Symbolic Simulation-Checking of Dense-Time Systems *

Farn Wang

Dept. of Electrical Engineering, National Taiwan University
1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC;
+886-2-33663602; FAX:+886-2-23671909;
farn@cc.ee.ntu.edu.tw; http://cc.ee.ntu.edu.tw/~farn

Abstract. Intuitively, an (implementation) automata is simulated by a (specification) automata if every externally observable transition by the implementation automata can also be made by the specification automata. In this work, we present a symbolic algorithm for the simulation-checking of timed automatas. We first present a simulation-checking procedure that operates on state spaces, representable with convex polyhedra, of timed automatas. We then present techniques to represent those intermediate result convex polyhedra with zones and make the procedure an algorithm. We then discuss how to handle Zeno states in the implementation automata. Finally, we have endeavored to realize the algorithm and report the performance of our algorithm in the experiment.

Keywords: simulation, implementation, refinement, dense-time, real-time, embedded, model-checking, verification, events

1 Introduction

In the last two decades, we have witnessed significant progress in both theory and applications of the formal verification of real-time systems. Especially, the technology of dense-time system model-checking [1] has been well-received by the academia, realized with many tools [8,19,22], and used for the verification of several industrial projects [6]. With model-checking, we represent the implementation as an automata (state-transition diagram or table) and the specification as a temporal logic formula and want to check whether the implementation satisfies the specification. However, we have to admit that the promise of the model-checking technology of real-time systems has not been fulfilled yet. One reason is that engineers are not trained in writing formulas in temporal logics, like TCTL (Timed Computational-Tree Logic) [1]. In many applications, engineers may also envision their specifications as automatas. With model-checking, it is usually difficult to completely and correctly represent a specification automata as a set of logic formulas. There are two common frameworks for checking implementation

* The work is partially supported by NSC, Taiwan, ROC under grants NSC 94-2213-E-002-085 and NSC 94-2213-E-002-092.
automatas against specification ones. The first is the *language inclusion problem* which checks if all runs of an implementation are also those of the specification. It was proved in [3] that when both the implementation and specification are represented as *timed automatas (TA)* [3], the language inclusion problem is undecidable. The second is called the *simulation-checking problem* that intuitively checks if every transition that can be performed by the implementation can also be matched by the specification at the same instant. (Formal definition in page 4.) It has been proved that the simulation checking problem of TAs is in EXPTIME [18]. However, the algorithms in the literature are either based on region graph analysis [1] of the product of the implementation and the specification TAs [18] or based on time-abstraction [13], which does not preserve the timing properties. In this work, we have the following contributions.

- **A symbolic procedure that checks for the simulation relation between dense-time systems.** The procedure handles convex polyhedra in dense spaces of variables and straightforwardly falls in the realm of state-representation manipulation of *linear hybrid automatas (LHA)* [2]. Thus the procedure is good for the simulation-checking of both LHAs and TAs.

- **Techniques to implement the above-mentioned procedure with zone-technology.** In general, the manipulation of convex polyhedra can be very complex and the verification problem of LHAs is undecidable. As a special subclass of LHAs, the state-spaces of TAs can be efficiently represented and manipulated with zones\(^1\) [14]. In section 6, we present techniques to represent the intermediate results of the above-mentioned procedure with zones. The techniques effectively make the procedure a symbolic algorithm.

- **A technique to handle Zeno states in simulation checking.** A Zeno state is one from which no computation yields divergent computation time. Intuitively, if the implementation TA can transit to a Zeno state, such a transition need not be matched by the specification TA and should not affect the answer to the simulation-checking problem. In section 7, we present a lemma that helps us extending our simulation-checking algorithm to handle Zeno states in the implementation TAs.

We have realized our algorithm and techniques with TCTL model-checker RED, version 7. Following is our presentation plan. Section 2 discusses related work. Section 3 defines our modeling language, TAs extended with event notations. Section 4 gives the definition and symbolic representation of the simulation relation between two TAs. Section 5 derives a symbolic procedure out of the definition of simulation relation. Section 6 discusses how to implement the procedure in section 5 with zones. Section 7 discusses how to filter out false negation from Zeno states of the implementation TA. Section 8 reports our program and the experiment. Section 9 is the conclusion.

\[^1\] A *zone* is a conjunction of atomic propositions and constraints like either \(x - y \leq c\) or \(x - y < c\) where \(x, y\) are either zero or clocks and \(c\) is an integer constant.
2 Related work

Cerans showed that the bisimulation-checking problem of timed processes is decidable [9]. Taşkın et al showed that the simulation-checking problem of TAs is in EXPTIME [18]. They also proposed an algorithm to check whether a location homomorphism between an implementation TA and a specification TA preserves timed behaviors. However, there is no general strategy to efficiently construct such homomorphisms. In comparison, our approach is purely algorithmic and automatic.

Henzinger et al presented an algorithm that computes the time-abstract simulation that does not preserve timed properties [13].

Nakata also discussed how to do symbolic bisimulation checking with integer-time labelled transition systems [16]. Beyer has implemented a refinement-checking algorithm for TAs with integer-time semantics [7]. In comparison, our algorithm is for dense-time semantics.

Lin and Wang presented a complete and sound proof system for the equivalence of TAs with dense-time semantics [15]. Usually, the proofs may need human guidance.

Aceto et al constructed a modal logic formula that completely characterizes a TA [4]. Thus the simulation checking problem can be reduced to the model-checking problem. However, the formula they constructed is not for TAs with timed invariance constraints and does not handle the effect of Zeno states. In a sense, our formula (A) in definition 4 is also such a formula for TA with timed invariance constraints. Specifically, their characteristic formula is also of greatest fixpoint in nature and falls in the realm of LHA verification. It is not clear whether they can efficiently evaluate such characteristic formulas. In contrast, we have proposed and implemented the simulation-checking algorithm with zone technology.

3 A modeling language of dense-time systems

We need the following notations for convenience of presentation. Given a set $P$ of atomic propositions and a set $X$ of clocks, we use $B(P,X)$ as the set of all Boolean combinations of atoms of the forms $p$ and $x \sim c$, where $p \in P$, $x \in X \cup \{0\}$, ‘$\sim$’ is one of $\leq, <, =, >, \geq$, and $c$ is an integer constant. An element in $B(P,X)$ is called a state predicate.

A valuation of a set $Y$ (domain) is a mapping from $Y$ to a codomain. A partial valuation is undefined for some elements in the domain. When it is not said specifically, a valuation means a total valuation that assigns a value to every element in the domain. A valuation $\nu$ satisfies a state-predicate $\eta$, in symbols $\nu \models \eta$, if the state-predicate evaluates to true when all its variables are interpreted according to $\nu$. Given two (partial) valuations $\Pi$ and $\Pi'$ on domain $Y$, $\Pi \Pi'$ is a new valuation defined in the following way. For all $y \in Y$,

- if $\Pi'(y)$ is defined, $\Pi \Pi'(y) = \Pi'(y)$;
from the proposition variables of each rule execution. If $y$ stays unchanged in the transition.

