SRI International

CSL Technical Report SRI-CSL-01-04 • September 7, 2001

TEMPO: A Model Checker for Event-Recording Automata

Maria Sorea*

This research was supported by the National Science Foundation under grants CCR-00-82560 and CCR-00-86096.

*Also affiliated with: Universität Ulm, Abt. Künstliche Intelligenz, Germany.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Abstract

We present a symbolic on-the-fly model checking algorithm for event-recording automata, a subclass of timed automata. This algorithm is based on a forward reachability analysis and uses a symbolic representation of clock constraints. It forms the core of the verification tool Tempo. We also develop a real-time logic for specifying properties of event-recording automata in a suitable way and demonstrate that the model checking problem for this logic is decidable.

Contents

List of Figures

Chapter 1

Introduction

Reactive systems engage in continuous interaction with their environment. These systems are ubiquitous in the world of computing and include many concurrent and real-time systems, embedded and process-control systems, and operating systems. Correct design and correct implementation of reactive systems is of utmost importance since these systems are often deployed in safety-critical environments. A formal approach to the correct realization of reactive systems is of particular interest since it guarantees correctness with respect to a formal specification by means of proof. Reactive systems can be described, for example, by means of transition systems, automata, or process algebras whereas formal specifications of reactive systems are usually stated in terms of temporal or modal logic formula (e.g. [Koz83, Sti87, Eme90]).

Given a description (implementation) $\mathcal{M}$ of a reactive program and a property (specification) φ, the problem of *model checking* is to show that the implementation $\mathcal{M}$ satisfies φ. Model checking techniques [CGP99] have been successfully used for verifying many complex reactive systems. Traditional techniques for model checking, however, are not suitable for the verification of real-time (reactive) systems since they do not support an explicit modeling of time. Formalisms such as timed automata [AD94] and timed transition systems [HMP92] have been introduced to model the behavior of real-time systems. A timed automaton is a finite-state automaton augmented with a finite set of real-valued clocks, which can be set to zero simultaneously with a transition. The clocks proceed at the same rate and measure the amount of time that has elapsed since they were reset. With each transition a clock constraint is associated, and transitions fire only if the current value of the clocks satisfies these constraints. Verification tools such as Kronos [DOTY96] and Uppaal [LPY97] have been used successfully to verify properties of timed automata. In many cases, it suffices to include only time differences between occurrences of certain events in the model. Although the relative occurrence of events can be expressed in terms of timed automata by introducing and resetting clocks, the direct association between clocks and events is usually lost.

Event-clock automata [AFH99] have been introduced to maintain such a correspondence between events and clocks. Unlike the timed automata model, where the clocks are maintained by the system, event-clock automata have a fixed, predefined association between clocks and input symbols. Although event-clock automata are strictly less expressive than timed automata, most real-time systems that come up in practice can still be modeled in this restricted formalism, and, arguably, event-clock automata models are often more succinct than corresponding timed automata models, because, in contrast to timed automata, the value of clocks is determined solely by the input.

Since an event-clock automaton can easily be expressed as a timed automaton, verification tools like Kronos or Uppaal are immediately applicable to the analysis of event-clock automata. These tools use variants of real-time logics such as TCTL [ACD93] or $\mathcal{L}_\nu$ [LLW95]. One characteristic of these logics is that the formulas usually include new clocks, which are not present in the model. Moreover, these logics include operators for introducing formula clocks. For example, the $\mathcal{L}_\nu$ formula (x in φ) introduces a formula clock x and initializes it to 0. By restricting ourselves to the model of event-clock automata, we develop a real-time logic that closely reflects the event-clock model. In particular, there are no operators for introducing and resetting clocks. This logic includes, for each possible event a, a *since* operator $S(a)$ for measuring the time since the last occurrence of this event. In addition, there are real-time modalities such as $\langle\, S(b) \leq 3, a \,\rangle \mathtt{tt}$, which expresses the fact that event a must occur at most three time units after the last occurrence of event b. Since our logic also includes fixpoint operators, it can be viewed as a real-time extension of the well-known modal μ-calculus [Koz83]. The model checking problem for our real-time logic is easily shown to be decidable.

Tempo is a tool for verifying event-clock automata. Its input is a network of event-clock automata and it computes a symbolic representation of the set of reachable states. The underlying algorithm is based on the standard technique of forward reachability, and, for efficiency considerations, the product of the network automata is computed on-the-fly. The reachability algorithm in Tempo is parameterized over data structures for representing and manipulating clock constraints. Thus, it is a suitable platform for comparing different representations. Currently, Tempo includes a *polyhedra* library [Wil93] and a *difference decision diagram* library [MLAH99a] for representing and manipulating clock constraints.

The rest of the paper is organized as follows. Event-clock automata and their properties are presented in Chapter 2. Chapter 3 presents the verification tool Tempo, and contains the description of the reachability algorithm and some examples that we have verified using Tempo. The real-time fixpoint logic is introduced in Chapter 4. Chapter 5 contains concluding remarks and an overview of work in progress.

Chapter 2

Event-Clock Automata

Event-clock automata were introduced by Alur, Fix, and Henzinger [AFH99] as a determinizable subclass of timed automata. Event-clock automata are characterized by a fixed, predefined association between the clocks and the symbols of the input alphabet. Two types of clocks are identified, *event-recording* and *event-predicting* clocks. For the purpose of this paper we restrict ourself to event-recording clocks. The event-recording clock $S(a)$ of the input symbol a gives the time of the last occurrence of the event a relative to the current time. Similar models are described by Schneider, Bloom, and Marzullo [SBM92] and Shankar [Sha93]. Unlike in the model of timed automata, the resets of the clocks are not controlled by the system. Instead, all clock values are determined by the input word.

2.1 Event-recording clocks

Let Σ be a finite alphabet. Every symbol $a \in \Sigma$ is associated with an event-recording clock $S(a)$. For a given event-recording automaton, the set of event-recording clocks is denoted by $C_\Sigma = \{S(a) | a \in \Sigma\}$. If the alphabet Σ is clear from the context, the subscript Σ is omitted. An infinite *timed word* w over the alphabet Σ is an infinite sequence $(a_0, t_0)(a_1, t_1) \cdots$ of symbols $a_i \in \Sigma$, paired with nonnegative real numbers $t_i \in \mathbb{R}^{\geq 0}$ such that the sequence $t = t_0 t_1 \cdots$ of time-stamps is nondecreasing (that is, $t_i \leq t_{i+1}$, for all $i \geq 0$). Without loss of generality we assume that $t_0 = 0$.

Given an infinite timed word $w = (a_0, t_0)(a_1, t_1) \cdots$, the value of the clock variable $S(a)$ at position j of w equals $t_j - t_i$, where i is the last position preceding j with $a_i = a$. If there is no such position i then the value of $S(a)$ is undefined and is denoted by $\perp$. In the following, the set of nonnegative reals (nonnegative rationals) will be denoted by $\mathbb{R}^{\geq 0}$ ($\mathbb{Q}^{\geq 0}$). We write $\mathbb{R}^{\geq 0}_{\perp}$ ($\mathbb{Q}^{\geq 0}_{\perp}$) for the set of nonnegative real (rational) numbers together with the value $\perp$. A clock-valuation function $v : C_\Sigma \to \mathbb{R}^{\geq 0}_{\perp}$ assigns a (positive) real or the undefined value to each clock. Clock constraints compare clock values with rational constants or the undefined value $\perp$.

Definition 1 Let w be a timed word. For each position $j \geq 0$ in w, the *clock-valuation function* v_j^w is a mapping from C_Σ to $\mathbb{R}_\perp^{\geq 0}$ defined by

$$v_j^w(S(a)) = \begin{cases} t_j - t_i & \text{if there exists a position } i \text{ such that } 0 \leq i < j \\ & \text{and } a_i = a, \text{ and for all } k \text{ with } i < k < j, a_k \neq a \\ \perp & \text{if } a_k \neq a \text{ for all } 0 \leq k < j. \end{cases}$$

Definition 2 Given a set C_Σ of clock variables (or simply *clocks*), $S(a) \in C_\Sigma$ an arbitrary clock, $\gamma \in \mathbb{Q}_\perp^{\geq 0}$, and $\sim \in \{\leq, \geq, <, >, =\}$, the set $\Phi(C_\Sigma)$ of *clock constraints* over C_Σ is defined by the grammar

$$g := \top \mid S(a) \sim \gamma \mid S(a) - S(b) \sim \gamma \mid g_1 \wedge g_2.$$

Usually, comparisons between two clocks such as $S(a) - S(b) \sim \gamma$ are not included in the event-clock automata model. However, as for the timed automata model [AD94], such an extension does not change the expressiveness of event-recording automata. Clock constraints over C_Σ are interpreted with respect to clock-valuation functions: atomic constraints of the form $\perp \sim \perp$ and $\perp - \perp \sim \perp$ evaluate to true, all other comparisons that involve $\perp$ (e.g., $\perp \leq 2$) evaluate to false. For a clock-valuation function v and a clock constraint g over C_Σ, we write $v \models g$ (to be read as "v *satisfies* g") to denote that according to the values given by v the constraint g evaluates to true. Formally, $v \models g$ is defined inductively over the syntactic structure of g, where $S(a), S(b) \in C$ are arbitrary clocks, $\gamma \in \mathbb{Q}_\perp^{\geq 0}$ and $\sim \in \{\leq, \geq, <, >, =\}$:

$$\begin{aligned} v &\models \top \\ v &\models S(a) \sim \gamma && \text{iff} \quad v(S(a)) \sim \gamma \\ v &\models S(a) - S(b) \sim \gamma && \text{iff} \quad v(S(a)) - v(S(b)) \sim \gamma \\ v &\models g_1 \wedge g_2 && \text{iff} \quad v \models g_1 \text{ and } v \models g_2 \end{aligned}$$

For $\delta \in \mathbb{R}^{\geq 0}$, $v + \delta$ denotes the clock valuation that maps each defined clock $S(a) \in C$ to the value $v(S(a)) + \delta$ and equals $\perp$ for all undefined clocks. For a clock $S(a) \in C$, $v[S(a) := 0]$ denotes the clock valuation for C that maps $S(a)$ to the value 0 and agrees with v over the rest of the clocks.

2.2 Syntax and Semantics

An event-recording automaton is a finite graph augmented with a finite set of *clocks*. The vertices are called *locations* and the edges *transitions*. Transitions are instantaneous, and time can elapse only in locations. On a transition the clock corresponding to the taken event is implicitly reset to zero. At any point the value of a clock $S(a)$ equals the time elapsed since the last occurrence of the event a. With each transition a clock constraint is associated. The transition may be taken only if the current values of the clocks satisfy this constraint. We

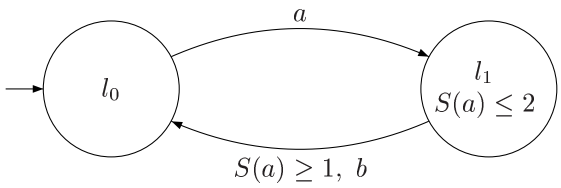


Figure 2.1: An event-recording automaton.

augment the model of event-recording automata from [AFH99], by associating a clock constraint, called *invariant*, with each location. We require that time can elapse in a location only as long as its invariant remains true.

Definition 3 An *event-recording automaton* A is a tuple $\langle L, L^0, \Sigma, I, E\rangle$ where

- Σ is a nonempty, finite input alphabet.
- L is a nonempty, finite set of locations.
- $L^0 \subseteq L$ is the set of initial locations.
- I is a mapping that labels each location l in L with some clock constraint.
- $E \subseteq L \times L \times \Sigma \times \Phi(C_\Sigma)$ is a finite set of edges. An edge $e = \langle l, l', a, g\rangle$ represents a transition from location l to location l' with the symbol a. The clock constraint g specifies when the transition is enabled.

Example 1 The event-recording automaton of Figure 2.1, for example, has two locations l_0 and l_1 and two clocks $S(a)$ and $S(b)$. Only the clock $S(a)$ is explicitly referred within a timing constraint. The initial location is l_0. The first occurrence of event a is at time $t = 0$. Whenever the system moves from location l_0 to location l_1 on input symbol a, the clock $S(a)$ is implicitly reset to 0. In l_1, the value of $S(a)$ yields the time since the last occurrence of the a event, that is, since the last transition from l_0 to l_1. The transition from l_1 to l_0 is enabled only if the value of $S(a)$ is greater than or equal to 1. Location l_1 has the invariant $S(a) \leq 2$. Consequently, the system may stay in l_1 at most two time units after entering l_1. The next transition must be taken before the invariant condition is violated. After taking the b-transition, the system reaches location l_0, where it can spend an arbitrary amount of time, since l_0 has the (implicit) invariant *true* associated with it. Altogether, the automaton expresses the timing constraint that the delay between an a and the following b is always between 1 and 2 time units.

Definition 4 A *state* of an event-recording automaton A is a pair (l, v) where l is a location of A, and v a clock valuation for C. An *initial state* of A is of the form (l_0, v_0) where $l_0 \in L_0$ and v_0 maps all clocks in C to $\perp$.

Definition 5 (Concrete Semantics) The (concrete) semantics of an event-recording automaton A is defined by associating a labeled transition system $\mathcal{T}_A = \langle S_A, S_A^0, \Sigma \cup \mathbb{R}^{\geq 0}, \rightarrow \rangle$ with it, where S_A is the set of states of A, and S_A^0 is the set of initial states. Two types of transition are defined in $\mathcal{T}_A$:

1. A state can change due to elapse of time: given a state (l, v) and $\delta \geq 0$, then $(l, v) \xrightarrow{\delta} (l', v')$ iff
   - $l = l'$, and
   - for all $0 \leq \delta' \leq \delta$, $v + \delta'$ satisfies the invariant $I(l)$, and
   - for all symbols $b \in \Sigma$, if $v(S(b)) = \perp$ then $v'(S(b)) = \perp$, else $v'(S(b)) = v(S(b)) + \delta$.
2. A state can change by taking a transition between two locations: $(l, v) \xrightarrow{a} (l', v')$ iff there is an edge $\langle l, l', a, g \rangle$ such that v satisfies g, $v' = v[S(a) := 0]$, and $v' \models I(l')$.

Definition 6 (Reachability) Let A be an event-recording automaton. The state (l', v') is *reachable* from the state (l, v) if there is a (possibly empty) sequence of transitions $(l, v) \xrightarrow{\rho}^{*} (l', v')$, where $\rho \in \Sigma \cup \mathbb{R}^{\geq 0}$. The state (l, v) is a *reachable state* in A if (l, v) is reachable from some initial state of A.

2.3 Networks of event-recording automata

Complex systems are modeled as *networks of event-recording automata* (i.e. parallel composition). Given two event-recording automata A_1 and A_2 and a synchronization set S, the parallel composition $A_1 \|_S A_2$ is defined in the obvious way. The locations of the product automaton are pairs of locations of its constituent automata. The invariant of a new location consists of the conjunction of the invariants of the component locations. We distinguish between synchronous (i.e., $S \neq \emptyset$) and asynchronous (i.e., $S = \emptyset$) composition. Symbols that belong to the set S are used for synchronization and must be taken simultaneously by both automata.

