1}]$ as follows:

$$A[\epsilon_{j-1}] = A[\epsilon_j] + v_s - \chi \tag{26}$$

Since $v_s - \chi \geq 0$, $A[\epsilon_{j-1}] \geq A[\epsilon_j]$.

Given the above assignment to ϵ_{j-1}, we first show that $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \mathbf{true}$. We have the following sequence of equalities:

$$\begin{array}{ll} & A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] \\ = & A[x_{i_j}] + A[\epsilon_{j-1}] \bowtie_j c_j \\ = & A[x_{i_j}] \bowtie_j c_j - A[\epsilon_{j-1}] \\ = & A[x_{i_j}] \bowtie_j c_j - v_s + \chi - A[\epsilon_j] \\ = & A[x_{i_j}] \bowtie_j \chi + c_j - \min\limits_k(-A[x_{i_k} + \epsilon_j] - c_k) - A[\epsilon_j] \\ = & A[x_{i_j}] \bowtie_j \chi + c_j + (A[x_{i_l} + \epsilon_j] + c_l) - A[\epsilon_j] \\ = & A[x_{i_j}] \bowtie_j \chi + A[x_{i_l}] + c_j + c_l \\ = & \mathbf{true} \quad \text{(since } \chi \in (0, A[x_j - x_l - c_j - c_l]) \text{ and from Eqn. 24)} \end{array}$$

We next show that the assignment to ϵ_{j-1} in Equation 26 preserves the truth assignment to other bounds on x_0; i.e., bounds in $\phi_{j-1} + \epsilon_j$ other than $(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j$. Formally, we show that for all bounds $x_0 \bowtie_k x_{i_k} + c_k$ where $k \neq j$:

$$A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_{j-1}] = A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j]$$

Note that the value of separation predicates of the form $x_{i_{k_1}} \bowtie x_{i_{k_2}} + c_{k_1 k_2}$ is unaffected by the assignment to ϵ_j or ϵ_{j-1}.

If $A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j] = \mathbf{false}$, then $A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_{j-1}] = \mathbf{false}$, since $A[\epsilon_{j-1}] \geq A[\epsilon_j]$.

On the other hand, if $A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_j] = \mathbf{true}$, then

$$\begin{array}{ll} & A[(x_0 \bowtie_k x_{i_k} + c_k) + \epsilon_{j-1}] \\ = & 0 \bowtie_k A[x_{i_k}] + c_k + A[\epsilon_{j-1}] \\ = & 0 \bowtie_k A[x_{i_k}] + c_k + A[\epsilon_j] + v_s - \chi \\ = & 0 \bowtie_k (c_k + A[x_{i_k}]) + A[\epsilon_j] + (-c_l - A[x_{i_l} + \epsilon_j]) - \chi \\ = & (-c_k - A[x_{i_k}]) \bowtie_k (-c_l - A[x_{i_l}]) - \chi \\ = & \mathbf{true} \quad \text{(since } \chi \geq 0 \text{ and from Equations 21 and 22)} \end{array}$$

To sum up, we have shown that $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_{j-1}] = \mathbf{true}$, even though $A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j] = \mathbf{false}$. Thus, we can conclude that

$$A[\phi_{j-1} + \epsilon_{j-1}] = A[\phi_j + \epsilon_j] = \mathbf{true}$$

This completes the proof for the first case.

(b) *Case 2:* $A[[\neg(x_{i_j} \bowtie_j x_0 + c_j) \wedge \phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{false}}] + \epsilon_j] = \mathbf{true}$.

Thus

$$A[\phi_{bool}^{0,j-1}|_{e_{i_j,0}^{\bowtie_j,c_j}=\mathbf{false}} + \epsilon_j] = \mathbf{true}$$

and

$$A[(x_{i_j} \bowtie_j x_0 + c_j) + \epsilon_j] = \mathbf{false}$$

Letting $A[\epsilon_{j-1}] = A[\epsilon_j]$ and from Equation 19, we get

$$A[\phi_{j-1} + \epsilon_{j-1}] = \mathbf{true}$$

as required.

Thus, $\omega_j \implies \omega_{j-1}$.

From parts 1 and 2 above, we conclude that ω_{j-1} and ω_j are equivalent.
□

Lemma 2 *Suppose the SL formula ϕ does not contain any separation predicates that are upper bounds on x_0; i.e., any satisfying assignment to ϕ sets all upper bounds on x_0 to* **false***, and all lower bound predicates to* **true***. Then, $\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon$ is equivalent to ϕ.*

Proof:(Lemma 2)
We first show that $\phi \implies (\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon)$.