**Example 1.** We have the transition diagrams of two example TEAs in figure 1. They share events send and receive and global proposition $q$. They respectively have local clocks $x_1$ and $x_2$. $A_1$ has two transitions while $A_2$ has four. We stack the events, triggering conditions, and the assignments made at each transition. The initial conditions are labeled by the arcs without a source.

We use the following extension of TA [3] for the modeling language in this work.

**Definition 1. timed event-automata (TEA)** A timed event-automata (TEA) $A$ is a tuple $\langle \Sigma, X, G, L, I, H, E, \epsilon, \tau, \pi \rangle$ with the following restrictions. $\Sigma$ is a finite set of event names. $X$ is a finite set of clocks. $G$ is a finite set of global atomic propositions. $G$ and $\Sigma$ represent the set of external observables of a TEA. $L$ is a finite set of local atomic propositions such that $G \cap L = \emptyset$. $I \subseteq B(G \cup L, X)$ is the initial condition. $H \subseteq B(G \cup L, X)$ is the invariance condition. $E$ is the finite set of transitions. $\epsilon : E \mapsto 2^\Sigma$ labels each rule with a set of input/output events. $\tau : E \mapsto B(G \cup L, X)$ defines the triggering condition of each rule execution. For each $e \in E$, $\pi(e)$ is a partial valuation from $X$ to $\{0\}$ and from $G \cup L$ to $\{\text{true}, \text{false}\}$ that defines the assignments to clocks and proposition variables of each rule execution. If $\pi(e)(y)$ is undefined, it means variable $y$ stays unchanged in the transition.

For convenience, we assume that for every TEA, there is a null transition $\bot$ such that $\epsilon(\bot) = \emptyset$, $\tau(\bot) = \text{true}$, and $\pi(\bot)$ is undefined on everything.

**Example 1.** We have the transition diagrams of two example TEAs in figure 1. They share events send and receive and global proposition $q$. They respectively have local clocks $x_1$ and $x_2$. $A_1$ has two transitions while $A_2$ has four. We stack the events, triggering conditions, and the assignments made at each transition. The initial conditions are labeled by the arcs without a source.

Let $\mathbb{N}$ be the set of non-negative integers and $\mathbb{R}^{\geq 0}$ the set of non-negative reals.

**Definition 2. States of a TEA** A state of TEA $A$ is a total valuation from $X$ to $\mathbb{R}^{\geq 0}$ and $G \cup L$ to $\{\text{true}, \text{false}\}$. Let $V_A$ denote the set of states of $A$. For any state $\nu$ and $\delta \in \mathbb{R}^{\geq 0}$, $\nu + \delta$ is a valuation identical to $\nu$ except that for every $x \in X$, $\nu(x) + \delta = (\nu + \delta)(x)$.

Given two states $\nu, \nu'$ and a transition $e$, we say $A$ transits with $e$ from $\nu$ to $\nu'$, in symbols $\nu \xrightarrow{e} \nu'$, if $\nu \models \tau(e), \nu \pi(e) = \nu'$, and $\nu' \models H$. Based on the above-presented notions, we are ready to define linear computations of TEAs.

**Definition 3. runs** Given a TEA $A = \langle \Sigma, X, G, L, I, H, E, \epsilon, \tau, \pi \rangle$, a run is an infinite computation of $A$ along which time diverges. Formally speaking, a run is an infinite sequence of state-time pairs $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots$ such that
• \( t_0 \leq t_1 \ldots \leq t_k \ldots \) is a monotonically increasing divergent real-number sequence, i.e.,
  \( \forall c \in \mathbb{N}, \exists k > 1, t_k > c \); and
• for all \( k \geq 0 \), for all \( \delta \in [0, t_{k+1} - t_k] \), \( \nu_k + \delta \models H \); and
• for all \( k \geq 0 \), there is an \( e \in E \) such that \( \nu_k + t_{k+1} - t_k \xrightarrow{e} \nu_{k+1} \).

\[ \blacksquare \]

4 Implementation, simulation, refinement, and equivalence

Suppose we are given two TEAs \( A_1 \) and \( A_2 \) with \( A_i = \langle \Sigma, X_i, G, L_i, \Pi_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle \), \( 1 \leq i \leq 2 \). Intuitively, \( A_1 \) implements (or refines, or is simulated by) \( A_2 \) if we can map every reachable state \( \nu_1 \) of \( A_1 \) to a reachable state \( \nu_2 \) of \( A_2 \) such that every externally observable that \( A_1 \) can do at a specific time from \( \nu_1 \), \( A_2 \) can also do it at the same instant from \( \nu_2 \). If \( A_2 \) can always direct its runs so that no difference in external behaviors of \( A_1 \) and \( A_1 \) will ever be observed, then we say \( A_1 \) implements \( A_2 \). This is formalized with the following definition.

We need the following notations for the convenience of discussion. Two transitions \( e_1 \in E_1 \) and \( e_2 \in E_2 \) are compatible if \( e_1(e_1) = e_2(e_2) \) and \( \forall p \in G(\pi_1(p) = \pi_2(p)) \). Given an \( e_1 \in E_1 \), we let \( E_{2(e_1)}^2 = \{ e_2 \mid e_2 \in E_2; e_2 \text{ is compatible with } e_1 \} \). Given \( \alpha \in V_{A_1}, \beta_1, \beta_2 \in V_{A_2}, \delta \in \mathbb{R}^{\geq 0} \), and \( Q \subseteq V_{A_1} \times V_{A_2} \), \( \text{stutter}_2(\alpha, \beta_1, \delta, \beta_2, Q) \) is true iff

• \( A_2 \) can go from \( \beta_1 \) to \( \beta_2 \) through a series of time-progression steps and discrete transition steps in \( E_{2(e_1)}^2 \);
• the finite run from \( \beta_1 \) to \( \beta_2 \) is \( \delta \) time units long; and
• for any \( 0 \leq \delta' \leq \delta \) and state \( \beta' \in V_{A_2} \) that is \( \delta' \) time units away from \( \beta_1 \) in the run, \( (\alpha + \delta', \beta') \in Q \).