Definition 7 Let $A_i = \langle L_i, L_i^0, \Sigma_i, I_i, E_i \rangle$, $i \in \{1, 2\}$ be two event-recording automata. The *product automaton* $A_1 \|_S A_2$ is the event-recording automaton $\langle L_1 \times L_2, L_1^0 \times L_2^0, \Sigma_1 \cup \Sigma_2, I, E \rangle$, where $I(l_1, l_2) = I(l_1) \wedge I(l_2)$, and the edges are defined as follows:

1. For $a \in S$, $\langle (l_1, l_2), (l_1', l_2'), a, g_1 \wedge g_2 \rangle \in E$ iff $\langle l_1, l_1', a, g_1 \rangle \in E_1$ and $\langle l_2, l_2', a, g_2 \rangle \in E_2$.
2. For $a \in \Sigma_1 \setminus \Sigma_2$, $\langle (l_1, l_2), (l_1', l_2), a, g \rangle \in E$ iff $\langle l_1, l_1', a, g \rangle \in E_1$ and $l_2 \in L_2$.
3. For $a \in \Sigma_2 \setminus \Sigma_1$, $\langle (l_1, l_2), (l_1, l_2'), a, g \rangle \in E$ iff $\langle l_2, l_2', a, g \rangle \in E_2$ and $l_1 \in L_1$.

The asynchronous composition is commutative (i.e., $A_1\|_\emptyset A_2 = A_2\|_\emptyset A_1$) and associative (i.e., $A_1\|_\emptyset(A_2\|_\emptyset A_3) = (A_1\|_\emptyset A_2)\|_\emptyset A_3$). The synchronous composition is also commutative. As long as the sets S_1 and S_2 are equal the synchronous composition is associative, but in the general case $A_1\|_{S_1}(A_2\|_{S_2}A_3)$ does not necessarily equal $(A_1\|_{S_1}A_2)\|_{S_2}A_3$.

Example 2 Consider an automatic control system that opens and closes a gate at a railroad crossing. The system consists of three components, *train, gate* and *controller*, each of them modeled by an event-recording automaton. The models are shown in Figure 2.2. Every component is modeled by an event-recording automaton with four locations. The train communicates with the controller via the events *approach* and *exit*. The event *in* marks the entry of the train in the railroad crossing, the event *out* the exit from the railroad crossing. The train must send the *approach* signal at least two time units before it enters the crossing. The timing constraint $S(a) > 2$ associated with the transition from location t_1 to location t_2 expresses this requirement.[1] The maximal delay between the events *approach* and *exit* is five time units. This is expressed by the invariant conditions in locations t_1, t_2, t_3. The gate and the controller communicate via *lower* and *raise*. The symbols *up* and *down* denote the actions of opening and closing of the gate. After receiving the signal *lower* the gate responds by closing within one time unit (expressed by the invariant condition in location g_1). Similarly, the gate responds to the signal *raise* within between one and two time units. This is expressed by the timing constraint on the transition from location g_3 to location g_0 and the invariant in location g_3. The controller sends the signal *lower* to the gate every time it receives the signal *approach* from the train. This happens exactly one time unit after the receiving of the signal from the train. By receiving the signal *exit* from the train, the controller responds with the signal *raise* to the gate after at most one time unit. The train-gate-controller system can be modeled as the product TRAIN $\|_{S_1}$ CONTROLLER $\|_{S_2}$ GATE, where $S_1 = \{approach, exit\}$ and $S_2 = \{lower, raise\}$.

[1] The events of the event-recording clocks are abbreviated to the initials.

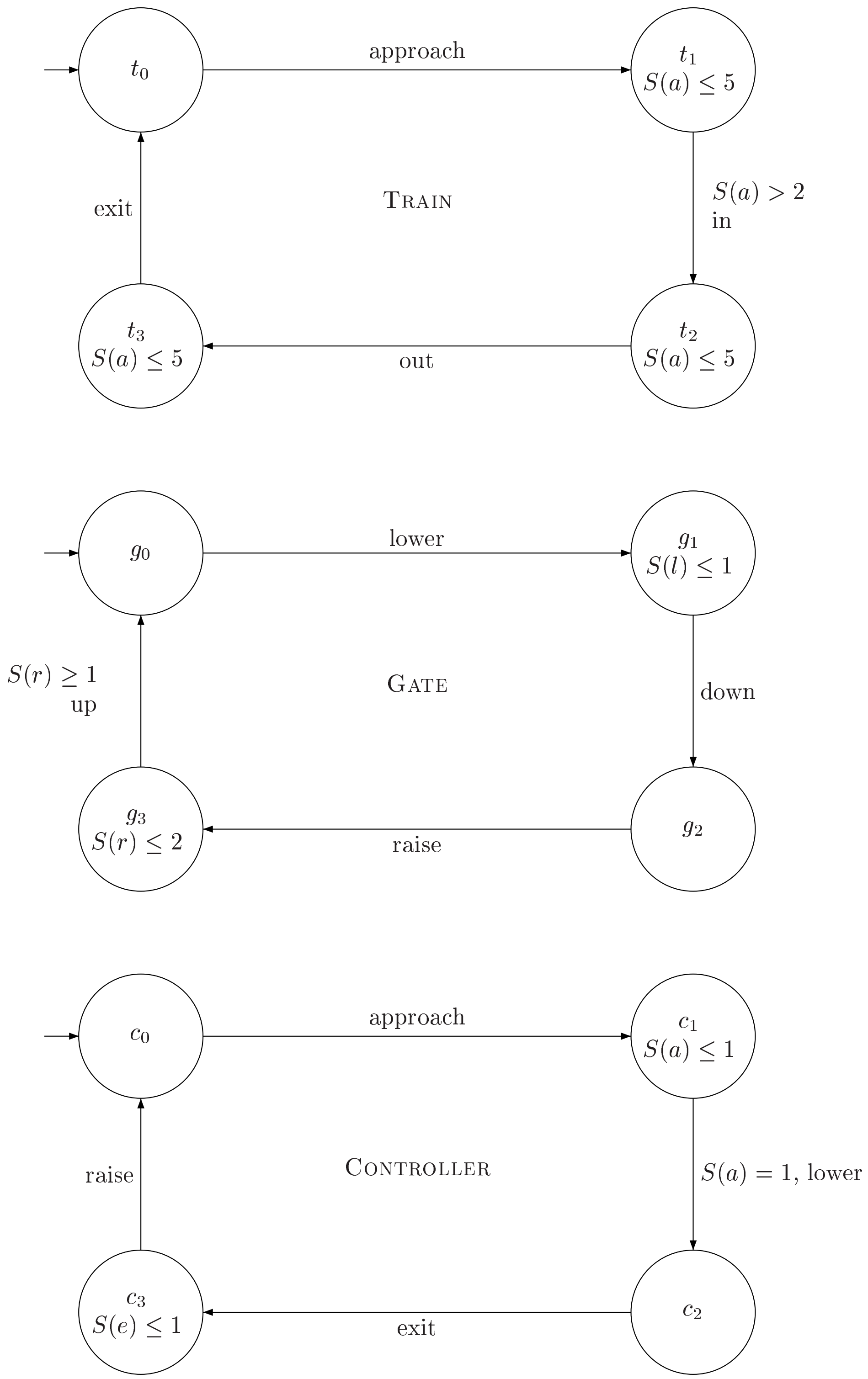


Figure 2.2: Train gate controller modeled as a network of event-recording automata.

Chapter 3

Tempo – a Model Checker for Event-Recording Automata

The reachability problem consists of computing the set of all states of an event-recording automaton that can be reached by starting from the state (l_0, v_0). Verification of safety properties can be formulated as a reachability problem. Consider, for example the train gate controller introduced in Figure 2.2. The safety requirement for the system is that the gate should be closed when the train is inside the railroad crossing. The train is inside the gate in location t_2. The gate is closed in location g_2. Therefore, in every reachable location of the product automaton, if the train is in location t_2 then the gate must be in location g_2. The timing constraints of the system ensure that no location with the train in t_2 and the gate not in g_2 can be reached.

3.1 Reachability Analysis

For finite-state systems, safety properties can be verified by checking that all the reachable states satisfy the desired property. But event-recodring automata are infinite state, since the clock values range over the real numbers. Decidability results for timed automata, however, can be obtained using the region construction [AD94]. The number of regions in this construction can be very large, as it grows exponentially with the number of clocks and the constants used in the clock constraints. Here, we present a different partition of the state space, which is based on the clock constraints of the given event-recording automaton. A similar construction for timed automata can be found in [LPY95].

Definition 8 (Constraint system) Let Σ be a finite alphabet. A *constraint* *system* is generated by

$$U ::= S(a) \sim \gamma \mid S(a) - S(b) \sim \gamma \mid U_1 \wedge U_2 \mid U_1 \vee U_2$$

where $S(a), S(b)$ are arbitrary clocks, $\sim \in \{<, \leq, =, >, \geq\}$, and $\gamma \in \mathbb{Q}_{\perp}^{\geq 0}$.

Each constraint denotes a set of clock valuations in the obvious way:

$$\begin{array}{rcl}
[\![S(a) \sim \gamma]\!] & = & \{v \mid v(S(a)) \sim \gamma\} \\
[\![S(a) - S(b) \sim \gamma]\!] & = & \{v \mid v(S(a)) - v(S(b)) \sim \gamma\} \\
[\![U_1 \wedge U_2]\!] & = & \{v \mid v \in [\![U_1]\!] \text{ and } v \in [\![U_2]\!]\} \\
[\![U_1 \vee U_2]\!] & = & \{v \mid v \in [\![U_1]\!] \text{ or } v \in [\![U_2]\!]\}
\end{array}$$

Notice that every interpretation of a constraint system can be expressed as a (possibly non-convex) union of regions [AD94]. Reachability analysis requires the following operations for manipulating constraint systems. Time elapse $\mathit{elapse}(U)$ denotes the set of clock valuations obtained by letting time elapse for all defined clocks in U, while $\mathit{reset}(S(a), U)$ denotes resetting clock $S(a)$ in U, and $U \bar{\wedge} U'$ and $U \bar{\vee} U'$ are simply intersection and union, respectively.

Definition 9 (Operations on clock zones) Let U, U' be constraint systems, $S(a)$ an arbitrary clock, and v a clock valuation function. Then $\mathit{elapse}(U)$, $\mathit{reset}(S(a), U)$, $U_1 \bar{\wedge} U_2$, $U_1 \bar{\vee} U_2$ are *operators on constraints* such that the following holds.

$$\begin{array}{rcl}
[\![\mathit{elapse}(U)]\!] & = & \{v + \delta \mid v \in [\![U]\!] \text{ and } \delta \in \mathbb{R}^{\geq 0}\} \\
[\![\mathit{reset}(S(a), U)]\!] & = & \{v[S(a) := 0] \mid v \in [\![U]\!]\} \\
[\![U_1 \bar{\wedge} U_2]\!] & = & [\![U_1]\!] \cap [\![U_2]\!] \\
[\![U_2 \bar{\vee} U_2]\!] & = & [\![U_1]\!] \cup [\![U_2]\!]
\end{array}$$

Each set obtained by the operators in Definition 9 is a denotation of some constraint system (see Proposition 1). This is well defined, since one can easily realize these operators for example as sets of *polyhedra* or a single *difference decision diagram*. Polyhedra are geometric representations of linear systems of equations and inequalities, while difference decision diagrams (DDD) have been specifically developed to symbolically represent difference constraint expressions, that is, inequalities of the form $x - y \leq c$, where x and y are real-valued variables and c is a rational constant. Difference constraint expressions are represented using a decision tree similar to the BDD representation of a Boolean expression.

The reset operator is implemented in the polyhedra representation by performing projection operations for each polyhedra in the set. The time elapse operator reduces in the polyhedra representation to adding a ray for each polyhedra in the set. A ray $\mathcal{K}$ is a vector r, such that $x \in \mathcal{K}$ implies $(x + \delta r) \in \mathcal{K}$ for all $\delta \geq 0$. Thus, by setting value 1 for defined clocks, and value 0 for undefined clocks it can be specified that time elapses only for defined clocks.

In the DDD representation, elapse of time is implemented by performing the operation of existential quantification of a special variable, which denotes the actual time (see [MLAH99b] for more details). However, this operation lets time elapse for all clocks of the system, which occurs also for the undefined clocks. Therefore, after performing the existential quantification, we must set the value of the undefined clocks to (-1), the value that represents $\bot$. The operation $reset(S(a), U)$ is implemented by the existential quantification of $S(a)$, followed by assigning $S(a)$ the value 0.

Proposition 1 Let Σ be a finite alphabet, $S(a)$ an arbitrary clock, and U, U' constraint systems. Then the results of $elapse(U)$, $reset(S(a), U)$, $U \bar{\wedge} U'$, and $U \bar{\vee} U'$ are also expressible as constraint systems.

Proof.

$\boxed{elapse(U)}$ We show, by induction over the structure of U, that $elapse(U)$ is expressible in terms of constraint system.

Case $U = S(a) \sim \gamma$

$$\begin{aligned}
& [\![elapse(U)]\!] \\
= \; & \{v + \delta \mid v \in [\![U]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \quad \{\text{by Definition 9}\} \\
= \; & \{v + \delta \mid v(S(a)) \sim \gamma \text{ and } \delta \in I\!R^{\geq 0}\} \quad \{\text{since } U = [S(a) \sim \gamma]\} \\
= \; & \left\{ \begin{array}{ll} \{v' \mid v'(S(a)) \sim \gamma + \delta\}, & \text{if } \gamma \neq \bot \\ \{v' \mid v' = v\} & \text{, otherwise} \end{array} \right. \quad \{\text{by Definition of } v + \delta\} \\
= \; & \left\{ \begin{array}{ll} [\![S(a) \sim \gamma + \delta]\!], & \text{if } \gamma \neq \bot \\ [\![S(a) \sim \gamma]\!] & \text{, otherwise} \end{array} \right. \quad \{\text{by Definition 8}\}
\end{aligned}$$

Thus, the result of $elapse(S(a) \sim \gamma)$ is expressible as a constraint system.