Let A be an assignment to the variables in ϕ such that $A[\phi] = \mathbf{true}$. We extend A with an evaluation of ϵ so that $A[\epsilon] = 0 = A[x_0]$. Then, $A[\epsilon \geq x_0 \wedge \phi + \epsilon] = \mathbf{true}$, since $A[\phi + \epsilon] = A[\phi]$. Therefore, $A[\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon] = \mathbf{true}$. Thus, $\phi \implies (\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon)$.

Next, we show that $(\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon) \implies \phi$. Let A be an assignment such that $A[\exists\epsilon.\epsilon \geq x_0 \wedge \phi + \epsilon] = \mathbf{true}$. Thus, $A[\epsilon \geq x_0] = \mathbf{true}$ and $A[\phi + \epsilon] = \mathbf{true}$. Since ϕ does not contain any separation predicates that are upper bounds on x_0, for any lower bound $x_0 \bowtie_k x_k + c_k$ on x_0, $A[(x_0 \bowtie_k x_k + c_k) + \epsilon] = \mathbf{true}$ and for an upper bound $x_l \bowtie_l x_0 + c_l$ on x_0, $A[(x_l \bowtie_l x_0 + c_l) + \epsilon] = \mathbf{false}$.

Then, since $A[\epsilon] \geq 0$,

$$A[(x_0 \bowtie_k x_k + c_k) + \epsilon] = \mathbf{true} = A[x_0 \bowtie_k (x_k + \epsilon) + c_k] = A[x_0 \bowtie_k x_k + c_k]$$

Similarly, for an upper bound predicate on x_0, $A[x_l \bowtie_l x_0 + c_l] = \mathbf{false}$.

It then follows that $A[\phi] = \mathbf{true}$.
□

From Lemma 1, we infer that $\omega_\epsilon = \exists\epsilon.\epsilon \geq x_0 \wedge \phi_0 + \epsilon$ is equivalent to $\exists\epsilon.\epsilon \geq x_0 \wedge \phi_m + \epsilon$. Additionally, since ϕ_m does not contain any upper bounds on x_0, using Lemma 2, we conclude that ω_ϵ is equivalent to $\phi_m = \phi_{ub}$. This completes the proof of Theorem 2. □

Example 2 *Let the subformula ϕ of ω_ϵ be*

$$\phi = (x_1 \geq x_0 + 3 \wedge x_2 \leq x_0 + 2) \vee (x_1 < x_0 + 3 \wedge x_2 \geq x_0 + 3)$$

ϕ is depicted geometrically as the shaded region in Figure 1(a). It comprises two sub-regions, one for each disjunct. The lower bounds on these regions, $x_1 \geq x_0 + 3$ and $x_2 \geq x_0 + 3$, are upper bounds on x_0. We encode these by $e_{1,0}^{\geq,3}$ and $e_{2,0}^{\geq,3}$.

Figure 1(b) shows ϕ_1, the result of eliminating $e_{1,0}^{\geq,3}$. Formally, we calculate

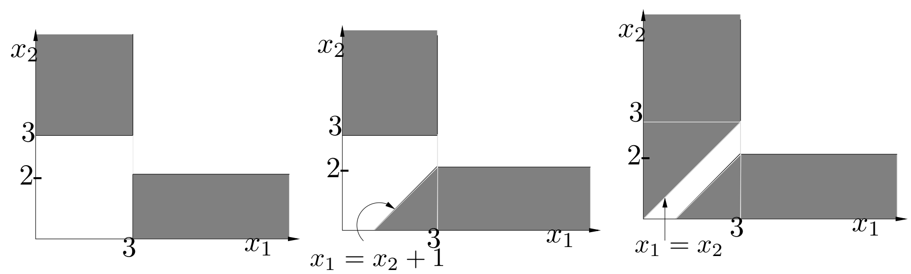
$$\begin{aligned}\phi_{bool}^{0,0} &= (e_{1,0}^{\geq,3} \wedge x_2 \leq x_0 + 2) \vee (\neg e_{1,0}^{\geq,3} \wedge x_2 \geq x_0 + 3)\\ \phi_{cons}^{0,0} &= (e_{1,0}^{\geq,3} \wedge x_2 \leq x_0 + 2) \implies (x_1 \geq x_2 + 1)\end{aligned}$$

Then, applying step 2(c) of the transformation, we get

$$\phi_1 = (x_2 \leq x_0 + 2 \wedge x_1 \geq x_2 + 1) \vee (x_1 < x_0 + 3 \wedge x_2 \geq x_0 + 3)$$