Formally speaking, \( \text{stutter}_2(\alpha, \beta_1, \delta, \beta_2, Q) \) is true iff there is a finite run \((\bar{\nu}_0, t_0)(\bar{\nu}_1, t_1)\ldots(\bar{\nu}_k, t_k)\) of \( A_2 \) and \( t_{k+1} \geq t_k \) such that

• \( t_{k+1} - t_0 = \delta \wedge \beta_1 = \bar{\nu}_0 \wedge \beta_2 = \bar{\nu}_k + t_{k+1} - t_k \); and
• \( \forall 0 \leq h \leq k \exists \bar{\nu}_{h+1} \in E_{2(e_1)}^2, \bar{\nu}_h + t_{h+1} - t_h \xrightarrow{\bar{\nu}_{h+1}} \bar{\nu}_{h+1} \).

Definition 4. Implementation, simulation, and equivalence

Suppose we are given two TEAs \( A_1 \) and \( A_2 \) such that \( A_i = \langle \Sigma, X_i, G, L_i, \Pi_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle \), \( 1 \leq i \leq 2 \), \( L_1 \cap L_2 = \emptyset \), and \( X_1 \cap X_2 = \emptyset \). A simulation relation \( Q \) from \( A_1 \) to \( A_2 \) is a binary relation from \( V_{A_1} \) to \( V_{A_2} \) with the following restriction. For every \( (\nu_1, \nu_2) \in Q \), \( \nu_1 \) and \( \nu_2 \) agree on interpretation of variables in \( G \) and for every \( \delta \in \mathbb{R}^{\geq 0} \) and transition \( e_1 \) of \( E_1 \) such that for all \( \bar{\delta} \in [0, \delta] \), \( \nu_1 + \bar{\delta} \models H_1 \) and \( \nu_1 + \delta \xrightarrow{e_1} \nu'_1 \), there are \( \nu'_2 \in V_{A_2} \) and \( e_2 \in E_{2(e_1)}^2 \) such that \( \text{stutter}_2(\nu_1, \nu_2, \delta, \nu'_2, Q), \nu'_2 \xrightarrow{e_2} \nu'_2 \pi_2(e_2), \) and \( (\nu'_2, \nu'_2 \pi_2(e_2)) \in Q \). Or equivalently, \( (\nu_1, \nu_2) \notin Q \) if there is \( e_1 \in E_1 \) satisfying formula (A) in the following.

\[
\exists \delta \in \mathbb{R}^{\geq 0} \left( \begin{array}{c}
(\nu_1 + \bar{\delta} \models \tau_i(e_1) \land (\nu_1 + \delta) \pi_1(e_1) \models H_1) \\
(\neg \exists \delta \leq \delta' \leq \delta (v_1 + \delta' \models \neg H_1)) \\
\text{stutter}(\nu_1, \nu_2, \delta, \nu'_2, Q) \\
\land \exists \nu'_2 \in V_{A_2} \\
(\exists \nu_2 \in E_{2(e_2)}^2) \\
(\nu'_2 \xrightarrow{e_2} \nu'_2 \pi_2(e_2) \\
(\nu_1 + \delta \models \tau_i(e_1), \nu'_2 \pi_2(e_2) \in Q)) \end{array} \right) \right) \ldots \text{(A)}
\]
Note that formula (A) has the following structure: $\exists \delta \in \mathbb{R}^2(\delta \land \neg C)$. Formula (B) says that $A_1$ can do transition $e_1$ at time $\delta$ from $\nu_1$. Formula (C) says that $A_2$ can match $e_1$ with $e_2$ at the same time from $\nu_2$ through a run with transitions internal to $A_2$. The quantification scope of $\delta$ contains both (B) and (C) to make sure $e_1$ and $e_2$ are matched at the same instant.

Given simulation relation $Q$ from $A_1$ to $A_2$, we denote $A_1 \preceq_Q A_2$ if for every state $\nu_1 \models I_1$, there is a $\nu_2 \models I_2$ with $(\nu_1, \nu_2) \in Q$. If $\exists Q(A_1 \preceq_Q A_2)$, we say $A_1$ implements (or refines, or is simulated by) $A_2$, in symbols $A_1 \preceq A_2$. Two TEAs $A_1$ and $A_2$ are equivalent (i.e. bisimulate), in symbols $A_1 \equiv A_2$, if $A_1 \preceq A_2$ and $A_2 \preceq A_1$.

**Example 2.** In example 1, $A_1$ is not simulated by $A_2$ since $A_1$ can make a transition after 5 time units in state $q = 1$ while $A_2$ cannot. $A_2$ cannot. Such sequences are from Zeno states with $x_2$’s value converging to 5. In fact, $A_2$ is NZ-simulated by $A_1$.

**Example 3.** In example 1, $A_2$ is not simulated by $A_1$ since $A_2$ can yield infinite sequences of the send events while $A_1$ cannot. Such sequences are from Zeno states with $x_2$’s value converging to 5. In fact, $A_2$ is NZ-simulated by $A_1$.

### 5 Symbolic procedure for simulation-checking

Formula (A) leads to a greatest fixpoint procedure for calculating a simulation relation from $A_1$ to $A_2$ if any. The idea is to first compute an initial image of $Q$ and then
iteratively delete state pairs from \( Q \) with formula (A) until a fixpoint is reached. Please be reminded that formula (A) has the following structure: \( \exists \delta \in \mathbb{R}^{\geq 0} (B \land \neg C) \). In the following, we first present a scheme for the symbolic representation of \( Q \). Then we discuss how to construct formulas (B) and (C) respectively out of \( Q \).

### 5.1 Symbolic representation of \( Q \)

We define *linear hybrid predicates* (LH-predicates) to represent convex polyhedra in dense space. An LH-predicate is a Boolean combination whose atoms are either

- atomic proposition, like \( p \), or
- linear constraints like \( \sum_{1 \leq i \leq n} c_i x_i \sim d \) where \( x_1, \ldots, x_n \) are clock variables, \( c_1, \ldots, c_n, d \) are integer constants, and \( \sim \in \{\leq, <\} \).

Let \( C_{(A_1, A_2)} \) be the biggest constants used in \( A_1 \) and \( A_2 \). An LH-predicate is called a *zone-predicate* if its linear constraints are like \( x_1 - x_2 \sim d \) where \( x_1, x_2 \) are either zeros or clock variables and \( d \in \mathbb{N} \cap [0, C_{(A_1, A_2)}] \). Clearly, zone-predicates are special cases of LH-predicates. In practice, such zone-predicates can be implemented with DBMs [10] or CRDs [19] while LH-predicates can be with convex polyhedra [12] or HRDs [20]. Given a \( (\nu_1, \nu_2) \in Q \), we require in definition 4 that \( \nu_1 \) and \( \nu_2 \) agree on the interpretation of variables in \( G \). Thus if we have LH-predicates \( \eta_1 \) and \( \eta_2 \) for \( \nu_1 \) and \( \nu_2 \) respectively, we can use \( \eta_1 \land \eta_2 \) to represent pairs like \( (\nu_1, \nu_2) \) in \( Q \).