Case $U = S(a) - S(b) \sim \gamma$

$$\begin{aligned}
& [\![elapse(U)]\!] \\
= \; & \{\text{by Definition 9}\} \\
& \{v + \delta \mid v \in [\![U]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\
= \; & \{\text{since } U = [S(a) - S(b) \sim \gamma]\} \\
& \{v + \delta \mid v \in \{v(S(a)), v(S(b))\} \text{ and } v(S(a)) - v(S(b)) \sim \gamma\} \\
= \; & \{\text{by definition of } v + \delta\} \\
& \{v' \mid v'(S(a)) - v'(S(b)) \sim \gamma'\} \\
= \; & \{\text{by Definition 8}\} \\
& [\![S(a) - S(b) \sim \gamma']\!] \\
& \text{where } \gamma' = \left\{ \begin{array}{ll} \gamma & \text{, if } (v(S(a)) \neq \bot \text{ and } v(S(b)) \neq \bot) \\ \gamma - \delta, & \text{if } v(S(b)) = \bot \\ \gamma + \delta, & \text{if } v(S(a)) = \bot \end{array} \right.
\end{aligned}$$

Thus, the result of $elapse(S(a) - S(b) \sim \gamma)$ is expressible as a constraint system.

Case $U = U_1 \wedge U_2$

By induction hypothesis follows that both $elapse(U_1)$ and $elapse(U_2)$ are constraint systems. We have to show that $elapse(U_1 \wedge U_2)$ yields another constraint system.

1. From Definition 8 $[\![U_1 \wedge U_2]\!] = \{v \mid v \in [\![U_1]\!] \text{ and } v \in [\![U_2]\!]\}$.
2. Definition 9 together with 1. leads to:

$$\begin{array}{rl} & [\![elapse(U_1 \wedge U_2)]\!] \\ = & \{v + \delta \mid v \in [\![U_1 \wedge U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ = & \{v + \delta \mid v \in [\![U_1]\!] \text{ and } v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \end{array}$$

3. Also from Definition 9 we get

$$\begin{array}{rcl} [\![elapse(U_1)]\!] & = & \{v + \delta \mid v \in [\![U_1]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ [\![elapse(U_2)]\!] & = & \{v + \delta \mid v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \end{array}$$

4. From 3. follows

$$\begin{array}{rl} & [\![elapse(U_1)]\!] \cap [\![elapse(U_2)]\!] \\ = & \{v + \delta \mid v \in [\![U_1]\!] \text{ and } v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ = & \{v + \delta \mid v + \delta \in [\![elapse(U_1)]\!] \text{ and } v + \delta \in [\![elapse(U_2)]\!]\} \\ = & [\![elapse(U_1) \wedge elapse(U_2)]\!] \end{array}$$

   which by 2. leads to

$$[\![elapse(U_1 \wedge U_2)]\!] = [\![elapse(U_1) \wedge elapse(U_2)]\!].$$

Case $U = U_1 \vee U_2$

By induction hypothesis it follows that both $elapse(U_1)$ and $elapse(U_2)$ are constraint systems. We have to show that $elapse(U_1 \vee U_2)$ yields another constraint system. This follows similar to the case $U = U_1 \wedge U_2$.

1. From Definition 8 we get $[\![U_1 \vee U_2]\!] = \{v \mid v \in [\![U_1]\!] \text{ or } v \in [\![U_2]\!]\}$.
2. Definition 9 together with 1. yields:

$$\begin{array}{rl} & [\![elapse(U_1 \vee U_2)]\!] \\ = & \{v + \delta \mid v \in [\![U_1 \vee U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ = & \{v + \delta \mid v \in [\![U_1]\!] \text{ or } v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \end{array}$$

3. Also from Definition 9 we get

$$\begin{array}{rcl} [\![elapse(U_1)]\!] & = & \{v + \delta \mid v \in [\![U_1]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ [\![elapse(U_2)]\!] & = & \{v + \delta \mid v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \end{array}$$

4. From 3. follows

$$\begin{array}{rl} & [\![elapse(U_1)]\!] \cup [\![elapse(U_2)]\!] \\ = & \{v+\delta \mid v \in [\![U_1]\!] \text{ or } v \in [\![U_2]\!] \text{ and } \delta \in I\!R^{\geq 0}\} \\ = & \{v+\delta \mid v+\delta \in [\![elapse(U_1)]\!] \text{ or } v+\delta \in [\![elapse(U_2)]\!]\} \\ = & [\![elapse(U_1) \vee elapse(U_2)]\!] \end{array}$$

which by 2. leads to

$$[\![elapse(U_1 \vee U_2)]\!] = [\![elapse(U_1) \vee elapse(U_2)]\!].$$

$\boxed{reset(S(a), U)}$ We show, by induction over the structure of U, that $reset(S(a), U)$ is expressible in terms of constraint system.

Case $U = S(a) \sim \gamma$

$$\begin{array}{rl} & [\![reset(S(a), U)]\!] \\ = & \{\text{by Definition 9}\} \\ & \{v[S(a) := 0] \mid v \in [\![U]\!]\} \\ = & \{\text{since } U = [S(a) \sim \gamma]\} \\ & \{v[S(a) := 0] \mid v \in \{v(S(a)) \sim \gamma\}\} \\ = & \{v' \mid v'(S(a)) = 0\} \\ = & \{\text{by Definition 8}\} \\ & [\![S(a) = 0]\!] \end{array}$$

Thus, the result of $reset(a, [S(a) \sim \gamma])$ is expressible as a constraint system.

Case $U = S(a) - S(b) \sim \gamma$

$$\begin{array}{rl} & [\![reset(S(a), U)]\!] \\ = & \{\text{by Definition 9}\} \\ & \{v[S(a) := 0] \mid v \in [\![U]\!]\} \\ = & \{\text{since } U = [S(a) - S(b) \sim \gamma]\} \\ & \{v' \mid v'(S(a)) - v'(S(b)) \sim \gamma, \text{ where } v'(S(a)) = 0 \text{ and } v'(S(b)) = v(S(b))\} \\ = & \{\text{by Definition 8}\} \\ & [\![S(a) - S(b) \sim \gamma]\!] \end{array}$$

Thus, the result of $reset(a, [S(a) - S(b) \sim \gamma])$ is expressible as a constraint system.

Case $U = U_1 \wedge U_2$

By induction hypothesis it follows that both $reset(S(a), U_1)$ and

$reset(S(a), U_2)$ are constraint systems. We show that $reset(S(a), U_1 \wedge U_2)$ yields another constraint system.

$$
\begin{array}{rl}
& [\![reset(S(a), U_1 \wedge U_2)]\!] \\
= & \{\text{by Definition 9}\} \\
& \{v[S(a) := 0] \mid v \in [\![U_1 \wedge U_2]\!]\} \\
= & \{\text{by Definition 8}\} \\
& \{v[S(a) := 0] \mid v \in [\![U_1]\!] \text{ and } [\![U_2]\!]\} \\
= & \{v[S(a) := 0] \mid v \in [\![U_1]\!]\} \cap \{v[S(a) := 0] \mid v \in [\![U_2]\!]\} \\
= & \{\text{by Definition 9}\} \\
& [\![reset(S(a), U_1)]\!] \wedge [\![reset(S(a), U_2)]\!]
\end{array}
$$

Thus, the result of $reset(a, U_1 \wedge U_2)$ is expressible as a constraint system.

Case $U = U_1 \vee U_2$

By induction hypothesis it follows that both $reset(S(a), U_1)$ and $reset(S(a), U_2)$ are constraint systems. We have to show that the operation $reset(S(a), U_1 \vee U_2)$ yields another constraint system. This follows similar to the case $U = U_1 \wedge U_2$.

$\boxed{U_1 \bar{\wedge} U_2}$

$$
\begin{array}{rcl}
[\![U_1 \bar{\wedge} U_2]\!] & = & \{\text{by Definition 9}\} \\
& & [\![U_1]\!] \cap [\![U_2]\!] \\
& = & \{v \mid v \in [\![U_1]\!] \text{ and } v \in [\![U_2]\!]\} \\
& = & \{\text{by Definition 8}\} \\
& & [\![U_1 \wedge U_2]\!]
\end{array}
$$

Thus, the result of $U_1 \bar{\wedge} U_2$ is expressible as a constraint system.

$\boxed{U_1 \bar{\vee} U_2}$

$$
\begin{array}{rcl}
[\![U_1 \bar{\vee} U_2]\!] & = & \{\text{by Definition 9}\} \\
& & [\![U_1]\!] \cup [\![U_2]\!] \\
& = & \{v \mid v \in [\![U_1]\!] \text{ or } v \in [\![U_2]\!]\} \\
& = & \{\text{by Definition 8}\} \\
& & [\![U_1 \vee U_2]\!]
\end{array}
$$

Thus, the result of $U_1 \bar{\vee} U_2$ is expressible as a constraint system.

□

Definition 10 A *symbolic state* of a given event-recording automaton A is a pair (l, U) consisting of a location l and a constraint system U. A start symbolic state is a pair (l_0, U_0) with $l_0 \in L_0$ a start location of A and $U_0 = [\forall a \in \Sigma,\ S(a) = \bot]$. For a symbolic state (l, U) and an edge $e = \langle l, l', a, g\rangle$ of A, there is a symbolic transition $(l, U) \overset{a}{\rightsquigarrow} (l', U')$ where U' is the successor constraint system of U under e, $U' = \mathit{succ}(U, e) = \mathit{elapse}(\mathit{reset}(S(a), U \barwedge g)) \barwedge I(l')$.

Thus, the successor of a clock constraint is computed in four steps: (i) take the intersection of the actual constraint system U with the constraint g, (ii) reset the clock $S(a)$ in the obtained constraint system, (iii) let time elapse, and (iv) take the intersection with the invariant of the location l'. An example illustrates these four steps.

Example 3 Let $U = [S(a) = \bot \wedge S(b) = \bot]$, $e = \langle l_0, l_1, a, \mathit{true}\rangle$ and $I(l_1) = [S(a) \leq 2]$.

- After step (i), $U_1 = U \barwedge g = [S(a) = \bot \wedge S(b) = \bot]$ is computed.
- Step (ii) resets the clock $S(a)$, $U_2 = \mathit{reset}(S(a), U_1) = [S(a) = 0 \wedge S(b) = 0]$.
- Step (iii) let time elapse for U_2, $U_3 = \mathit{elapse}(U_2) = [S(a) \geq 0 \wedge S(b) = \bot]$.
- And finally by taking the intersection with the invariant of location l_1 we obtain $U' = U_3 \barwedge I(l_1) = [S(a) \geq 0 \wedge S(a) \leq 2 \wedge S(b) = \bot]$, which can be also written as $[0 \leq S(a) \leq 2 \wedge S(b) = \bot]$.

We use $(l_0, U_0) \overset{a}{\rightsquigarrow}^{*} (l', U')$ to denote that the symbolic state (l', U') is reachable from (l_0, U_0) through a (possibly empty) sequence of transitions, where only events from Σ are taken. The following theorem illustrates that the symbolic semantics of event-clock automata is a full characterization of the concrete.

Theorem 1 Given an event-recording automaton A, $\rho \in \Sigma \cup I\!R^{\geq 0}$, $a \in \Sigma$ then

- (*Soundness*) whenever $(l_0, U_0) \overset{a}{\rightsquigarrow}^{*} (l', U')$ then $(l_0, v_0) \xrightarrow{\rho}^{*} (l', v')$ for all $v' \in U'$.
- (*Completeness*) whenever $(l_0, v_0) \xrightarrow{\rho}^{*} (l', v')$ then $(l_0, U_0) \overset{a}{\rightsquigarrow}^{*} (l', U')$ for some U' and $v' \in U'$.

Proof. In the following we assume (without loss of generality) that all transition sequences $(l_0, v_0) \xrightarrow{\rho}^{*} (l', v')$ have alternating actions, where the first action is an event from Σ[1] that is, they are of the form

$$(l_0, v_0) \xrightarrow{a_1} (l_1, v_1) \xrightarrow{\delta_1} (l_1, v_1 + \delta) \xrightarrow{a_2} (l_2, v_2) \xrightarrow{\delta_2} (l_2, v_2 + \delta) \cdots \xrightarrow{a_n} (l_n, v_n) \xrightarrow{\delta_n} (l_n, v_n + \delta).$$

[1]This follows from the property of time additivity, that is, two transitions $(l_0, v_0) \xrightarrow{\delta_1} (l_1, v_1)$ and $(l_1, v_1) \xrightarrow{\delta_2} (l_2, v_2)$ are equivalent to the transition $(l_0, v_0) \xrightarrow{\delta_1 + \delta_2} (l_2, v_2)$, which in fact is the same as $(l_0, v_0) \xrightarrow{\delta_1 + \delta_2} (l_0, v_0 + \delta_1 + \delta_2)$.

The above sequence can also be written as

$$(l_0, v_0) \overset{(a_1,\delta_1)}{\longrightarrow} (l_1, v_1 + \delta) \cdots \overset{(a_n,\delta_n)}{\longrightarrow} (l_n, v_n + \delta).$$

I. Soundness. Assume $(l_0, U_0) \overset{a}{\rightsquigarrow}{}^{*} (l', U')$. We show by induction over the length of the above transition sequence, that $(l_0, v_0) \overset{(a,\delta)^*}{\longrightarrow} (l', v')$.

Basis Case. $(l_0, U_0) \overset{a}{\rightsquigarrow}{}^{0} (l_0, U_0)$, with $v_0 \in [\![U_0]\!]$. Then $(l_0, v_0) \overset{(a,\delta)^0}{\longrightarrow} (l_0, v_0)$.

Induction Hypothesis. If (l_n, U_n) is reachable from (l_0, U_0) with n a-transitions, and $v_n \in [\![U_n]\!]$, then (l_n, v_n) is also reachable from (l_0, v_0) in n (a, δ)-transitions, that is

$$(l_0, U_0) \overset{a}{\rightsquigarrow}{}^{n} (l_n, U_n) \text{ and } v_n \in [\![U_n]\!] \text{ implies } (l_0, v_0) \overset{(a,\delta)}{\longrightarrow}{}^{n} (l_n, v_n)$$

Induction Step. That is, (l_{n+1}, U_{n+1}) is reachable from (l_0, U_0) in $n + 1$ steps, that is,

$$(l_0, U_0) \overset{a_1}{\rightsquigarrow} \cdots \overset{a_n}{\rightsquigarrow} (l_n, U_n) \overset{a_{n+1}}{\rightsquigarrow} (l_{n+1}, U_{n+1})$$

where U_{n+1} is the successor constraint of U_n.
By Induction Hypothesis it follows that (l_n, v_n) is reachable form (l_0, v_0) for all $v_n \in [\![U_n]\!]$. From $(l_n, U_n) \overset{a_{n+1}}{\rightsquigarrow} (l_{n+1}, U_{n+1})$ it follows that there exists an edge $e = \langle l_n, l_{n+1}, a_{n+1}, g_{n+1} \rangle \in E$, such that

$$U_{n+1} = \mathit{elapse}(\mathit{reset}(S(a_{n+1}), U_n \mathbin{\bar\wedge} g_{n+1})) \mathbin{\bar\wedge} I(l_{n+1})$$

And for all $v_n \in [\![U_n]\!]$ we obtain for all $v_{n+1} := v_n[S(a) := 0] + \delta_{n+1}$ that $v_{n+1} \in [\![U_{n+1}]\!]$. Thus, $(l_n, v_n) \overset{(a_{n+1},\delta_{n+1})}{\longrightarrow} (l_{n+1}, v_{n+1})$.