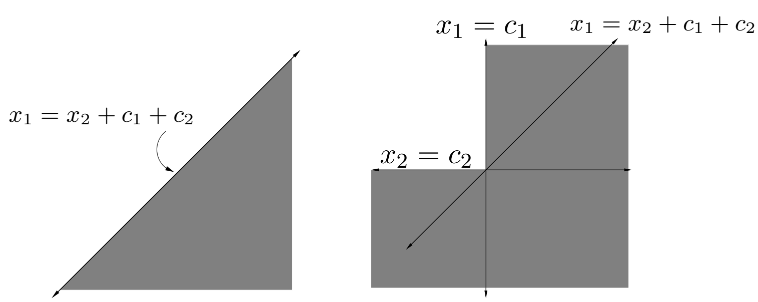
Similarly, in the next iteration, we introduce and eliminate $e_{2,0}^{\geq,3}$ to get ϕ_2, shown in Figure 1(c), which is equivalent to ω_ϵ. $\square$

Note that the QSL formula obtained after eliminating the inner quantifier in Formula 4 is not of the form ω_ϵ, and so we cannot avoid introducing the $\overline{\delta}$ variable.



(a) $\phi_0 = \phi$ (b) ϕ_1 (c) $\phi_2 = \omega_\epsilon$

Figure 1: **Eliminating upper bounds on x_0**



(a) Exact (b) Weakened

Figure 2: **Weakening Transitivity Constraints.** The shaded area denotes the region satisfying the constraint.

5.3 Overapproximation by Weakening Transitivity Constraints

In spite of the methods of Sections 5.1 and 5.2, generating transitivity constraints while eliminating the quantifier on $\overline{\delta}$ might create too many new BDD variables, causing the BDD to blow up. In the case of reachability properties, a partial solution is to weaken the transitivity constraints added so as to not create new variables, yielding an overapproximation of the time elapse operation. For reachability properties, overapproximating the time elapse ("pre") operation makes our model checking procedure incomplete, but retains soundness.

Consider a transitivity constraint for $\overline{\delta}$ of type (a) as defined in Section 4.1, reproduced below:

$$e_{i,\overline{\delta}}^{\bowtie_1,c_1} \wedge e_{\overline{\delta},j}^{\bowtie_2,c_2} \implies (x_i \bowtie x_j + c_1 + c_2)$$

We replace the above constraint by the following weakened constraint:

$$e_{i,\overline{\delta}}^{\bowtie_1,c_1} \wedge e_{\overline{\delta},j}^{\bowtie_2,c_2} \implies [(x_i \bowtie_1 x_0 + c_1) \vee (x_0 \bowtie_2 x_j + c_2)] \tag{27}$$

The difference between the two constraints is depicted geometrically in Figure 2.

Note that the consequent of the weakened constraint (Formula 27) only involves separation predicates involving x_i, x_j, and x_0, and these already occurred in formula ϕ of Formula 4, since they are the predicates in which $\overline{\delta}$ was substituted for x_0. Thus, we avoid adding new BDD variables.

5.4 Eliminating Infeasible Paths in BDDs

Suppose β is the Boolean encoding of SL formula ϕ. Let ϕ_{cons} denote the conjunction of transitivity constraints for all real-valued variables in ϕ, and let β_{cons} denote its Boolean encoding. Finally, denote the BDD representations of β and β_{cons} by $\mathsf{Bdd}(\beta)$ and $\mathsf{Bdd}(\beta_{cons})$ respectively.

We would like to eliminate paths in $\mathsf{Bdd}(\beta)$ that violate transitivity constraints, i.e., those corresponding to assignments to variables in β for which $\beta_{cons} =$ **false**. We can do this by using the BDD `Restrict` operator, replacing $\mathsf{Bdd}(\beta)$ by `Restrict(`$\mathsf{Bdd}(\beta), \mathsf{Bdd}(\beta_{cons})$`)`. Informally, `Restrict(`$\mathsf{Bdd}(\beta)$`, `$\mathsf{Bdd}(\beta_{cons})$`)` traverses $\mathsf{Bdd}(\beta)$, eliminating a path on which β_{cons} is **false** as long as it doesn't involve adding new nodes to the resulting BDD. Details about the `Restrict` operator may be found in the paper by Coudert and Madre [9].

Since eliminating infeasible paths in a large BDD can be quite time consuming, we only apply this optimization to the BDD for the set of reachable states, once on each fixpoint iteration.