**Example 4.** Given \( G = \{a\} \), \( X_1 = \{x_1\} \), \( L_1 = \{b_1\} \), \( X_2 = \{x_2\} \), \( L_2 = \{b_2\} \), we may have the following LH-predicate (also zone-predicate) \( (f_Q) \) for \( Q \).

\[
(a \land b_1 \land \neg b_2 \land 0 \leq x_1 \land 3 < x_2 \leq 5 \land x_2 - x_1 \leq 5) \lor (\neg a \land 2 \leq x_1 < 9 \land 1 < x_2 \land x_1 - x_2 < 8) \quad (f_Q)
\]

In the following, we propose procedures that manipulate LH-predicates to represent pairs like \( (\nu_1, \nu_2) \).

### 5.2 Basic building blocks

One fundamental procedure is *Fourier-Motzkin elimination* [11]. Suppose we have a formula \( F \) of variables in set \( Y \). In this work, Fourier-Motzkin elimination constructs \( \exists y(F) \) as a formula without \( y \in Y \). For LH-predicates and zone-predicates, efficient implementation of Fourier-Motzkin elimination has been discussed in [14, 19, 20].

**Example 5.** For formula \( (f_Q) \) in example 4,

\[
\exists a(f_Q) \equiv (b_1 \land \neg b_2 \land 0 \leq x_1 \land 3 < x_2 \leq 5 \land x_2 - x_1 \leq 5) \lor (2 \leq x_1 < 9 \land 1 < x_2 \land x_1 - x_2 < 8)
\]

And \( \exists x_1(f_Q) \equiv (a \land b_1 \land \neg b_2 \land 3 < x_2 \leq 5) \lor (\neg a \land 1 < x_2 < 6) \).

We also need the following procedures to present our procedure. Given a symbolic representation \( \eta \) of \( \nu \), the symbolic representation of \( \nu + \delta \) can be obtained by replacing each clock \( x \) in \( \eta \) with \( x + \delta \) [14].

7
Example 6. For the \((f_Q)\) in example 4, \(f_Q + \delta\) is the following.
\[
(a \land b_1 \land \neg b_2 \land 0 \leq x_1 + \delta \land 3 < x_2 + \delta \leq 5 \land x_2 - x_1 \leq 5) \\
\lor (\neg a \land 2 \leq x_1 + \delta < 9 \land 1 < x_2 + \delta \land x_1 - x_2 < 8)
\]
Note that for constraints like \(x_1 - x_2 < 8\), the positive and negative \(\delta\) respectively from \(x_1\) and \(x_2\) cancel with each other. 

Let \(\text{path}(\eta_1, \delta)\) be the constraint that from now to \(\delta\) time units in the future, \(\eta_1\) is true, i.e., \(\text{path}(\eta_1, \delta) \equiv \neg 3\delta'((-\eta_1) + \delta\land 0 \leq \delta' \land \delta' \leq \delta).\) We can then define \(\text{Tbck}(\eta', \eta)\) that computes the space representation of states
- from which we can go to states in \(\eta\) simply by time-passage; and
- every state in the time-passage also satisfies condition \(\eta'\).
\(\text{Tbck}(\eta', \eta)\) can be constructed as \(\exists t (0 \leq \eta + t \land \text{path}(\eta', t))\) [14].

Given a partial assignment \(\Pi\) of \(G \cup L_1 \cup L_2 \cup X_1 \cup X_2\), we let \(\eta \Pi\) be the precondition to \(\eta\) before the assignment. Suppose \(\Pi\) is defined for \(\{y_1, \ldots, y_n\}\).
\[
\eta \Pi \equiv \langle \forall x \text{ is a clock defined in } \Pi. \, x \geq 0 \rangle \land \exists y_1 \ldots \exists y_n (\eta \land \bigwedge_{1 \leq i \leq n} y_i = \Pi(y_i))
\]

Given \(e_1 \in E_1\) and \(e_2 \in E_{2(e_1)}\) such that \(e_1\) and \(e_2\) are compatible, the weakest precondition to \(\eta\) through discrete transition pair \((e_1, e_2)\) can be represented as \(\text{Xbck}(e_1, e_2)(\eta) \equiv \tau_1(e_1) \land \tau_2(e_2) \land (\eta(\pi_1(e_1)\pi_2(e_2)))\).

With procedures \(\text{Xbck}(e_1, e_2)()\) and \(\text{Tbck}()\), we can construct the symbolic backward reachability procedure, denoted \(\text{Rbck}_{E_2}^{(e_1)}(\eta_1, \eta_2)\) for convenience, as in [14, 19]. Intuitively, \(\text{Rbck}_{E_2}^{(e_1)}(\eta_1, \eta_2)\) characterizes the state-space for \(\exists \eta_2 \forall \eta_1\) through transitions in \(E_2^{(e_1)}\). Computationally, \(\text{Rbck}_{E_2}^{(e_1)}(\eta_1, \eta_2)\) is the least fixpoint solution of equation: \(Y = \eta_2 \lor \text{Tbck}\left(\eta_1, \bigvee_{e_2 \in E_2^{(e_1)}} \text{Xbck}_{(e_1, e_2)}(Y)\right)\). That is, \(\text{Rbck}_{E_2}^{(e_1)}(\eta_1, \eta_2) \equiv \exists \eta_2 \forall \text{Tbck}\left(\eta_1, \bigvee_{e_2 \in E_2^{(e_1)}} \text{Xbck}_{(e_1, e_2)}(Y)\right)\). Now we need to construct a symbolic characterization of \((\nu_1, \nu_2)\) for \(\text{stutter}(\nu_1, \nu_2, \delta, \nu'_2, Q)\) when \(\nu'_2\) and \(Q\) are represented as formulas \(f_{\nu'_2}\) and \(Q\) respectively. We use an auxiliary clock \(z \notin X_1 \cup X_2\) to measure the length of the stuttering run. Then, we have
\[
f_{\text{stutter}}(\delta, f_{\nu'_2}, Q) \equiv \exists z (z = 0 \land \text{Rbck}_{E_2}^{(e_1)}(Q, z = \delta \land f_{\nu'_2}))
\]

5.3 Construction of formula (B)

We rewrite formula (B) in formula (A) as follows.
\[
\nu_1 + \delta \models \tau_1(e_1) \land (\nu_1 + \delta)\pi_1(e_1) \models H_1 \land \neg \exists 0 \leq \delta' \leq \delta (\nu_1 + \delta' \models \neg H_1) \quad \ldots \ldots (B)
\]
The first conjunct says that after \(\delta\) time units from state \(\nu_1\), \(A_1\) satisfies the triggering condition of transition \(e_1\). The second conjunct says that in this time-progression of \(\delta\) time units, \(A_1\) always satisfies \(H_1\). The third conjunct says that at the end of the time-progression, \(A_1\) goes from \(\nu_1 + \delta\) to \((\nu_1 + \delta)\pi_1(e_1)\) and still satisfies \(H_1\). The weakest characterization, i.e. formula (B), of \(\nu_1\) can be derived backward from \(H_1\) as follows.
\[(\text{Xbck}_{e_1, \downarrow})(H_1) + \delta) \land \text{path}(-H_1, \delta) \]  

It characterizes those states that can take transition \(e_1\) at \(\delta\) time units away. The first conjunct corresponds to the first two conjuncts in formula (B) while the last one to the last one in (B).