II. Completeness. Assume $(l_0, v_0) \overset{(a,\delta)^*}{\longrightarrow} (l', v')$. We show by induction over the length of the above transition sequence, that $(l_0, U_0) \overset{a}{\rightsquigarrow}{}^{*} (l', U')$ for $v' \in U'$.

Basis Case. $(l_0, v_0) \overset{(a,\delta)}{\longrightarrow}{}^{0} (l_0, v_0)$. Then $(l_0, U_0) \overset{a}{\longrightarrow}{}^{0} (l_0, U_0)$, where $U_0 = v_0$.

Induction Hypothesis. If (l_n, v_n) is reachable from (l_0, v_0) within n (a, δ)-transitions, then (l_n, U_n) is also reachable from (l_0, U_0) in n a-transitions, that is

$$(l_0, v_0) \overset{(a,\delta)}{\longrightarrow}{}^{n} (l_n, v_n) \text{ implies } (l_0, U_0) \overset{a}{\longrightarrow}{}^{n} (l_n, U_n) \text{ and } v_n \in [\![U_n]\!]$$

Induction Step. That is, (l_{n+1}, v_{n+1}) is reachable from (l_0, v_0) in $n + 1$ (a, δ)-steps,

$$(l_0, v_0) \overset{(a_1,\delta_1)}{\longrightarrow} \cdots \overset{(a_n,\delta_n)}{\longrightarrow} (l_n, v_n) \overset{(a_{n+1},\delta_{n+1})}{\longrightarrow} (l_{n+1}, v_{n+1}),$$

where $v_{n+1} = v_n[S(a) := 0] + \delta$ and $v_{n+1} \mathrel{|\!\approx} I(l_{n+1})$.

By induction hypothesis it follows that $(l_0, U_0) \xrightarrow{a}{}^{n} (l_n, U_n)$ and $v_n \in [\![U_n]\!]$. We show that $(l_n, U_n) \xrightarrow{a_{n+1}} (l_{n+1}, U_{n+1})$ and $v_{n+1} \in [\![U_{n+1}]\!]$ hold.

From $(l_n, v_n) \xrightarrow{(a_{n+1}, \delta_{n+1})} (l_{n+1}, v_{n+1})$ we obtain:

$$\exists e = \langle l_n, l_{n+1}, a_{n+1}, g_{n+1} \rangle \in E.\ v_n \approx g_{n+1} \tag{3.1}$$

$$v_{n+1} := v_n[S(a_{n+1}) := 0] + \delta_{n+1} \text{ and } v_{n+1} \approx I(l_{n+1}) \tag{3.2}$$

From (3.1) and (3.2) and Definition 9 we get

$$(l_n, U_n) \overset{a_{n+1}}{\rightsquigarrow} (l_{n+1}, U_{n+1})$$

and

$$U_{n+1} = \mathit{elapse}(\mathit{reset}(S(a_{n+1}), U_n \bar{\wedge} g_{n+1})) \bar{\wedge} I(l_{n+1}), \text{ with } v_{n+1} \in [\![U_{n+1}]\!].$$

□

Now, Theorem 1 can be strengthened.

Corollary 1 Given an event-recording automaton A, a state (l', v') is reachable from an initial state (l_0, v_0) through a sequence of n (a, δ)-transitions if and only if the symbolic state (l', U') is reachable from the initial symbolic state (l_0, U_0) through a sequence of n a-transitions.

Although the symbolic semantics of event-recording automata is coarser than the concrete semantics, it is not an adequate basis for a reachability algorithm because it is still infinite, since the number of clock constraints in each symbolic state is unbounded. The solution to this problem is to consider only constraint systems in which the clocks are compared only with constants less or equal to the largest constant k appearing in the guards of the given automaton. Similar to [WT94] we define a restricted form of clock constraints called *rounded clock constraints*. First, we introduce an equivalence relation $\approx_{cz}$, which partition the clock assignments into *constraint zones*. These constraints zones are used in describing rounded constraint systems. The below presentation partially follows [WT94].

Definition 11 Let $A = \langle L, L^0, \Sigma, I, E \rangle$ be an event-clock automaton with C_Σ the corresponding set of event-recording clocks. Let $x_i = S(a_i)$, $x_j = S(a_j)$ denote two arbitrary event-recording clocks. Further let k be the largest constant appearing in the clock constraints and invariants of A. The equivalence relation $\approx_{cz}$ is defined over the set of clock assignments for C_Σ. We say $v \approx_{cz} v'$ iff they satisfy the exact same subset of *legal constraints* of the form:

- *Primary constraints*: $v(x_i) \leq c,\ v(x_i) < c,\ v(x_i) \geq c,\ v(x_i) > c,\ v(x_i) = \perp$, where $c \leq k$,
- *Secondary constraints*: $v(x_i) - v(x_j) \leq c,\ v(x_i) - v(x_j) < c$, where $-k \leq c \leq k$.

The constraint zones induced by two clocks $S(a)$ and $S(b)$ with $c = 2$ are shown in Figure 3.1. These regions differs from the regions defined by Alur and Dill in [AD94] by the representation of the secondary constraints by the extension of the diagonals.

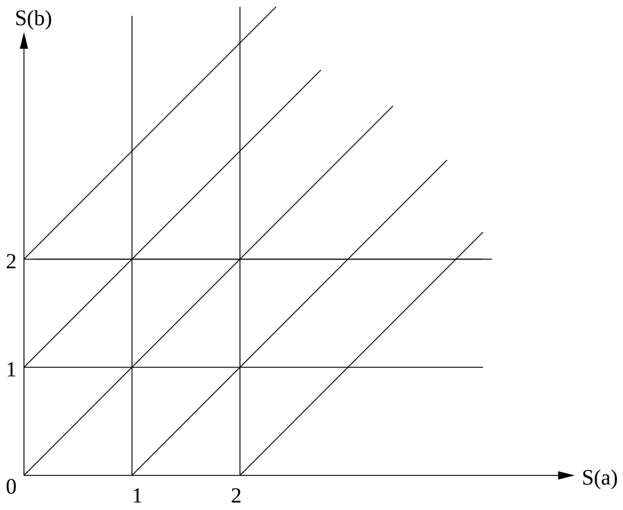


Figure 3.1: Constraint zones.

Definition 12 A *rounded constraint system* is a union of constraint zones.

Definition 13 The function *round* maps any constraint system to the intersection of all rounded constraint systems, which includes it, that is:

$$round(U) = \bigcap\{U' \mid U \subseteq U' \text{ and } U' \text{ is a rounded constraint system}\}$$

The following lemma is due to Wong-Toi [WT94].

Lemma 1 A constraint system U is a rounded constraint system iff it is definable as a conjunction of a set of legal constraints, that is, there exists a set of legal constraints Θ such that

$$U = \{v \mid v \text{ satisfies every constraint in } \Theta\}$$

For a given constraint system U the rounded constraint system $\mathcal{U}$ can be obtained by applying the following *widening* operator on U.

Definition 14 Let U be a constraint system and $k \in I\!\!N$ a positive natural number. The *k-widening of U*, denoted by $widening_k(U)$ is defined inductively over the structure of U:

- $widening_k(S(a) = \bot) \stackrel{\text{def}}{=} S(a) = \bot$

- $widening_k(S(a) \sim \gamma) \stackrel{\text{def}}{=} \begin{cases} S(a) \sim \gamma & \text{iff } \gamma \leq k \\ S(a) < \infty & \text{else} \end{cases}$
  for $\sim \in \{<, \leq\}$

- $widening_k(S(a) - S(b) \sim \gamma) \stackrel{\text{def}}{=} \begin{cases} S(a) - S(b) \sim \gamma & \text{iff } \gamma \leq k \\ S(a) - S(b) < \infty & \text{else} \end{cases}$
  for $\sim \in \{<, \leq\}$

- $widening_k(S(a) \sim \gamma) \stackrel{\text{def}}{=} \begin{cases} S(a) \sim \gamma & \text{iff } \gamma \leq k \\ S(a) > k & \text{else} \end{cases}$
  for $\sim \in \{>, \geq\}$

- $widening_k(S(a) - S(b) \sim \gamma) \stackrel{\text{def}}{=} \begin{cases} S(a) - S(b) \sim \gamma & \text{iff } \gamma \leq k \\ S(a) - S(b) > k & \text{else} \end{cases}$
  for $\sim \in \{>, \geq\}$

- $widening_k(S(a) = \gamma) \stackrel{\text{def}}{=} widening_k(S(a) \leq \gamma \wedge S(a) \geq \gamma)$

- $widening_k(U_1 \wedge U_2) \stackrel{\text{def}}{=} widening_k(U_1) \wedge widening_k(U_2)$

- $widening_k(U_1 \vee U_2) \stackrel{\text{def}}{=} widening_k(U_1) \vee widening_k(U_2)$

Intuitively, $widening_k(U)$ is the clock constraint U where all upper bounds greater than k are removed (replaced with ∞) and all lower bounds greater than k are replaced with k. In [Pet99] Pettersson defines a similar normalization operator.

Lemma 2 For a given constraint system U the round function and the widening operator produce the same results, that is, $[\![round(U)]\!] = [\![widening_k(U)]\!]$.
Proof. Follows by induction over the structure of U and by Lemma 1. $\square$

Figure 3.2 illustrates the rounded constraint system for $U = [2 \leq S(b) \leq 3 \wedge S(a) - S(b) \leq 0]$ and $k = 2$. This is obtained by applying the widening operator on U: $widening_2(U) = [S(b) \geq 2 \wedge S(a) - S(b) \leq 0]$.

For a given event-recording automaton the rounded constraint systems are the union of Alur-Dill equivalence classes (i.e., regions). And since the number of such regions is finite [AD94] there are only finitely many rounded clock constraints. Proposition 2 states this fact.

Proposition 2 Given an event-recording automaton A over an alphabet Σ. Let k be the greatest constant appearing in the clock constraints of A. The set of denotations $[\![widening_k(U)]\!]$ is finite.

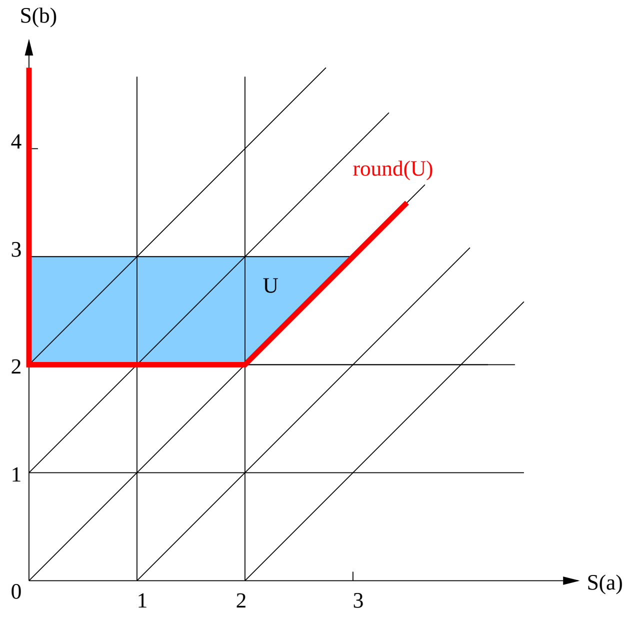


Figure 3.2: Example for a rounded constraint system.

Definition 15 (Rounded Symbolic Transitions) Let A be an event-recording automaton, $a \in \Sigma$, $l, l' \in L$ two locations and U, U' two constraint systems. Then the *rounded symbolic transition relation* $\overset{a}{\Longrightarrow}$ is defined as follows:

$$(l, \mathit{widening}_k(U)) \overset{a}{\Longrightarrow} (l', \mathit{widening}_k(U')) \ \text{ iff } \ (l, U) \overset{a}{\rightsquigarrow} (l', U'),$$

where $U' = \mathit{elapse}(\mathit{reset}(S(a), U \bar{\wedge} g)) \bar{\wedge} I(l')$.

The following lemma is due to Wong-Toi [WT94].

Lemma 3 The rounding function is sound, that is *round(U)* introduces no states that are not included in the Alur-Dill regions of U.

Thus, every reachable state in the rounded symbolic semantics (Definition 15) of a given event-recording automaton is also reachable in the concrete semantics, which is defined in Section 2.1. Therefore, for a given event-recording automaton the reachability algorithm can be performed over rounded symbolic states $(l, \mathcal{U})$, of which there are only finitely many.

Recall, that we have introduced three types of semantics: the concrete semantics, with the transition relation $\xrightarrow{a,\delta}$, the symbolic semantics with the transition relation $\overset{a}{\rightsquigarrow}$, and the rounded semantics with the transition relation $\overset{a}{\Longrightarrow}$. The following figure illustrates the inclusion relations (to be read counter clockwise) between the three transition relations.

$$\begin{array}{ccccc} \xrightarrow{a,\delta} & & \supseteq & & \overset{a}{\Longrightarrow} \\ & \subseteq & & \subseteq & \\ & & \overset{a}{\leadsto} & & \end{array}$$

The inclusion

$$\begin{array}{rl} & \xrightarrow{a,\delta} \subseteq \overset{a}{\leadsto} \\ \overset{\text{def}}{=} & \{(l',v') \mid (l_0,v_0) \xrightarrow{a,\delta}{}^{*} (l',v')\} \subseteq \{(l',U') \mid (l_0,U_0) \overset{a}{\leadsto}{}^{*} (l',U') \text{ and } v' \in U'\} \end{array}$$

follows from Theorem 1. The inclusion

$$\begin{array}{rl} & \overset{a}{\leadsto} \subseteq \overset{a}{\Longrightarrow} \\ \overset{\text{def}}{=} & \{(l',U') \mid (l_0,U_0) \overset{a}{\leadsto}{}^{*} (l',U')\} \subseteq \{(l',\mathcal{U}') \mid (l_0,\mathcal{U}_0) \overset{a}{\Longrightarrow}{}^{*} (l',U') \text{ and } U' \in \mathcal{U}'\} \end{array}$$

follows directly from Definition 15, and

$$\begin{array}{rl} & \overset{a}{\Longrightarrow} \subseteq \xrightarrow{a,\delta} \\ \overset{\text{def}}{=} & \{(l',\mathcal{U}') \mid (l_0,\mathcal{U}_0) \overset{a}{\Longrightarrow}{}^{*} (l',\mathcal{U}')\} \subseteq \{(l',v') \mid (l_0,v_0) \xrightarrow{a,\delta}{}^{*} (l',v') \text{ and } v' \in \mathcal{U}'\} \end{array}$$

from Lemma 3. The following theorem summarizes the results and shows that the rounded symbolic semantics characterizes the concrete semantics.