6 Experimental Results

We implemented a model checker that uses BDDs to represent Boolean functions and incorporates all the optimizations described in Section 5. The model checker is written in the OCaml language and uses the CUDD package[7] for BDD manipulation. We have performed preliminary experiments comparing the performance of our model checker for both reachability and non-reachability TCTL properties, without using the over-approximation scheme of Section 5.3. For reachability properties, we compare against the other unbounded, fully symbolic model checkers, viz., a DDD-based checker (DDD) [14] and RED version 4.1 [18], which have been shown to outperform UPPAAL2K and KRONOS for reachability analysis. For non-reachability properties, such as checking that a system is non-zeno, we compare against KRONOS and RED, the only other unbounded model checkers that check such properties.

As an illustrative example, we use Fischer's protocol for mutual exclusion. Tools such as DDD and RED that we compare against have been shown to perform well on this example for reachability properties. The automaton for the ith process in this protocol is shown in Figure 3. We ran two

[7] `http://vlsi.colorado.edu/~fabio/CUDD`

experiments with this example. The first experiment compared our model checker against DDD and RED, checking that the system preserves mutual exclusion (a reachability property). In the second experiment, we compared against KRONOS and RED for checking that the product automaton is non-zeno (a non-reachability property). All experiments were run on a notebook computer with a 1 GHz Pentium-III processor and 128 MB RAM, running Linux. We ran DDD, KRONOS, and RED with their default options. For our implementation, we turned off dynamic variable reordering in CUDD. To come up with a static variable ordering, we classified the BDD variables in our Boolean encoding as follows. The first class, $C_{\mathtt{id}}$, consists of variables encoding the shared integer `id`. For each i, class $C(i)$ contains the BDD variables encoding locations and clock constraints for process i. Finally, class $C(i,j)$ encodes predicates relating clock variables from processes i and j. We used a static variable ordering that groups together variables in the same class, places class $C_{\mathtt{id}}$ at the top, orders $C(i)$ before $C(j)$ if $i < j$, and places $C(i,j)$ right after $C(j)$ for $j > i$. New BDD variables added during model checking are inserted into the order at positions that depend upon the class they fall into. The same static variable order was used for the corresponding Boolean variables and separation predicates in DDD.

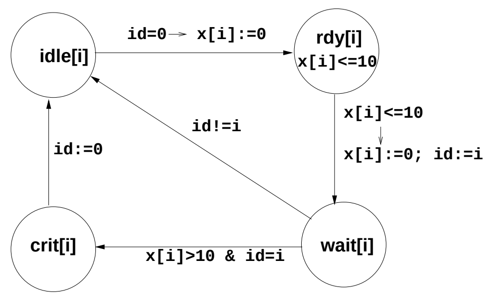


Figure 3: **Fischer's mutual exclusion protocol.** The timed automaton for the ith process is shown.

Table 1 shows the results of the comparison against DDD and RED for checking mutual exclusion for increasing numbers of processes. We refer to our model checker as TMV. For DDD and TMV, the table lists both the run-times and the peak number of nodes in the decision diagram for the reachable state set. We find that DDD outperforms TMV due to the blow-up of BDDs. In spite of the optimizations of Section 5, the peak node count in the case of DDD is less than that for TMV for the larger benchmarks. In particular, in addition to eliminating infeasible paths as TMV does, the local reduction operations performed by DDD during node creation can eliminate unnecessary DDD nodes without adding any time overhead. For example, DDD can reduce a function of the form $e_1 \wedge e_2 \wedge e_3$ under the transitivity constraint $[e_1 \wedge e_2] \implies e_3$ to simply the conjunction $e_1 \wedge e_2$. The BDD `Restrict` operator cannot always achieve this as it is sensitive to the BDD variable ordering. Furthermore, TMV contains many other BDDs, such as those for the transitivity constraints, to which we do not apply the `Restrict` optimization due to its runtime overhead. Finally, in comparison to RED, we see that while TMV is faster on the smaller benchmarks, RED's superior memory performance enables it to complete for 7 processes while TMV runs out of memory.

Table 2 shows the comparison with KRONOS and RED for checking non-zenoness. The time for KRONOS is the sum of the times for product construction and backward model checking. We notice that while KRONOS does better for smaller numbers of processes, the product automaton it constructs grows very quickly, becoming too large to construct at 6 processes. The run times for TMV, on the other hand, grow much more gradually, demonstrating the advantages of a

Number of Processes	RED Time (sec.)	DDD Time (sec.)	DDD Reach Set (peak nodes)	TMV Time (sec.)	TMV Reach Set (peak nodes)
3	0.21	0.06	130	0.11	101
4	1.13	0.14	352	0.38	316
5	4.53	0.33	854	1.85	1127
6	15.11	0.90	2375	17.41	4685
7	46.31	2.65	6346	*	*

Table 1: **Checking mutual exclusion for Fischer's protocol.** A "*" indicates that the model checker ran out of memory.

fully symbolic approach. For this property, the BDDs remain small even for larger numbers of processes. Thus, TMV outperforms RED, especially as the number of processes increases. These results indicate that when the representation (BDDs) remains small, Boolean methods for quantifier elimination and deciding SL can outperform non-Boolean methods by a significant factor.