### 5.4 Construction of formula (C)

We rewrite formula (C) as follows.

\[
\exists \nu'_2 \in V_{A_2} \left( \text{stutter}_2(\nu_1, \nu_2, \delta, \nu'_2, Q) \land \exists e_2 \in E_2^{(e_1)} \left( \nu'_2 \not\in e_2, \nu'_2 \not\in e_2 \right) \land (\nu_1 + \delta) \pi_1(e_1), \nu_2 \pi_2(e_2) \in Q \right) \]  

The first conjunct in the outer quantification is for the measurement of the length, \(\delta\), of the stuttering run through transitions internal to \(A_2\). The second is for the execution of \(e_2 \in E_2^{(e_1)}\). Specifically, the quantified \(\nu'_2\) is for the precondition of \(e_2\). Given a simulation relation representation \(Q\), the constraint of \(\nu'_2\) from the second conjunct is as follows.

\[
\bigvee_{e_2 \in E_2^{(e_1)}} \text{Xbck}_{e_1, e_2}(Q) \]  

To make sure that \(A_1\) and \(A_2\) observe the same behavior with \(e_1\) and \(e_2\) respectively, we construct the precondition of both \(e_1\) and \(e_2\) out of \(Q\) instead of the representation of \(\nu_2\). Now with the formulations of \(f_{\nu'_2}\) in the above and \(f_{\text{stutter}}(\delta, f_{\nu'_2}, Q)\) in page 8, we find the following formulation for formula (C).

\[
\exists z (z = 0 \land \text{Xbck}_{e_1, e_2}(Q), z = \delta \land \bigvee_{e_2 \in E_2^{(e_1)}} \text{Xbck}_{e_1, e_2}(Q)) \]  

### 5.5 Procedure

With formulas \((f_B)\) and \((f_C)\), we find that formula (A) can be constructed as

\[
f_A(e_1, Q) \equiv \exists \delta \geq 0 \left( (\text{Xbck}_{e_1, \downarrow})(H_1) + \delta) \land \text{path}(-H_1, \delta) \land \exists z (z = 0 \land \text{Xbck}_{e_1, e_2}(Q), z = \delta \land \bigvee_{e_2 \in E_2^{(e_1)}} \text{Xbck}_{e_1, e_2}(Q)) \right) \]  

With formula \(f_A(e_1, Q)\), we are now ready to present our procedure for simulation checking. The procedure is a greatest-fixpoint one. We start from \(Q = H_1 \land H_2\). Then we iteratively delete state pairs described in \(f_A(e_1, Q)\) for each \(e_1 \in E_1\) until a fixpoint is reached. For convenience, we let \(FM_{\text{elm}}(F, \{y_1, \ldots, y_n\}) = \exists y_1 \exists y_2 \ldots \exists y_n(F)\).

```plaintext
Simulation_Check(A_1, A_2) /* A_i = (\Sigma, X_i, G, L_i, I_i, H_i, E_i, e_i, \tau_i, \pi_i), 1 \leq i \leq 2 */ { 
let Q := H_1 \land H_2; Q' := false; ............................................(F) 
while (Q \neq Q'), do {
    Q' := Q; 
    for each e_1 \in E_1, Q := Q \land \neg f_A(e_1, Q); 
    if (I_1 \neq FM_{\text{elm}}(I_1 \land I_2 \land Q, L_2 \cup X_2)) ....................................(J)
} 
```
print “A₁ does not implement A₂.” and return false.
}
print “A₁ implements A₂.” and return true.
}

We can start the greatest fixpoint image from \( H₁ \land H₂ \) at statement (F) because of the following lemma.

**Lemma 1.** If \( A₁ \preceq A₂ \), then there is a \( Q \) such that \( A₁ \preceq Q \) \( A₂ \) and \( \forall (ν₁, ν₂) \in Q(ν₁ν₂ \models H₁ \land H₂) \).

In statement (J), we check whether all initial states of \( A₁ \) is paired with some initial states of \( A₂ \) in \( Q \). The statement employs a technique called *early decision of greatest fixpoint (EDGF)* [21] and could significantly reduce the computation time of greatest fixpoint evaluation when no simulation relation exists. The following lemma establishes the correctness of our procedure.

**Lemma 2.** When Simulation\_Check\((A₁, A₂)\) halts, it returns true iff \( A₁ \preceq A₂ \).

### 6 Algorithm with zone-technology

When the initial condition, invariance condition, and transition triggering conditions of \( A₁ \) and \( A₂ \) are all presented as LH-predicates, we can prove that all operations in Simulation\_Check\((A₁, A₂)\) yield only LH-predicates. This is based on the following lemma and all our operations in Simulation\_Check\((A₁, A₂)\) are based on the basic operations listed in the following lemma.

**Lemma 3.** Given LH-predicates \( η₁ \) and \( η₂ \), \( \neg η₁ \), \( η₁ \lor η₂ \), \( η₁ + δ \), \( \exists x(η₁) \) with \( x ∈ X₁ \cup X₂ \), and \( \exists p(η₁) \) with \( p ∈ G \cup L₁ \cup L₂ \) are all LH-predicates.

However, the representation and manipulation of LH-predicates are usually less than efficient. In this section, we present techniques that allow us to implement procedure Simulaton\_Check() with the zone-technology. The correctness of such techniques is based on theorem 10 in [18] which asserts that for any \( ν₁, ν₁' \) in the same *region*\(^2\) of \( A₁ \) and \( ν₂, ν₂' \) in the same region of \( A₂ \), \( (ν₁, ν₂) \in Q \) iff \( (ν₁', ν₂') \in Q \). In the following, we carefully examine formula \( (fₐ) \) for operations that yield non-zone-predicates and discuss how to rewrite the the predicates to make it representable with zones. We find two classes of operations that create non-zone-predicates. We present techniques in subsections 6.1 and 6.2 to represent the results of such operations with zones. In subsection 6.3, we combine the techniques of the two subsections and reformulate procedure Simulation\_Check() as an algorithm.