Theorem 2 Given an event-recording automaton A, $\rho \in \Sigma \cup \mathbb{R}^{\geq 0}$, $a \in \Sigma$, with k the largest constant appearing in the guards and the invariants of A. Then

$$\begin{array}{rcl} (l_0, \mathit{widening}_k(U_0)) & \overset{a}{\Longrightarrow}{}^{*} & (l', \mathit{widening}_k(U')) \quad \text{iff} \\ (l_0, v_0) & \xrightarrow{\rho}{}^{*} & (l', v') \quad \text{for all } v' \in \mathit{widening}_k(U') \end{array}$$

Proof. Follows from Theorem 1, Definition 15, and Lemma 3. □

Thus, a reachability analysis can be performed over the rounded symbolic states, instead over the individual states for a given event-recording automaton.

In what follows, all operations on constraint systems are performed on rounded constraint systems. For a constraint system U we denote by $\mathcal{U}$ the constraint system obtained by rounding U.

Definition 16 A *set of rounded symbolic states* $\mathcal{S}_A$ of a given event-recording automaton A is a set of pairs $(l,\mathcal{U})$, where $l \in L$ and $\mathcal{U}$ is a rounded constraint system. On $\mathcal{S}_A$ the following operations are defined, where the function $\mathit{def} : \mathcal{U} \to C_\Sigma$ denotes for a given constraint system $\mathcal{U}$ the set of defined clocks.

$$\mathit{add}(l,\mathcal{U})(\mathcal{S}_A) = \begin{cases} (\mathcal{S}_A \setminus (l,\mathcal{U}')) \cup (l, \mathcal{U} \mathbin{\bar{\nabla}} \mathcal{U}'), & \text{if } \exists \mathcal{U}'.\,(l,\mathcal{U}') \in \mathcal{S}_A \\ & \text{with } \mathit{def}(\mathcal{U}) = \mathit{def}(\mathcal{U}') \\ \mathcal{S}_A \cup (l,\mathcal{U}), & \text{otherwise} \end{cases}$$

$$\mathit{remove}(l,\mathcal{U})(\mathcal{S}_A) = \begin{cases} \mathcal{S}_A \setminus (l,\mathcal{U}), & \text{if } \exists\,(l,\mathcal{U}) \in \mathcal{S}_A \\ \mathcal{S}_A, & \text{otherwise} \end{cases}$$

reachable(A, k) (3.1)
  waiting $= \{(l_0,\mathcal{U}_0)\}$; *visited* $= \emptyset$; (3.2)
  while *waiting* $\neq \emptyset$ **do** (3.3)
    $(l,\mathcal{U}) :=$ choose from *waiting*; (3.4)
    remove $(l,\mathcal{U})$ from *waiting*; (3.5)
    if $\exists (l',\mathcal{U}') \in$ *visited* with $l = l'$ and $\mathcal{U} \subseteq \mathcal{U}'$ (3.6)
    then skip (3.7)
    else (3.8)
      for all $\langle l, l_1, a, g\rangle \in Edges(A)$ **do** (3.9)
        $\mathcal{U}_1 := widening_k(elapse(reset(S(a),\mathcal{U} \bar{\wedge}\, g)) \bar{\wedge}\, I(l_1))$; (3.10)
        if $\mathcal{U}_1 \neq \emptyset$ **then** add $(l_1,\mathcal{U}_1)$ to *waiting* **endif** (3.11)
      endforall (3.12)
    endif; (3.13)
    add $(l,\mathcal{U})$ to *visited* (3.14)
  done; (3.15)
  return *visited* (3.16)
end reachable. (3.17)

Figure 3.3: Forward reachability analysis.

The algorithm in Figure 3.3 computes the set of all symbolic states $(l_i,\mathcal{U}_i)$ that are reachable from the start symbolic state $(l_0,\mathcal{U}_0)$ for a given event-recording automaton. In the following, this set is denoted by *Reach*. Two data structures are used for storing symbolic states: *visited* contains the set of symbolic states that have already been examined, while *waiting* contains the symbolic states to be visited. The algorithm starts with *waiting* containing the start symbolic state $(l_0,\mathcal{U}_0)$ and *visited* the empty set (line (2)). If for a symbolic state $(l,\mathcal{U}) \in$ *waiting* there is a symbolic state $(l',\mathcal{U}') \in$ *visited* with $l' = l$, and $\mathcal{U} \subseteq \mathcal{U}'$, then $(l,\mathcal{U})$ does not have to be examined any more (lines (6)-(7)). Otherwise, all the symbolic states that can be reached from $(l,\mathcal{U})$ in one step (line (9)-(10)) are added to *waiting* (line (11)). The algorithm terminates when *waiting* is empty and outputs the set *visited* that contains all the reachable symbolic states.

The following theorem states the correctness of the reachability algorithm from Figure 3.3.

Theorem 3 Let A be an event-recording automaton over the alphabet Σ, l_0 the initial state, and $\mathcal{U}_0 = \bigwedge_{a\in\Sigma} S(a) = \perp$ the initial rounded clock constraint. Then the reachability algorithm always terminates with the set containing all reachable rounded symbolic states that can be reach from $(l_0,\mathcal{U}_0)$.

Proof. By Proposition 1 and Definition 15 follows that all $\mathcal{U}_1$, which are computed in line (10) of the reachability algorithm, are rounded constraint systems. And since there are only finitely many rounded constraint systems, as stated in Proposition 2, the number of rounded symbolic states is also finite. □

3.2 On-the-fly reachability

The input of the reachability algorithm presented above consists of a single event-recording automaton. Therefore, for computing the set of reachable states for a given network of automata, the entire product automaton has to be computed first. Since this is inefficient, we present here a second method, which computes the reachable states on-the-fly, without computing the entire product automaton. Let N be a network of event-recording automata together with synchronization sets, $(A_1 \parallel_{S_1} A_2 \parallel_{S_2} \cdots \parallel_{S_{n-1}} A_n)$, where two automata A_i and A_{i+1} synchronize with each other over the set S_i of events. A location of the system is a tuple consisting of the n locations of the composing event-recording automata. The start location[2] of the system is given by $\tilde{l}_0 = (l_0^{A_1}, l_0^{A_2}, \cdots, l_0^{A_n})$,

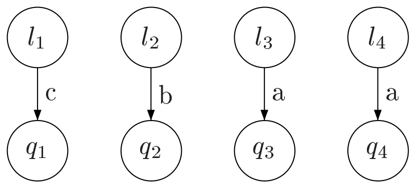
$$
\begin{array}{rl}
\textbf{transitions}(\mathit{system}, & \mathit{source}) \\
\textbf{match}\ \mathit{system} & \textbf{with} \\
A \ \rightarrow & e := \{\langle l, l', a, g\rangle \in \mathit{Edges}(A)\}; \\
(A_1 \parallel_{S_1} \cdots \parallel_{S_{n-1}} A_n) \ \rightarrow & e_1 := \{\langle l_1, l_1', a, g\rangle \in \mathit{Edges}(A_1)\}; \\
 & \mathit{sync} := S_1; \\
 & \mathit{system}' := (A_2 \parallel_{S_2} \cdots \parallel_{S_{n-1}} A_n); \\
 & \mathit{source}' := (l_2, \cdots, l_n); \\
 & e_2 := \mathrm{transitions}(\mathit{system}', \mathit{source}'); \\
 & e := \mathrm{product}(l_1, \mathit{source}', e_1, e_2, \mathit{sync}); \\
\textbf{return} & e \\
\textbf{end transitions}. &
\end{array}
$$

Figure 3.4: On-the-fly next step.

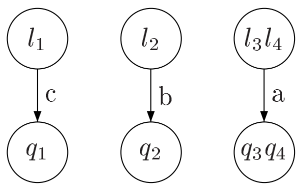
where $l_0^{A_i}$ is the start location of the ith event-recording automaton. The algorithm is similar to the forward reachability algorithm for one event-recording automaton presented above. The main difference is in the computation of one-step transitions that are possible from a given system location (line (9) in Figure 3.3). For on-the-fly reachability, this is done by the algorithm in Figure 3.4. For a given network of event-recording automata denoted by *system* in Figure 3.4, the algorithm computes all the transitions that can be taken in one step from a given list of locations, *source*, by the components of the network. The parameter *source* contains the actual locations of all the automata. If the network consists of only one event-recording automaton A, *transitions* is identical to line (9) in

[2] We assume that each event-recording automaton has exactly one start location.

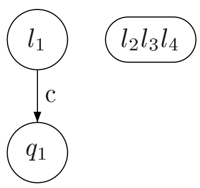
Figure 3.3. The procedure *product*(l_1, l_2, e_1, e_2, S) computes for two automata A_1 and A_2, which are in locations l_1, and l_2 respectively, the set of transitions that can be taken in one step from the compound location $l_1 l_2$ by the product automaton $A_1 \parallel_S A_2$ accordingly to the synchronization set S, where e_i denotes the set of transitions of A_i with source l_i, for $i \in \{1, 2\}$. An example illustrates the computation steps of the procedure *transitions*. Consider the following situation: given a network consisting of four automata, $A_1 \|_{\{a,b\}} A_2 \|_{\{a,b\}} A_3 \|_{\{a,b\}} A_4$. All four automata synchronize which each other over the set $S = \{a, b\}$. The automata are supposed to be in the locations l_1, l_2, l_3, l_4 respectively. We want to compute the set of all transitions that can be taken from the location $l_1 l_2 l_3 l_4$.



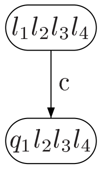
Four successive steps are performed. After the first step the following sets $e_1 = \{(l_3, a, q_3\}$ and $e_2 = \{(l_4, a, q_4)\}$ are computed. Therefore both automata A_3 and A_4 can perform synchronously an a-action. Thus, the set $e = \{(l_3 l_4, a, q_3 q_4)\}$ is computed by *product*. The resulting situation is illustrated by the following picture:



During the next step, the set of transitions that are possible from the location $l_2 l_3 l_4$ are computed. Now $e_1 = \{(l_2, b, q_2)\}$ and $e_2 = \{(l_3 l_4, a, q_3 q_4)\}$. Because the events a and b are both in the synchronization set no transition is simultaneously possible from the new location $l_2 l_3 l_4$. Therefore the set e is empty. After this step the situation is given by:



Finally, A_1, can perform the set $e_1 = \{(l_1, c, q_1)\}$ of events. Since the set e_2 is empty, $e = e_1 = \{(l_1, c, q_1)\}$ is the final result.



Consequently only the transition on the event c can be performed simultaneously by all four event-recording automata.

3.3 Implementation

The system TEMPO is an automatic verification tool for real-time systems based on the reachability algorithms for event-clock automata presented above. It is implemented in the functional language Ocaml. The (possibly non-convex) constraint systems are represented either by unions of polyhedra or by difference decision diagrams.

The algorithms in TEMPO are parameterized over a data type for representing and manipulating constraint systems. In its current incarnation, it includes two different instantiations based on polyhedra and DDD. A main point in the integration of the polyhedra and DDD libraries is the connection of the C data structures of the libraries with the Ocaml garbage collector in order to dynamically garbage collect constraint systems. With a flag on the command level, it is possible to choose between the desired representations. Other libraries for the representation of constraint systems, such as *clock difference diagrams* [LPWY99], can be added to TEMPO by suitably instantiating parameterized modules of TEMPO.

The input to TEMPO consists of a network of event-recording automata. Figure 3.5 shows the input file for the automaton from Figure 2.1. TEMPO is started with the command: "`./tempo [-p -m] [-t] [-c] [-o] filename.tempo`". The flags on the control level are used to set parameters:

`-p` If this flag is set, the polyhedra-based representation of constraint systems is active. Default is the DDD-based representation.

`-m` This is a specific parameter of the polyhedra library, which denotes the maximum number of rays.

`-t,-c` These are specific parameters of the DDD library, which set the maximal number of DDD entries and the size of the DDD cache. If these flags are not set, the default values are taken.

`-o` Switch between the on-the-fly and the naïve method for computing the set of reachable symbolic states. Default is the nïve method.

The output consists of the set of reachable symbolic states, represented as l `|->` $\mathcal{U}$, for a location l of the event-recording automaton, and a constraint system

```
era foo {                              % name of the automaton
    start l0                           % start location
    invariants                         % l1 has the invariant S(a)<=2,
        l1 |-> S(a) <= 2               % l2 has true as an invariant
    trans                              % transitions ...
        l0 -> l1 {a},                  % from l0 to l1 with an 'a' event
        l1 -> l0 {b, S(a) >= 1}        % from l1 to l0 with 'b',
                                       % and clock constraint S(a)>=1
}
```

Figure 3.5: Example of an input file for TEMPO.

$\mathcal{U}$. Using the input from Figure 3.5, the command `./tempo -o era.tempo` computes on-the-fly the following set of reachable states based on the DDD representation:

```
l0 |-> S(b)=-1 /\ S(a)=-1
l1 |-> S(b)=-1 /\ 0<=S(a)<=2
l0 |-> S(b)>=0 /\ S(a)>=1 /\ 1<=S(a)-S(b)<=2
l1 |-> S(b)>=0 /\ 0<=S(a)<=2 /\ S(a)<=S(b)
```

In the polyhedra-based representation the output is in the form of a system of equations and inequalities. The command `./tempo -p -m 50 -o era.tempo` produces the following result:

```
l0 |-> 1* S(b) + 1 = 0
       1* S(a) + 1 = 0

l1 |-> 1* S(b) + 1 = 0
       1* S(a) + 0 >= 0
      -1* S(a) + 2 >= 0

l0 |-> 1* S(b) + 0 >= 0
       1* S(b) + -1* S(a) + 2 >= 0
      -1* S(b) + 1* S(a) + -1 >= 0

l1 |-> 1* S(a) + 0 >= 0
       1* S(b) + -1* S(a) + 0 >= 0
      -1* S(a) + 2 >= 0
```

3.4 Examples

We present two examples for which the set of reachable symbolic states has been computed by TEMPO: the railroad crossing controller and Fischer's mutual exclusion protocol [Lam87].

Fischer's Mutual Exclusion Protocol. This protocol guarantees mutual exclusion between several processes using clock constraints and a shared variable. The shared variable and the processes are modeled as event-recording automata, and the entire protocol is modeled as a network of event-recording automata. This is a simplified version of the original protocol, which permits only one process to enter the critical section, which it never exits. Recovery actions for the case that a process fails to enter the critical section are omitted. The automata for Fischer's protocol with two processes can be found in Appendix A, the corresponding input file in Appendix B. We compute the set of reachable states for 2 to 6 processes on a Pentium III, 500 MHz. The resulting times (in seconds) are shown in Table 3.1.

No. of processes	Polyhedra		DDD	
	on product	on-the-fly	on product	on-the-fly
2	0.06	0.04	0.07	0.07
3	0.25	0.22	0.21	0.20
4	6.86	3.15	2.58	2.51
5	MO	52.41	34.6	29.8
6	MO	MO	MO	301

Table 3.1: Timings for computing the reachable states for Fischer's protocol in seconds (MO denotes memory overflow).