Number of Processes	KRONOS Time (sec.)	RED Time (sec.)	TMV Time (sec.)	TMV Reach Set (peak nodes)
3	0.03	0.28	0.24	28
4	0.23	1.30	0.44	39
5	1.98	5.05	0.80	54
6	*	17.80	2.15	69
7	*	57.95	6.61	88

Table 2: **Checking non-zenoness for Fischer's protocol.** A "*" indicates that KRONOS exited with an "out of memory" error.

Although they are preliminary, our results indicate that our model checker based on a general purpose BDD package can outperform methods based on specialized representations of SL formulas. The drawback of our BDD-based implementation is its poor memory performance on some examples. However, there is still scope for improving our implementation, especially in finding more efficient ways of eliminating unnecessary BDD nodes as is possible with DDDs. Furthermore, note that the memory problems we face arise from our use of BDDs, while the techniques we have presented in this paper can make use of any representation of Boolean functions. In particular, we are starting to work on a SAT-based implementation of our method; such an implementation might better handle the growth in the number of Boolean variables. Finally, we are also exploring heuristics for automatically generating good BDD variable orderings, such as those based on compositional methods [7].

Acknowledgments

We thank Joël Ouaknine and Ofer Strichman for their comments. We also thank the authors of DDD and RED for providing their tools and answering our queries.

References

[1] L. Aceto, P. Bouyer, A. Burgueno, and K. G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, To appear.

[2] Rajeev Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, July 1999.

[3] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[4] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Korniowicz, and Roberto Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In Andrei Voronkov, editor, *18th International Conference on Automated Deduction (CADE)*, volume 2392 of *Lecture Notes in Computer Science*, pages 195–210. Springer-Verlag, July 2002.

[5] Gilles Audemard, Alessandro Cimatti, Artur Korniowicz, and Roberto Sebastiani. Bounded model checking for timed systems. In Doron Peled and Moshe Y. Vardi, editors, *Formal Techniques for Networked and Distributed Systems - FORTE'02*, volume 2529 of *Lecture Notes in Computer Science*, pages 243–259. Springer, November 2002.

[6] Gerd Behrmann, Kim Guldstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer-Verlag, July 1999.

[7] Dirk Beyer. Improvements in BDD-based reachability analysis of timed automata. In Jose Nuno Oliveira and Pamela Zave, editors, *Proceedings of the 10th International Symposium of Formal Methods Europe (FME)*, volume 2021 of *Lecture Notes in Computer Science*, pages 318–343. Springer-Verlag, 2001.

[8] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In O. Grumberg, editor, *Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 179–190. Springer-Verlag, 1997.

[9] Olivier Coudert and Jean Christophe Madre. A unified framework for the formal verification of sequential circuits. In *Proc. Intl. Conference on Computer-Aided Design (ICCAD'90)*, pages 126–129, 1990.

[10] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory A*, 14:288–297, 1973.

[11] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[12] Manolis Koubarakis. Complexity results for first-order theories of temporal constraints. In *Proc. 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 379–390, 1994.

[13] Ken L. McMillan. Applying SAT methods in unbounded symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Proc. 14th Intl. Conference on Computer-Aided Verification (CAV'02)*, LNCS 2404, pages 250–264. Springer-Verlag, July 2002.

[14] J. B. Møller. Simplifying fixpoint computations in verification of real-time systems. In *Proc. Second Workshop on Real-Time Tools*, Copenhagen, Denmark, 1 August 2002.

[15] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, N. Jain, and O. Maler. Verification of timed automata via satisfiability checking. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems FTRTFT'02*, volume 2469 of *Lecture Notes in Computer Science*, pages 225–244, 2002.

[16] O. Strichman. Optimizations in decision procedures for propositional linear inequalities. Technical Report CMU-CS-02-133, Carnegie Mellon University, 2002.

[17] O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding separation formulas with SAT. In E. Brinksma and K. G. Larsen, editors, *Proc. 14th Intl. Conference on Computer-Aided Verification (CAV'02)*, LNCS 2404, pages 209–222. Springer-Verlag, July 2002.

[18] Farn Wang. Efficient verification of timed automata with BDD-like data-structures. In *Proc. 4th Intl. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 2575 of *Lecture Notes in Computer Science*, pages 189–205, New York, January 2003.

[19] Sergio Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.