Formulas \( (f₋) \) and \( (fₐc) \) both use basic procedures \( \mathbb{Rbck}_{E₁'2}() \), \( \mathbb{Tbck}() \), and \( \mathbb{Xbck}_{e₁,e₂}() \), which are in turn built upon Fourier-Motzkin elimination, Boolean operations, and

\(^2\) A *region* is a smallest state-space that can be characterized with a zone.
“+δ” operations of predicates. We let $ZP_c(X, P)$ be the set of all zone-predicates whose atoms are either atomic propositions in $P$ or inequalities like $x_1 - x_2 \sim d$ where $x_1, x_2 \in X \cup \{0\}$, ‘$\sim’ \in \{<,\leq\}$, and $|d| \leq c$. According to the literature [14, 19], we find that given zone-predicates in $ZP_{C(a_1, a_2)}(X, P)$ as their arguments, all such basic procedures yield zone-predicates in $ZP_{C(a_1, a_2)}(X, P)$. The challenge here is that some arguments in formulas ($f_B$) and ($f_C$) are not exactly zone-predicates. Our idea is to rearrange the arguments to those basic procedures so that they all appear as zone-predicates in $ZP_{C(a_1, a_2)}(X, G \cup L_1 \cup L_2)$ for some $X$.

After examining formulas ($f_B$) and ($f_C$), we find out that there are only two ways that we may yield non-zone-predicates.

6.1 Time-progress operations in formulas ($f_B$) and ($f_C$)

The first class of operations happens when we execute the “+δ” operation in formula ($f_B$) and when we call the “+t” and “+t’” operations in procedures $Tbck()$ and $Rbck_{t_2}(\_\_\_\_)$. In these cases, δ, t, t’ are not exactly clocks (i.e., their values do not change with time).

Now we focus on the case of “+δ.” The other two cases for $t, t’$ are similar. After operation “+δ,” we convert literals like $x \sim c$ or $-x \sim -d$ respectively to something like $x + δ \sim c$ and $-x - δ \sim -d$ which do not look like atomic constraints in zone-predicates. What we do is that we introduce a new dense variable ‘−δ’ and instead convert those literals to $x - (-δ) \sim c$ and $(-δ) - x \sim -d$. In this way, given any argument in $ZP_{C(a_1, a_2)}(X, P)$, the “+δ” operations (and “+t” and “+t’” operations) all yield zone-predicates in $ZP_{C(a_1, a_2)}(X \cup \{-δ, -t, -t’\}, P)$. So we can establish the following lemma.

**Lemma 4.** Given TEAs $A_1$ and $A_2$ and $η \in ZP_{C(a_1, a_2)}(G \cup L_1 \cup L_2, X_1 \cup X_2)$, $η + δ$ and $\text{path}(\neg η, δ)$ are both in $ZP_{C(a_1, a_2)}(G \cup L_1 \cup L_2, X_1 \cup X_2 \cup \{-δ\})$.

Note that the correctness of this first conversion relies on the fact that we never do a double time-progression operation like $(η + t) + t’$ in ($f_B$) and ($f_C$).

6.2 Measuring time-progress with δ and clock z in formula ($f_C$)

The second way that we may yield non-zone-predicate stems from equality $z = δ$ in formula ($f_C$). This equality is represented as zone-predicate $z - δ \leq 0 \land δ - z \leq 0$. This could make trouble since when we apply the “+t” (or “+t’”) operations in formula ($f_C$), the literals are converted to $z + t - δ \leq 0$ and $δ - z - t \leq 0$ since $δ$ does not change its values with time progress. Such literals are certainly not zone-predicates. One observation from formula ($f_A$) is that the quantification of $z$ appears inside that of $δ$. Thus in the scope of processing $z$-related predicates in formula ($f_C$), $δ$ is static and stays unchanged. Moreover, only the “+t” and “+t’” operations appear in formula ($f_C$) while no “+δ” operation does. Thus our idea is to use auxiliary clock ‘$z - δ$’ instead of $z$. Note that clock ‘$z - δ$’ is special in that its value may be less than zero. Then
equality $z = \delta$ is instead represented as $(z - \delta) \leq 0 \land -(z - \delta) \leq 0$ and $(z = \delta) + t$ yields $(z - \delta) - (-t) \leq 0 \land (-t) - (z - \delta) \leq 0$, which again falls in the syntax of zone-predicates. Theorem 1. Simulation

In this section, we present the following lemma that helps us adapt procedure Sim-

Lemma 5. Given TEAs $A_1$ and $A_2$ and $\eta \in ZPC(A_1, A_2)$, $Rbck_{E_1, l}(\eta, z = \delta \land \bigvee_{e_2 \in E_2^{(e_1)}} Xbck_{(e_1, e_2)}(\eta))$ is in ZPC(A_1, A_2) $(X_1 \cup X_2 \cup \{(z - \delta)\}, \delta, G, L_1 \cup L_2)$. 

With lemma 5, we can assume that formula $Rbck_{E_1, l}(\eta, z = \delta \land \bigvee_{e_2 \in E_2^{(e_1)}} Xbck_{(e_1, e_2)}(\eta))$ yields a zone-predicate $f_M$. Formula (K) can be evaluated as zone-predicates by replacing every occurrence of $z$ in $f_M$ with $0$. We can implement a procedure, replace($\eta, z$), that replace every occurrence of clock variable $'z - \delta'$ in zone-predicate $\eta$ with clock variable value $'z - \delta'$. Thus $\exists z (z = 0 \land f_M) \equiv \text{replace}(f_M, z)$.

6.3 Implementing Simulation_Check() with zones

Combining the techniques in the previous two subsections, we can establish the following three lemmas whose proof are omitted due to page-limit.

Lemma 6. When $A_1$ and $A_2$ are both TEAs, $f_A(e_1, Q)$ is equivalent to

$$\exists \delta \geq 0 \left( Xbck_{e_1, l}(H_1) + \delta \land \text{path}(\neg H_1, \delta) \land \neg \text{replace}(Rbck_{E_1, l}(Q, z = z - \delta = 0 \land \bigvee_{e_2 \in E_2^{(e_1)}} Xbck_{(e_1, e_2)}(Q)), z, 0) \right)$$

With lemmas 6, 4, and 5, the main result of this section is established as follows.

Theorem 1. Simulation_Check($A_1, A_2$) is implementable as an algorithm with zones.

7 Algorithm for NZ-simulation checking

In this section, we present the following lemma that helps us adapting procedure Simulation_Check() for the checking of NZ-simulation. Please be reminded that in definition 5, $NZ_1$ denotes a representation of the non-Zeno states in the reachable state-space of $A_1$. The construction of zone-predicates for $NZ_1$ was discussed in [14, 21].

Lemma 7. Given $A_i = (\Sigma, X_i, G, L_i, I_i, H_i, E_i, e_i, \pi_i), 1 \leq i \leq 2$, if $A_1 \preceq_{NZ} A_2$, then $A_1 \preceq_{NZ} A_2$ for some $Q$ such that for all $(\nu_1, \nu_2) \in Q$, $\nu_1 \models \text{NZ}_1$.