These timings must be taken with a grain of salt, since the polyhedra package uses static data structures, requiring the setting of parameters such as the maximal number of rays, which is quite sensitive to the problem at hand. For the DDD package we used fixed default values.

Railroad crossing. The input file for the railroad crossing controller can be found in Appendix C. On a Pentium III the set of all reachable symbolic states was computed in 0.33 seconds, by using polyhedra for representing clock constraints, and in 0.13 seconds by using DDDs.

It is not surprising that DDDs are both more time and space efficient than polyhedra for representing and manipulating the non-convex sets of clock constraints.

Chapter 4

Event-Recording Fixpoint Logic

By restricting ourselves to the model of event-recording automata, we develop a real-time logic, which closely reflects the event-recording automata model. The logic presented here includes, for each event a from a given finite alphabet Σ, a *since operator* for measuring the time since the last occurrence of this event. This operator has the same semantics as an event-recording clock in the model of event-recording automata. Therefore, we denote the since operator also by $S(a)$, for an arbitrary event a. Since our logic includes least and greatest fixpoint operators, it can be viewed as a real-time extension of the modal μ-calculus [Koz83]. The usual box and diamond modalities are augmented with constraints. For example, the real-time modality $\langle\, S(b) \leq 3, a \,\rangle\mathtt{tt}$ expresses the fact that event a must occur at most three time units after the last occurrence of event b.

4.1 Syntax and Semantics

Definition 17 (Syntax of ERL$_\mu$) Let Σ be a finite alphabet, g a timing constraint, and X a variable. The formulas of the *Event-Recording Fixpoint Logic* ERL$_\mu$ are defined by the following grammar:

$$\varphi ::= \mathtt{tt} \mid \mathtt{ff} \mid X \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [\,g, a\,]\varphi \mid \langle\, g, a \,\rangle\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

Given an event-recording automaton $A = \langle L, L^0, \Sigma, I, E\rangle$ together with the corresponding labeled transition system $\mathcal{M} = \langle Q_A, Q_A^0, \Sigma \cup \mathbb{R}^{\geq 0}, \rightarrow\rangle$, the semantics of an ERL$_\mu$ formula is given by the set of states $(l, v) \in Q_A$, for which the formula holds. Subformulas containing free variables are dealt with using *valuation functions* $\vartheta : V \rightarrow \mathcal{P}(Q_A)$, where V denotes the set of free variables. The updating notation $\vartheta[X := (l, v)]$ denotes the valuation ϑ' that agrees with ϑ on all variables except X, where $\vartheta'(X) = (l, v)$.

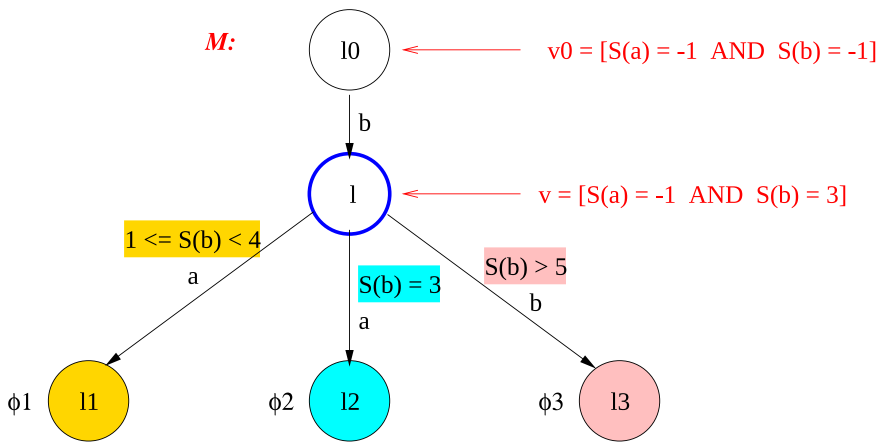


Figure 4.1: A small example.

Definition 18 (Semantics of ERL$_\mu$) Let $\mathcal{M}$ be a model, and $\vartheta : V \to \mathcal{P}(Q)$ an assignment function, the set of states (l, v) in which a formula φ is true, $[\![\varphi]\!]^{\mathcal{M}}_{\vartheta}$, is defined inductively as follows:

$$
\begin{array}{rcl}
[\![\mathtt{tt}]\!]^{\mathcal{M}}_{\vartheta} & := & Q \\
[\![\mathtt{ff}]\!]^{\mathcal{M}}_{\vartheta} & := & \emptyset \\
[\![X]\!]^{\mathcal{M}}_{\vartheta} & := & \vartheta(X) \\
[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{M}}_{\vartheta} & := & [\![\varphi_1]\!]^{\mathcal{M}}_{\vartheta} \cap [\![\varphi_2]\!]^{\mathcal{M}}_{\vartheta} \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{M}}_{\vartheta} & = & [\![\varphi_1]\!]^{\mathcal{M}}_{\vartheta} \cup [\![\varphi_2]\!]^{\mathcal{M}}_{\vartheta} \\
[\![\langle\, g, a\,\rangle\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \{(l, v) \in Q \mid \exists l' \in \{\tilde{l} \,|\, \langle l, \tilde{l}, a, \tilde{g}\rangle \in E\}, \exists \delta \geq 0. \\
 & & \quad (\, v \approx g \wedge \tilde{g} \text{ and } v' = v[S(a) := 0] + \delta \approx I(l')\,) \\
 & & \quad \text{and } (l', v') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![[\, g, a\,]\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \{(l, v) \in Q \mid \forall l' \in \{\tilde{l} \,|\, \langle l, \tilde{l}, a, \tilde{g}\rangle \in E\}, \forall \delta \geq 0. \\
 & & \quad (\, v \approx g \wedge \tilde{g} \text{ and } v' = v[S(a) := 0] + \delta \approx I(l')\,) \\
 & & \quad \text{implies } (l', v') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![\mu X.\varphi(X)]\!]^{\mathcal{M}}_{\vartheta} & := & \cap\{Q' \subseteq Q \mid [\![\varphi(X)]\!]^{\mathcal{M}}_{\vartheta[X:=Q']} \subseteq Q'\} \\
[\![\nu X.\varphi(X)]\!]^{\mathcal{M}}_{\vartheta} & := & \cup\{Q' \subseteq Q \mid Q' \subseteq [\![\varphi(X)]\!]^{\mathcal{M}}_{\vartheta[X:=Q']}\}
\end{array}
$$

We also write $\mathcal{M}, (l, v), \vartheta \models \varphi$, to denote that $(l, v) \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}$. The subscript ϑ is omitted if the formula does not contain free variables (φ is a closed formula).

Example 4 The following examples of ERL$_\mu$ formulas demonstrate the expressiveness of ERL$_\mu$. The satisfiability of these formulas is illustrated using the model $\mathcal{M}$ from Figure 4.1. Here v_0 is the initial clock valuation and v is the clock valuation in location l.

- $F_1 = \langle\, S(b) \geq 1 \wedge S(b) \leq 3, a\,\rangle \phi_1$

  F_1 states that there is a time point at least 1 time unit later and not later than 3 time units than the precedent b, such that an a event occurs, which leads to a state in which the formula ϕ_1 is true. This formula holds in state (l_0, v_0) of $\mathcal{M}$: $\mathcal{M}, (l_0, v_0) \models F_1$.

- $F_2 = [\, S(b) \geq 1 \wedge S(b) \leq 3, a\,] \phi_1$

  For all time points between 1 and 3 time units after the precedent b, for all a which are then possible, a state in which ϕ_1 holds will be reached. This formula does not hold in $\mathcal{M}$ at state $(l_0.v_0)$, since there is an a-transition from l exactly 3 time units after the b, which leads to the state (l_2, v_2) where ϕ_1 does not hold: $\mathcal{M}, (l_0, v_0) \not\models F_2$.

- $F_3 = \langle\, S(b) = 3, a\,\rangle \phi_1$

  This formula is true in $\mathcal{M}$ at state (l_0, v_0) since there is a time point exactly 3 time units later than the preceding b, such that an a-event is possible. $\mathcal{M}, (l_0, v_0) \models F_3$.

- $F_4 = [\, S(b) \geq 4, a\,]\mathtt{ff}$

  Four time units after a b no more a events are allowed to occur. $\mathcal{M}, (l_0, v_0) \models F_4$.

Proposition 3 Let φ be a ERL_μ-formula with free variables and ϑ a valuation function. The functional $\varphi(X)$ is monotonic in the sense that if $[\![X]\!]_\vartheta \subseteq [\![Y]\!]_\vartheta$ then $[\![\varphi(X)]\!]_\vartheta \subseteq [\![\varphi(Y)]\!]_\vartheta$.
Proof. The proof follows by induction over the structure of φ. □

Lemma 4 Let $F(X) := [\![\varphi(X)]\!]^{\mathcal{M}}_\vartheta$ be a monotonic functional. Then, for a sufficiently large ordinal $\mathcal{O}$, the least (greatest) fixpoint $[\![\mu X.\varphi(X)]\!]^{\mathcal{M}}_\vartheta$ ($[\![\nu X.\varphi(X)]\!]^{\mathcal{M}}_\vartheta$) of this functional can be computed by:

$$[\![\mu X.\varphi(X)]\!]^{\mathcal{M}}_\vartheta := \bigcup_{i \in \mathcal{O}} F^i(\emptyset) \qquad \left([\![\nu X.\varphi(X)]\!]^{\mathcal{M}}_\vartheta := \bigcap_{i \in \mathcal{O}} F^i(\mathcal{S}_A)\right)$$

Proof. Follows by Knaster-Tarski fixpoint theorem. □

Lemma 5 For a model $\mathcal{M}$, a state (l, v), and a fixpoint formula $\mu X.\varphi \in \mathrm{ERL}_\mu$ ($\nu X.\varphi$) the following holds:

$$\begin{array}{lll} \mathcal{M}, (l, v) \models \mu X.\varphi & \text{iff} & \exists i_0 \in \mathcal{O}.\mathcal{M}, (l, v) \models \varphi^{i_0}(\mathtt{ff}) \\ \mathcal{M}, (l, v) \models \nu X.\varphi & \text{iff} & \exists i_0 \in \mathcal{O}.\mathcal{M}, (l, v) \models \varphi^{i_0}(Q) \end{array}$$

Proof. Follows immediately from Definition 18 and Lemma 4 and the fact that all possible functionals are monotonic. □

Similar to [Sti96], we use the following abbreviations. Assume that Σ is a finite set of actions. For a subset K of Σ the set $\Sigma \setminus K$ is abbreviated by $-K$. We write $-a_1, \ldots, a_n$ for $-\{a_1, \ldots a_n\}$. Further "$-$" is used to abbreviate $-\emptyset$, which is equal to Σ. The operator $[-]$ expresses "next" over all actions in Σ. Immediate "necessity" is expressed by $\langle - \rangle$. CTL properties are defined in the usual way:

$$\begin{aligned} EF\varphi &\stackrel{\text{def}}{=} \mu X.\, \varphi \vee \langle - \rangle X \\ AG\varphi &\stackrel{\text{def}}{=} \nu X.\, \varphi \wedge [-] X \\ AF\varphi &\stackrel{\text{def}}{=} \mu X.\, \varphi \vee [-] X \\ EG\varphi &\stackrel{\text{def}}{=} \nu X.\, \varphi \wedge \langle - \rangle X \end{aligned}$$

The ERL$_\mu$ logic allows the specification of safety and liveness properties. Informally, a *safety* property states that some bad feature is always excluded. Safety can either be related to states, that bad states can never be reached, or to actions, that bad actions never happen. The safety property of the train gate controller from Figure 2.2 is that whenever the train is in the crossing the gate should be closed. This corresponds to establishing that in the system TRAIN $\|_{S_1}$ CONTROLLER $\|_{S_2}$ GATE, the event *in* is never taken before the event *down*.[1] This property can be specified by the following formula:

$$\varphi = AG\Big([\, S(down) = \bot, in\,]\mathtt{ff} \wedge [\, S(down) > S(up), in\,]\Big)\mathtt{ff}$$

which is equivalent to

$$\varphi = \nu Z.\, [\, S(down) = \bot, in\,]\mathtt{ff} \wedge [\, S(down) > S(up), in\,]\mathtt{ff} \wedge [-] Z.$$

The safety property of Fischer's mutual exclusion protocol is that there is never more than one process in its critical section. For two processes this property can be specified by the following formula:

$$\varphi = AG\Big([\, v = 1\,]\mathtt{ff} \vee [\, v = 2\,]\mathtt{ff}\Big).$$

A liveness property states that some good feature is eventually fulfilled. Again it can be related to states, that a good state is eventually reached, or to actions, that a good action eventually occurs. A real-time liveness property of the train gate controller from Figure 2.2 is that the gate is never closed for more than 10 seconds. This is specified by the following formula:

$$\varphi = EF\Big(\langle\, S(down) \le 10, up\,\rangle \mathtt{tt}\Big)$$

which is equivalent to

$$\varphi = \mu X.\, \langle\, S(down) \le 10, up\,\rangle \vee \langle - \rangle X.$$

The following definitions are used for the proofs at the end of this section.

[1] This is equivalent to the property, described at the beginning of Section 3, that the train cannot be in location t_2 without the gate being in location g_2.

Definition 19 (Length of a formula) The length function, $l : \mathrm{ERL}_\mu \to \mathbb{N}$ for a formula $\varphi \in \mathrm{ERL}_\mu$, is defined inductively over the structure of the formula:

$$
\begin{aligned}
l(\top) &= l(\bot) = l(X) = 1 \\
l([\,g, a\,]\varphi) &= l(\langle\, g, a\,\rangle\varphi) = 1 + l(\varphi) \\
l(\varphi_1 \diamond \varphi_2) &= l(\varphi_1) + l(\varphi_2), \quad \diamond \in \{\wedge, \vee\} \\
l(\mu X.\varphi) &= l(\nu X.\varphi) = 1 + l(\varphi)
\end{aligned}
$$

Definition 20 (Measure of a formula) For a formula $\varphi \in \mathrm{ERL}_\mu$ we define a measure:

$$m(\varphi) = (|X|, l(\varphi))$$

where $|X|$ denotes the cardinality of the set of the bounded variables in φ and $l(\varphi)$ is the length of the formula.

Definition 21 Let $\prec_\varphi \subset \mathrm{ERL}_\mu \times \mathrm{ERL}_\mu$ be a well-founded ordering relation on formulas such that:

$$\varphi \prec_\varphi \psi \text{ iff } m(\varphi) \;<\; m(\psi)$$

where $<$ denotes the lexicographic ordering relation on pairs.

4.2 Symbolic Model Checking

The *model checking problem* for ERL_μ is formulated as follows. Given an event-recording automaton A, a state $(l, v) \in Q_A$, and an ERL_μ formula φ, determine whether A satisfies φ, that is, $A, (l, v), \vartheta \models \varphi$, where ϑ is an assignment function for the free variables of φ. In the following we give for a formula φ a symbolic semantics in terms of a set of symbolic states $\mathcal{S}_A$, in which the formula φ holds.

Definition 22 (Symbolic Semantics) Let $\mathcal{M}$ be a model, and $\vartheta : V \to \mathcal{P}(\mathcal{S}_A)$ be an assignment function. The set of symbolic states (l, U) in which a formula φ is true, $[\![\varphi]\!]^{\mathcal{M}}_{\vartheta}$, is defined inductively as follows, where for a given edge $e = \langle l, l', a, g'\rangle$, $succ(U, e) = elapse(reset(S(a), U \mathbin{\bar\wedge} g')) \mathbin{\bar\wedge} I(l')$.

$$
\begin{aligned}
[\![\mathtt{tt}]\!]^{\mathcal{M}}_{\vartheta} &:= \mathcal{S}_A \\
[\![\mathtt{ff}]\!]^{\mathcal{M}}_{\vartheta} &:= \emptyset \\
[\![X]\!]^{\mathcal{M}}_{\vartheta} &:= \vartheta(X) \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{M}}_{\vartheta} &:= [\![\varphi_1]\!]^{\mathcal{M}}_{\vartheta} \cup [\![\varphi_2]\!]^{\mathcal{M}}_{\vartheta} \\
[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{M}}_{\vartheta} &:= [\![\varphi_1]\!]^{\mathcal{M}}_{\vartheta} \cap [\![\varphi_2]\!]^{\mathcal{M}}_{\vartheta} \\
[\![\langle\, g, a\,\rangle\varphi]\!]^{\mathcal{M}}_{\vartheta} &:= \{(l, U) \mid g \in U \text{ and } \exists\, e = \langle l, l', a, g'\rangle \in E, \exists\, U' = succ(U, e) \\
&\qquad \text{such that } (l', U') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![[\,g, a\,]\varphi]\!]^{\mathcal{M}}_{\vartheta} &:= \{(l, U) \mid g \in U \text{ and } \forall\, e = \langle l, l', a, g'\rangle \in E, \forall\, U' = succ(U, e) \\
&\qquad \text{implies } (l', U') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![\mu X.\varphi(X)]\!]^{\mathcal{M}}_{\vartheta} &:= \cap\{\mathcal{S}' \subseteq \mathcal{S}_A \mid [\![\varphi(X)]\!]^{\mathcal{M}}_{\vartheta[X:=\mathcal{S}']} \subseteq \mathcal{S}'\} \\
[\![\nu X.\varphi(X)]\!]^{\mathcal{M}}_{\vartheta} &:= \cup\{\mathcal{S}' \subseteq \mathcal{S}_A \mid \mathcal{S}' \subseteq [\![\varphi(X)]\!]^{\mathcal{M}}_{\vartheta[X:=\mathcal{S}']}\}
\end{aligned}
$$

We also write $\mathcal{M},(l,U),\vartheta \models_s \varphi$, for the symbolic satisfaction relation, to denote that $(l,U) \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}$.

Lemma 6 Let A be an event-recording automaton over the alphabet Σ, and let φ be a closed ERL_μ formula. Then $A,(l_0,v_0) \models \varphi$ iff $A,(l_0,U_0) \models_s \varphi$.
Proof. The proof is by measure induction over φ using the measure $m(\varphi)$ from Definition 20. As Induction Hypothesis (I.H.) we assume that $(\forall v \in [\![U]\!].\ A,(l,v) \models \varphi$ iff $A,(l,U) \models_s \varphi)$.

$$
\begin{array}{rcl}
(l,U) \models_s \mathtt{tt} & \iff & (l,v) \models \mathtt{tt} \\[2ex]
(l,U) \models_s \mathtt{ff} \equiv \mathit{false} & \iff & (l,v) \models \mathtt{ff} \equiv \mathit{false} \\[2ex]
(l,U) \models_s \varphi_1 \wedge \varphi_2 & \iff & \{\text{by Def. 22}\} \\
 & & (l,U) \models_s \varphi_1 \text{ and } (l,U) \models_s \varphi_2 \\
 & \iff & \{\text{by I.H. since } \varphi_1 \prec_\varphi \varphi \text{ and } \varphi_2 \prec_\varphi \varphi\} \\
 & & (l,v) \models \varphi_1 \text{ and } (l,v) \models \varphi_2 \\
 & \iff & \{\text{by Def. 18}\} \\
 & & (l,v) \models \varphi_1 \wedge \varphi_2 \\[2ex]
(l,U) \models_s \varphi_1 \vee \varphi_2 & \iff & \{\text{by Def. 22}\} \\
 & & (l,U) \models_s \varphi_1 \text{ or } (l,U) \models_s \varphi_2 \\
 & \iff & \{\text{by I.H. since } \varphi_1 \prec_\varphi \varphi \text{ and } \varphi_2 \prec_\varphi \varphi\} \\
 & & (l,v) \models \varphi_1 \text{ or } (l,v) \models \varphi_2 \\
 & \iff & \{\text{by Def. 18}\} \\
 & & (l,v) \models \varphi_1 \vee \varphi_2 \\[2ex]
(l,U) \models_s \langle g,a \rangle \varphi & \iff & \{\text{by Def. 22}\} \\
 & & g \in U \text{ and } \exists\, e = \langle l,l',a,g' \rangle \in E, \exists\, U' = \mathit{succ}(U,e) \\
 & & \text{such that } (l',U') \models_s \varphi \\
 & \iff & \{\text{by I.H. since } \varphi \prec_\varphi \langle g,a \rangle \varphi\} \\
 & & g \in U \text{ and } \exists\, e = \langle l,l',a,g' \rangle \in E, \exists\, U' = \mathit{succ}(U,e) \\
 & & \text{such that } (l',v') \models_s \varphi,\ \forall v' \in [\![U']\!] \\
 & \iff & \{\text{by Def. 9}\} \\
 & & \exists\, e = \langle l,l',a,g' \rangle \in E,\ \exists\, \delta \geq 0. \\
 & & (v \approx g \wedge g' \text{ and } v' = v[S(a) := 0] \text{ and } v' \approx I(l')) \\
 & & \text{and } (l',v') \models_s \varphi \\
 & \iff & (l,v) \models \langle g,a \rangle \varphi \quad \{\text{by Def. 18}\} \\[2ex]
(l,U) \models_s [g,a] \varphi & \iff & \{\text{by Def. 22}\} \\
 & & g \in U \text{ and } \forall\, e = \langle l,l',a,g' \rangle \in E, \forall\, U' = \mathit{succ}(U,e)
\end{array}
$$

$$
\begin{array}{rcl}
 & & \text{implies } (l',U') \models_s \varphi \\
 & \iff & \{\text{by I.H. since } \varphi \prec_\varphi \langle\, g,a\,\rangle\varphi\} \\
 & & g \in U \text{ and } \forall\, e = \langle l,l',a,g'\rangle \in E, \forall\, U' = \mathit{succ}(U,e) \\
 & & \text{implies } (l',v') \models_s \varphi,\ \forall v' \in [\![U']\!] \\
 & \iff & \{\text{by Def. 9}\} \\
 & & \forall\, e = \langle l,l',a,g'\rangle \in E,\ \forall\,\delta \geq 0. \\
 & & (v \mathrel{|\!\!\approx} g \wedge g' \text{ and } v' = v[S(a) := 0] \text{ and } v' \mathrel{|\!\!\approx} I(l')) \\
 & & \text{implies } (l',v') \models_s \varphi \\
 & \iff & \{\text{by Def. 18}\} \\
 & & (l,v) \models [\, g,a\,]\varphi \\
 & & \\
(l,U) \models_s \mu X.\varphi & \iff & \{\text{by Lemma 7}\} \\
 & & \exists i.\, (l,U) \models_s \varphi^i(\mathtt{ff}) \\
 & \iff & \{\text{by I.H., since } \varphi^i(\mathtt{ff}) \prec_\varphi \mu X.\varphi\} \\
 & & \exists i.\, (l,v) \models \varphi^i(\mathtt{ff}) \\
 & \iff & \{\text{by Lemma 5}\} \\
 & & (l,v) \models \mu X.\varphi \\
 & & \\
(l,U) \models_s \nu X.\varphi & \iff & \{\text{by Lemma 7}\} \\
 & & \exists i.\, (l,U) \models_s \varphi^i(\mathtt{tt}) \\
 & \iff & \{\text{by I.H., since } \varphi^i(\mathtt{tt}) \prec_\varphi \nu X.\varphi\} \\
 & & \exists i.\, (l,v) \models \varphi^i(\mathtt{tt}) \\
 & \iff & \{\text{by Lemma 5}\} \\
 & & (l,v) \models \nu X.\varphi
\end{array}
$$

□

Using the widening operator introduced in Section 3, we define a rounded symbolic semantics of ERL_μ, which, in contrast to the symbolic semantics, is finite.

Definition 23 (Rounded Symbolic Semantics) Let $\mathcal{M}$ be a model with a (finite) set of rounded symbolic states $\mathcal{S}_A^R$, and $\vartheta : V \to \mathcal{P}(\mathcal{S}_A^R)$ be an assignment function. The set of rounded symbolic states $(l,\mathcal{U}) \in \mathcal{S}_A^R$ in which a formula φ is true, $[\![\varphi]\!]_\vartheta^\mathcal{M}$, is defined inductively as follows, where for a given edge $e = \langle l,l',a,g'\rangle$, $\mathit{succ}(\mathcal{U},e) = \mathit{elapse}(\mathit{reset}(S(a),\mathcal{U} \mathbin{\bar\wedge} g')) \mathbin{\bar\wedge} I(l')$.

$$
\begin{array}{rcl}
[\![\mathtt{tt}]\!]_\vartheta^\mathcal{M} & := & \mathcal{S}_A^R \\
[\![\mathtt{ff}]\!]_\vartheta^\mathcal{M} & := & \emptyset \\
[\![X]\!]_\vartheta^\mathcal{M} & := & \vartheta(X) \\
[\![\varphi_1 \vee \varphi_2]\!]_\vartheta^\mathcal{M} & := & [\![\varphi_1]\!]_\vartheta^\mathcal{M} \cup [\![\varphi_2]\!]_\vartheta^\mathcal{M}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{M}}_{\vartheta} & := & [\![\varphi_1]\!]^{\mathcal{M}}_{\vartheta} \cap [\![\varphi_2]\!]^{\mathcal{M}}_{\vartheta} \\
[\![\langle\, g, a \,\rangle\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \{(l,\mathcal{U}) \mid g \in \mathcal{U} \text{ and } \exists\, e = \langle l, l', a, g' \rangle \in E, \exists \mathcal{U}' = \mathit{succ}(\mathcal{U}, e) \\
& & \text{ such that } (l',\mathcal{U}') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![[\, g, a \,]\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \{(l,\mathcal{U}) \mid g \in \mathcal{U} \text{ and } \forall\, e = \langle l, l', a, g' \rangle \in E, \forall \mathcal{U}' = \mathit{succ}(\mathcal{U}, e) \\
& & \text{ implies } (l',\mathcal{U}') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta}\} \\
[\![\mu X.\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \cap\{(l,\mathcal{U}) | \{(l',\mathcal{U}') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta[X:=(l,\mathcal{U})]}\} \subseteq (l,\mathcal{U})\} \\
[\![\nu X.\varphi]\!]^{\mathcal{M}}_{\vartheta} & := & \cup\{(l,\mathcal{U}) | \{(l,\mathcal{U}) \subseteq \{(l',\mathcal{U}') \in [\![\varphi]\!]^{\mathcal{M}}_{\vartheta[X:=(l,\mathcal{U})]}\}
\end{array}
$$

Since by Definition 15, $(l,\mathcal{U}) \overset{a}{\Longrightarrow} (l',\mathcal{U}')$ iff $(l,U) \overset{a}{\rightsquigarrow} (l',U')$ where $\mathcal{U} = \mathit{widening}_k(U)$, it follows that

$$A,(l_0,U_0) \models_s \varphi \text{ iff } A,(l_0,\mathcal{U}_0) \models_{rs} \varphi$$

where $\mathcal{U}_0 = \mathit{widening}_k(U_0)$ and $\models_{rs}$ denotes the rounded symbolic satisfaction relation. This is stated in the following Proposition.

Proposition 4 Let A be an event-recording automaton over the alphabet Σ, and let φ be a closed ERL_μ formula. Then $A,(l_0,U_0) \models_s \varphi$ iff $A,(l_0,\mathcal{U}_0) \models_{rs} \varphi$.

Lemma 7 For a model $\mathcal{M}$, a rounded symbolic state $(l,\mathcal{U})$, and a fixpoint formula $\mu X.\varphi \in \mathrm{ERL}_\mu$ ($\nu X.\varphi \in \mathrm{ERL}_\mu$) the following holds:

$$
\begin{array}{lll}
\mathcal{M},(l,\mathcal{U}) \models_{rs} \mu X.\varphi & \text{iff} & \exists i_0 \in \mathcal{O}.\mathcal{M},(l,\mathcal{U}) \models_{rs} \varphi^{i_0}(\mathtt{ff}) \\
\mathcal{M},(l,\mathcal{U}) \models_{rs} \nu X.\varphi & \text{iff} & \exists i_0 \in \mathcal{O}.\mathcal{M},(l,\mathcal{U}) \models_{rs} \varphi^{i_0}(\mathcal{S}_A)
\end{array}
$$

Proof. Follows immediately from Definition 23 and Lemma 4 and monotonicity. □

Thus, according to Lemma 6 and Proposition 4, the model checking problem $A,(l_0,v_0) \models \varphi$ can be reduced to the verification of $A,(l_0,\mathcal{U}_0) \models_{rs} \varphi$. The latter problem can be decided by calculating the set $[\![\varphi]\!]^A$ and testing if $(l_0,\mathcal{U}_0) \in [\![\varphi]\!]^A$. This set is computable and can be calculated by recursion over the structure of φ, using the above symbolic semantics, and by evaluating fixpoints outside-in using the Knaster-Tarski theorem. The termination argument is given by Proposition 2.