Proof: We assume there is $(\nu_1, \nu_2) \in Q$ such that $\nu_1$ is a Zeno state in $V_{A_1}$. According to definition 5, by deleting all such pairs from $Q$, we still get an NZ-simulation relation out of $Q$. Thus the lemma is proven.
Lemma 7 leads to the following algorithm for NZ-simulation-checking.

\[
\text{NZ-Simulation\_Check}(A_1, A_2)
\]
\[
/* A_i = \langle \Sigma, X_i, G, L_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i \rangle, 1 \leq i \leq 2 */ \{
\text{Construct } NZ_1 \text{ and let } A'_1 \text{ be } \langle \Sigma, X_1, G, L_1, I_1 \wedge NZ_1, NZ_1, E_1, \epsilon_1, \tau_1, \pi_1 \rangle. 
\text{Return } \text{Simulation\_Check}(A'_1, A_2). 
\}
\]

8 Experiments

We have implemented the techniques discussed in this manuscript in **RED** 7.0, a model-checker for TEAs and parametric safety analysis for LHAs based on CRD and HRD-technology [19, 20]. We have experimented with the following parameterized benchmarks with various numbers of processes. \( A_1 \) and \( A_2 \) differ in only one process.

- **Fischer’s timed mutual exclusion algorithm** [5]: The algorithm relies on a global lock and a local clock per process to control access to the critical section. Two timing constants used are 10 and 19. We use two versions of this benchmark, one with a simulation relation and one without.

- **CSMA/CD benchmark** [22]: This is the ethernet bus arbitration protocol with the idea of collision-and-retry. The timing constants used are 26, 52, and 808. We use three versions of this benchmark, one with an NZ-simulation relation, one with a simulation relation, and one without.

- **Timed consumer/producer**: There are a buffer, some producers, and some consumers in the benchmark. The producers periodically write data to the buffer. The consumers also periodically wipe out data, if any, in the buffer. We use two versions of this benchmark, one with the biggest timing constant 15 and a simulation relation while the other with the biggest timing constant 20 and without a simulation relation.

The performance data is reported in table 1. For each row, we report the computation time for constructing \( NZ_1 \) and the time for simulation-checking. The total memory consumption for the data-structures in state-space representations is also reported. In this experiment, we did not run benchmarks with large concurrency sizes. But according to the grow-rates of the memory consumptions, we predict that benchmarks with larger concurrency sizes could be passed with our program.

9 Conclusion

In this work, we present a characterization of the simulation relation between TEAs and derive a symbolic simulation-checking procedure out of this characterization. We then present techniques to implement the algorithm with zone-technology. It would be interesting to see what classes of LHAs can be verified with zone-technology using
### Table 1. Performance data of scalability w.r.t. various strategies

| benchmarks          | versions | time (non-Zeno restriction) | total time and memory |
|---------------------|----------|-----------------------------|-----------------------|
|                     |          | Simulation check            |                       |
| Fischer’s mutual    | 1        | 0.01s                       | 0.01s/23k             |
| exclusion (m        | 2        | 0.31s                       | 0.74s/90k             |
| processes           | 3        | 1.27s                       | 3.5s/201k             |
|                     | 4        | 4.24s                       | 12.5s/431k            |
|                     | 5        | 12.98s                      | 39.82s/897k           |
| CSMA/CD (1 bus+     | 1        | 0.27s                       | 0.38s/88k             |
| m senders)          | 2        | 0.74s                       | 1.73s/190k            |
|                     | 3        | 3.53s                       | 5.46s/390k            |
|                     | 4        | 10.02s                      | 17.90s/792k           |
| Consumer & producer | 1        | 0.06s                       | 0.07s/39k             |
| (1 buffer +1        | 2        | 0.24s                       | 0.27s/48k             |
| producer +m         | 3        | 0.62s                       | 0.67s/76k             |
| consumers)          | 4        | 2.01s                       | 2.09s/173k            |
|                     | 5        | 6.51s                       | 6.72s/403k            |
|                      |          | 0.06s                       | 0.08s/52k             |
|                      |          | 0.28s                       | 0.30s/60k             |
|                      |          | 0.70s                       | 0.92s/104k            |
|                      |          | 2.75s                       | 3.08s/245k            |
|                      |          | 10.64s                      | 11.56s/590k           |

Data collected on a Pentium 4 1.7GHz with 380MB memory running LINUX; s: seconds; k: kilobytes of memory in data-structure;

This technique. Our algorithm can also be adapted to handle the effect of Zeno states. Finally, our implementation and experiment shows the promise that our algorithm could be useful in practice in the future.

**References**

1. R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.
2. R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho. Hybrid Automata: an Algorithmic Approach to the Specification and Verification of Hybrid Systems. in Proceedings of Workshop on Theory of Hybrid Systems, LNCS 736, Springer-Verlag, 1993.
3. R. Alur, D.L. Dill. A Theory of Timed Automata. Theoretical Computer Science, 126:183-235, 1994.
4. L. Aceto, A. Ingólfsdóttir, M.L. Pedersen, J. Poulsen. Characteristic Formulae for Timed Automata. Theoretical Informatics and Applications, 2000, Vol. 34, No.6, pp.565-584. EDP Sciences.
5. F. Balarin. Approximate Reachability Analysis of Timed Automata. IEEE RTSS, 1996.
6. J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, Wang Y. Verification of an Audio Protocol with Bus Collision Using UPPAAL. CAV 1996, LNCS 1119, Springer-Verlag.
7. D. Beyer. Efficient Reachability Analysis and Refinement Checking of Timed Automata Using BDDs. CHARME 2000, LNCS 2144, Springer-Verlag.
8. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Wang Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. Hybrid Control System Symposium, 1996, LNCS, Springer-Verlag.
9. K. Cerans. Decidability of bisimulation equivalence for parallel timer processes. CAV’92, LNCS 663, Springer-Verlag.
10. D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV’89, LNCS 407, Springer-Verlag.
11. J.B. Fourier. (reported in:) Analyse des travaux de l’Académie Royale des Sciences pendant l’année 1824, Partie Mathématique, 1827.
12. N. Halbwachs. Delay Analysis in Synchronous Programs. CAV’93, LNCS 697, Springer-Verlag.
13. M.R. Henzinger, T.A. Henzinger, P.W. Kopke. Computing simulations on finite and infinite graphs. IEEE FOCS’95.
14. T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.
15. H. Lin, Wang Y. Axiomatising timed automata. Acta Informatica, Vol.38, Nr.4, 2002, pp.277-305.
16. A. Nakata. Symbolic Bisimulation Checking and Decomposition of Real-Time Service Specifications. Ph.D. Dissertation, Faculty of the Engineering Science, Osaka University, 1997.
17. A. Shaw. Communicating Real-Time State Machines. IEEE Transactions on Software Engineering 18(9), September, 1992.
18. S. Taşkın, R. Alur, R.P. Kurshan, R.K. Brayton. Verifying abstractions of timed systems. CONCUR’1996, LNCS 1119, Springer-Verlag.
19. F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures, STTT (Software Tools for Technology Transfer), Vol. 6, Nr. 1, June 2004, Springer-Verlag; special issue for the 4th VMCAI, Jan. 2003, LNCS 2575, Springer-Verlag.
20. F. Wang. Symbolic Parametric Safety Analysis of Linear Hybrid Systems with BDD-like Data-Structures. IEEE Transactions on Software Engineering, Volume 31, Issue 1 (January 2005), pp. 38-51, IEEE Computer Society. A preliminary version is in proceedings of 16th CAV, 2004, LNCS 3114, Springer-Verlag.
21. F. Wang, G.-D. Huang, F. Yu. TCTL Inevitability Analysis of Dense-Time Systems: From Theory to Engineering. IEEE Transactions on Software Engineering, Vol. 32, Nr. 7, July 2006, IEEE Computer Society.
22. S. Yovine. Kronos: A Verification Tool for Real-Time Systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Nr. 1/2, October 1997.
**APPENDIX**