Theorem 4 The model checking problem for event-recording automata and ERL_μ, that is given an event-recording automaton A, a state $(l_0,v_0) \in Q_A$, and an ERL_μ formula φ, determine whether $A,(l_0,v_0) \models \varphi$, is decidable.
Proof. From Lemma 6 and Proposition 4 it follows that verifying $A,(l_0,v_0) \models \varphi$ is equivalent to verifying $A,(l_0,\mathcal{U}_0) \models_{rs} \varphi$, where $\mathcal{U}_0 = v_0$. This is equivalent to calculating the set $[\![\varphi]\!]^A$ and testing if $(l_0,\mathcal{U}_0) \in [\![\varphi]\!]^A$. This set of symbolic states is calculated by recursion over the structure of φ, using Definition 22, and by evaluating fixpoints outside-in using the Knaster-Tarski theorem. Each

step is computable and the fixpoint iterations terminate. The set $[\![\mu X.\varphi]\!]$ is the least fixpoint of the functional $F(X) = [\![\varphi]\!]$. This set can be defined as

$$[\![\mu X.\varphi]\!]^A = \lim_{i\to\infty} F^i(\mathtt{ff}).$$

This least fixpoint of $F(X)$ is computed as the limit of the successive approximation $\mathtt{ff}, F(\mathtt{ff}), F^2(\mathtt{ff}), \ldots$. All elements of this sequence are symbolic states $(l, \mathcal{U})$. Since there are only finitely many symbolic states, and F is monotone, the limit of the above approximation sequence is reached within a finite number of steps. Similarly, for the greatest fixpoint, the set $[\![\nu X.\varphi]\!]$ is defined by

$$[\![\nu X.\varphi]\!]^A = \lim_{i\to\infty} F^i(\mathtt{tt}).$$

This value can be computed by the successive approximation $\mathtt{tt}, F(\mathtt{tt}), F^2(\mathtt{tt}), \ldots$. The elements of this sequence are also symbolic states. □

Besides the decidability of model checking, ERL_μ has other nice properties. We were able to show that ERL_μ characterizes timed bisimulation and we also showed the decidability of the validity problem of the fixpoint free fragment of ERL_μ using a tableau construction. These results are subject of a forthcoming paper.

4.3 Related Work

Raskin and Schobbens [RS99] propose the class of recursive event-clock automata and a corresponding real-time logic, rEventClockTL, as an extension of the linear time temporal logic LTL with event-predicting ($\triangleright$) and event-recording ($\triangleleft$) operators. The formula $\triangleright_{\sim c}\varphi$ expresses that the delay before the next observation of φ satisfies constraint $\sim c$, and $\triangleleft_{\sim c}\varphi$ constrains the previous observation of φ, where $\sim \in \{\leq, \geq, <, >, =\}$ and c is an integer constant. The above event-recording operator is more general than our $S(\cdot)$ operator, since it can constrain any formula φ rather than only events. However, we can allow similar operators in the ERL_μ logic by using some auxiliary constructs without affecting the expressiveness of the logic and the decidability results. In [RS99] it is shown that the problems of satisfiability, validity and model checking are decidable for the rEventClockTL logic. These results are based on a Büchi-like construction for each formula of a suitable event-clock automaton, whose language is empty if the formula is not satisfiable. D'Souza [D'S00] gives a logical characterization of event-recording automata with a monadic second-order logic, MSO_{er}, interpreted over timed words. He introduces the operator $\Delta_a(x)$ to measure the time elapsed since the last occurrence of the a action with respect to the position x. D'Souza mainly concentrates on language-theoretical aspects rather than on model checking issues. Both logics are extensions of linear time temporal logics and do not include fixpoint operations. Further, the timing operators of both logics do not allow the comparison of two different clocks.

Other real-time extensions of the modal μ-calculus, such as T_μ [HNSY94] or $\mathcal{L}_\nu$ [LLW95] introduce explicit clocks in the formula. Therefore, to interpret a formula in a given model $\mathcal{M}$, two different sets of clocks, and thus two different clock valuation functions, must be regarded. For example, in $\mathcal{L}_\nu$, given a timed automaton A, formulas are interpreted over *extended states* of the form $\langle l, vu \rangle$ where l is a location of A, v a clock valuation for the set of clocks of A, and u a clock valuation for the formula clock set. The satisfiability problem for both T_μ and $\mathcal{L}_\nu$ logics is undecidable, since it is not possible to deduce the number of clocks of the automaton for the number of clocks in the formula. However, the bounded satisfiability problem for $\mathcal{L}_\nu$ is decidable, as shown in [LLW95], that is, given a formula and bounds k and M a timed automaton with no more than k clocks and no clock being compared with constants greater than M can be found. In our logic we do not make any distinction between formula and automata clocks, since clocks are related in both formalisms to events.

Chapter 5

Conclusion

We have presented symbolic model checking algorithms for event-recording automata. The algorithms are based on standard techniques from timed automata such as symbolic representation of the state space and reachability analysis. Both algorithms compute the set of all reachable states for a given network of event-recording automata. With the naïve method, the product automaton is first computed from the components of the network. Since this method is not very efficient, we presented a second method, which computes the reachable states on-the-fly, without computing the entire product automaton.

The reachability algorithms have been implemented in TEMPO, a tool for automatic verification of real-time systems parameterized over a data type for representing and manipulating constraint systems. For representing clock constraints we use a polyhedra and a difference decision diagrams library. Furthermore, TEMPO provides a platform for comparing different representations of clock constraints. We were not too surprised to find that DDDs easily outperform polyhedra packages. Experimental results show that the set of reachable states is computed faster using DDDs than using polyhedra, and that larger systems can be handled.

Furthermore, we have investigated a real-time fixpoint logic, which is closely related to the model of event-recording automata. A since operator is introduced to measure the time since the last occurrence of a certain event. Our logic includes least and greatest fixpoint operators, and can thus be viewed as a real-time extension of the modal μ-calculus. The model checking problem for our real-time logic is easily shown to be decidable.

Currently, TEMPO only includes reachability algorithms, but we are in the process of extending TEMPO with the model checking algorithm for ERL_μ. Furthermore, we plan to extend TEMPO and ERL_μ to event-predicting clocks. In future work, we would also like to study the interplay of model checking with theorem proving by connecting TEMPO with either SAL [BGL$^+$00] or PVS [ORS92]. For example, we may use a variant of predicate abstraction as studied by Shankar and Saïdi [SS99], for computing a finite state event-clock system from one with infinite data. Then, TEMPO could be used to compute the reachable state set of the abstract system, which yields, after concretization an overapproximation of the reachable state set of the concrete system. We believe that the model and the logic described in this paper form a suitable basis for such an integration.

Acknowledgment

We thank N. Shankar and Bruno Dutertre from SRI for useful comments and Jesper Møller from IT University in Copenhagen for providing the DDD package. We also thank the anonymous referees for their helpful comments.

Bibliography

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1–2):253–273, January 1999. A preliminary version appeared in the Proceedings of the Sixth International Conference on Computer-aided Verification (CAV 1994), LNCS 818, Springer-Verlag, 1994, pp. 1-13.

[BGL+00] Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Muñoz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saïdi, N. Shankar, Eli Singerman, and Ashish Tiwari. An overview of SAL. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, June 2000. NASA Langley Research Center.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. *Lecture Notes in Computer Science*, 1066:208–219, 1996.

[D'S00] Deepak D'Souza. A logical characterization of event recording automata. In *Proceedings of FTRTFT*, volume 1926 of *LNCS*, 2000.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.

[HMP92] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. *Lecture Notes in Computer Science*, 600:226–251, 1992.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, June 1994.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, February 1987.

[LLW95] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic — and back. *Lecture Notes in Computer Science*, 969:529–539, 1995.

[LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, Fall 1999.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. *Lecture Notes in Computer Science*, 965:62–88, 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[MLAH99a] J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. In *Computer Science Logic*, The IT University of Copenhagen, Denmark, September 1999.

[MLAH99b] J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *FLoC'99 Workshop on Symbolic Model Checking*, July 1999.

[ORS92] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[Pet99] Paul Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice.* PhD thesis, Uppsala University, 1999.

[RS99] Jean-François Raskin and Pierre-Yves Schobbens. The logic of event clocks: Decidability, complexity and expressiveness. *Journal of Automata, Languages and Combinatorics*, 4(3):247–282, 1999.

[SBM92] F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. *Lecture Notes in Computer Science*, 600:618–639, 1992.

[Sha93] N. Shankar. Verification of Real-Time Systems Using PVS. In Costas Courcoubetis, editor, *Computer-Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 280–291, Elounda, Greece, June/July 1993. Springer-Verlag.

[SS99] Hassen Saïdi and Natarajan Shankar. Abstract and model check while you prove. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification (CAV'99)*, number 1633 in LNCS, pages 443–454, Trento, Italy, July 1999. Springer-Verlag.

[Sti87] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 49(2-3):311–347, 1987.

[Sti96] C. Stirling. Modal and temporal logics for processes. *Lecture Notes in Computer Science*, 1043:149–237, 1996.

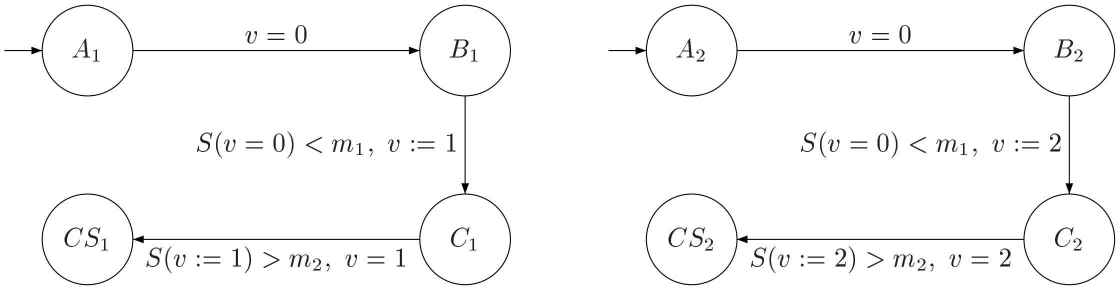
[Wil93] Doran K. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, France, 1993. http://www.irisa.fr/bibli/publi/pi/1993/785/785.html.

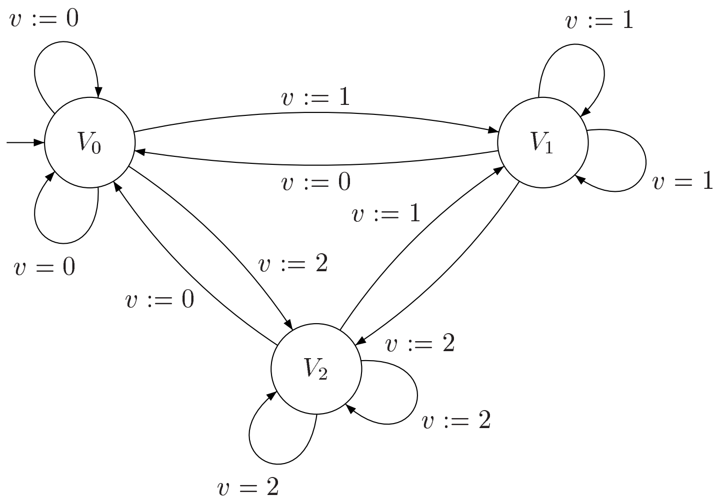
[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, November 1994.

Appendix A

Fischer's protocol with two processes

Given two processes P_1 and P_2. Each process is modeled as an event-recording automaton, over the alphabet $\Sigma_1 = \{v = 0, v := 1, v = 1\}$, $\Sigma_2 = \{v = 0, v := 2, v = 2\}$ respectively. The shared variable V is also modeled as an event-recording automaton over the alphabet $\Sigma_V = \{v := 0, v = 0, v := 1, v = 1, v := 2, v = 2\}$. The automata are illustrated above. The system is modeled as the product $(P_1 \|_{S_1} P_2 \|_{S_2} V)$, where $S_1 \stackrel{\mathrm{def}}{=} \{v = 0\}$ and $S_2 \stackrel{\mathrm{def}}{=} \{v = 0, v = 1, v := 1, v = 2, v := 2\}$.


A_1
$v = 0$
B_1
$S(v = 0) < m_1,\ v := 1$
CS_1
C_1
$S(v := 1) > m_2,\ v = 1$
A_2
$v = 0$
B_2
$S(v = 0) < m_1,\ v := 2$
CS_2
C_2
$S(v := 2) > m_2,\ v = 2$


$v := 0$
$v := 1$
$v := 1$
V_0
V_1
$v = 1$
$v := 0$
$v := 1$
$v = 0$
$v := 2$
$v := 0$
$v := 2$
V_2
$v := 2$
$v = 2$

Appendix B

Input file for the Fischer's mutual exclusion protocol with two processes

```
% event-recording automaton modelling the first process
  era q {
    start q0
    invariants
      q1 |-> true
    trans
      q0 -> q1 { a0 },                % event v=0
      q1 -> q2 { a4, S(a0) <= 1 },    % event v:=2
      q2 -> q3 { a5, S(a4) >= 2 }     % event v=2
  }

  || {a0}      % Synchronisation set {v=0}

% event-recording automaton modelling the second process
  era p {
    start p0
    invariants
      p1 |-> true
    trans
      p0 -> p1 { a0 },                % event v=0
      p1 -> p2 { a2, S(a0) <= 1 },    % event v:=1
      p2 -> p3 { a3, S(a2) >= 2 }     % event v=1
  }

  || {a0,a2,a3,a4,a5}  % Synchronisation set {v=0, v:=1, v=1, v:=2, v=2}
```

```
% event-recording automaton modelling the shared variable
  era v {
    start v0
    invariants
      v1 |-> true
    trans
      v0 -> v0 { a0 },   % event v=0
      v0 -> v0 { a1 },   % event v:=0
      v0 -> v1 { a2 },   % event v:=1
      v0 -> v2 { a4 },   % event v:=2
      v1 -> v0 { a1 },   % event v:=0
      v1 -> v1 { a2 },   % event v:=1
      v1 -> v1 { a3 },   % event v=1
      v1 -> v2 { a4 },   % event v:=2
      v2 -> v0 { a1 },   % event v:=0
      v2 -> v1 { a2 },   % event v:=1
      v2 -> v2 { a4 },   % event v:=2
      v2 -> v2 { a5 }    % event v=2
  }
```

Appendix C

Input file for the railroad crossing system

```
% event-recording automaton modelling the TRAIN
  era train {
    start t0
    invariants
      t1 |-> S(a) <= 5,
      t2 |-> S(a) <= 5,
      t3 |-> S(a) <= 5
    trans
      t0 -> t1 {a},              % event approach
      t1 -> t2 {i, S(a) >= 3 },  % event in
      t2 -> t3 {o},              % event out
      t3 -> t0 {e}               % event exit
  }

  ||{a,e}  % train synchronizes with controller on the events
           % approach and exit

% event-recording automaton modelling the CONTROLLER
  era controller {
    start c0
    invariants
      c1 |-> S(a) <= 1,
      c3 |-> S(e) <= 1
    trans
      c0 -> c1 {a},             % event approach
      c1 -> c2 {l, S(a) = 1 },  % event lower
      c2 -> c3 {e},             % event exit
      c3 -> c0 {r}              % event raise
```

```
  }

  ||{l,r}   % gate synchronizes with controller on the events
            % lower and raise

% event-recording automaton modelling the GATE
  era gate {
    start g0
    invariants
      g1 |-> S(l) <= 1,
      g3 |-> S(r) <= 2
    trans
      g0 -> g1 {l},             % event lower
      g1 -> g2 {d},             % event down
      g2 -> g3 {r},             % event raise
      g3 -> g0 {u, S(r) >= 1}  % event up
  }
```