A Definition of non-Zeno simulation and equivalence

Suppose we are given two TEAs $A_1$ and $A_2$ such that $A_i = (\Sigma, X_i, G, L_i, I_i, H_i, E_i, \epsilon_i, \tau_i, \pi_i)$, $1 \leq i \leq 2$, and two states $\nu_1 \in V_{A_1}$ and $\nu_2 \in V_{A_2}$. Let $NZ_1$ be a representation of the non-Zeno states in the reachable state-space of $A_1$. An $NZ$-simulation $Q$ from $A_1$ to $A_2$ is a binary relation from $V_{A_1}$ to $V_{A_2}$ that satisfies the following requirement. For every $(\nu_1, \nu_2) \in Q$, $\nu_1$ and $\nu_2$ agrees on interpretation of variables in $G$ and for every $\delta \in \mathbb{R}^\geq$ and transition $e_1$ of $E_1$ such that for all $\hat{\delta} \in [0, \delta]$, $\nu_1 + \delta \vdash H_1$ and $\nu_1 + \delta \xrightarrow{e_1} \nu_1'$, there are $\nu_2' \in V_{A_2}$ and $e_2 \in E_2^{(e_1)}$ such that $stutter_2(\nu_1, \nu_2, \delta, \nu_2, Q)$, $\nu_2' \xrightarrow{e_2} \nu_2'\pi_2(e_2)$, and $(\nu_1', \nu_2'\pi_2(e_2)) \in Q$. Or in logic notations,

$$\forall (\nu_1, \nu_2) \in Q \forall e_1 \in E_1 \forall \delta \in \mathbb{R}^\geq \left( \begin{array}{c}
\nu_1 + \delta \vdash \pi(e_1) \\
\langle \forall \theta \leq \delta \leq (\nu_1 + \delta) \vdash H_1, (\nu_1 + \delta)\pi_1(e_1) \vdash ennz_1 \land H_1 \rangle \\
\xrightarrow{\pi(e_1)} \exists \nu_2' \in V_{A_2} \left( \begin{array}{c}
\nu_2' \vdash \pi(e_1) \\
\exists e_2 \in E_2^{(e_1)} \left( \begin{array}{c}
\nu_2' \xrightarrow{e_2} \nu_2'\pi_2(e_2) \\
\land (\nu_1 + \delta)\pi_1(e_1) \vdash (\nu_2, \pi_2(e_2)) \in Q \right) \\
\land \exists e_2 \in E_2^{(e_1)} \left( \begin{array}{c}
\nu_2' \xrightarrow{e_2} \nu_2'\pi_2(e_2) \\
\land (\nu_1 + \delta)\pi_1(e_1) \vdash (\nu_2, \pi_2(e_2)) \in Q \right) \end{array} \right) \right) \right)$$

If there is an NZ-simulation relation $Q$ from $A_1$ to $A_2$ such that for every state $\nu_1 \models I_1 \land NZ_1$, there is a $\nu_2 \models I_2$ such that $(\nu_1, \nu_2) \in Q$, we denote $A_1 \preceq Q A_2$. If $\exists Q(A_1 \preceq Q A_2)$, we say $A_1$ implements (or is simulated by) $A_2$, in symbols $A_1 \preceq A_2$. Two TEAs $A_1$ and $A_2$ are equivalent (i.e. bisimulate), in symbols $A_1 \equiv A_2$, if $A_1 \preceq A_2$ and $A_2 \preceq A_1$.

B Fourier-Motzkin elimination for the special case

There are two cases to discuss. The first case is for calculating $\exists p_1(f(p_1, \ldots, p_m, x_1, \ldots, x_n))$. According to Shannon expansion, we have

$$\exists p_1(f(p_1, \ldots, p_m, x_1, \ldots, x_n)) \equiv f(\text{false}, \ldots, p_m, x_1, \ldots, x_n) \lor f(\text{true}, \ldots, p_m, x_1, \ldots, x_n)$$

The second case, for the calculation of $\exists x_1(f(p_1, \ldots, p_m, x_1, \ldots, x_n))$, can be handled with the following steps.

1. Rewriting $f(p_1, \ldots, p_m, x_1, \ldots, x_n)$ in disjunctive normal form.
2. For each disjunct,

   (2.1) rewrite the linear constraints in one of the following three forms.
   - **TYPE I:** $\sum_{2 \leq i \leq n} a_i x_i \sim -a_1 x_1 + d$ when $a_1 > 0$.
   - **TYPE II:** $\sum_{2 \leq i \leq n} b_i x_i \sim' -b_1 x_1 - d'$ when $b_1 < 0$.
   - **TYPE III:** $\sum_{2 \leq i \leq n} c_i x_i \sim d$ when $c_1 = 0$.

   (2.2) for each TYPE I constraint $\sum_{2 \leq i \leq n} a_i x_i \sim -a_1 x_1 + d$ and TYPE II constraint $\sum_{2 \leq i \leq n} b_i x_i \sim' -b_1 x_1 - d'$, conjunct the following constraint to the disjunct.
\[
\sum_{2 \leq i \leq n} (a_i|b_1| + b_i|a_1|x_i \sim'' |b_1|d + |a_1|d')
\]
where \(\sim'' = \leq\) if both \(\sim = \leq\) and \(\sim = \leq\); and \(\sim'' = <\) otherwise.

(2.3) delete every constraint with a non-zero coefficient for \(x_1\) in the disjunct.