# Widen, Narrow and Relax

Giorgio Delzanno and Andreas Podelski
Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
{delzanno,podelski}@mpi-sb.mpg.de

**Abstract**

We apply results from linear programming to show that the relaxation of model checking over integers to reals is accurate, i.e. yields a full test of temporal properties, for a large class of concurrent systems. We define abstractions similar to widening and narrowing that accelerate least and greatest fixpoint computations in model checking over integers or reals. We show that these abstractions are accurate in the same sense. Preliminary experimental results (e.g. safety for the ticket algorithm, liveness of a parameterized elevator program) indicate the potential usefulness of our abstraction techniques.

## 1 Introduction

The verification problem for concurrent systems with (unbounded) integer values is receiving increasing attention; see e.g. [BW94, BW98, Bul98, BGP97, BGP98, Čer94, CJ98, FR96, SKR98]. The problem is undecidable for most classes of practical importance. So what can you do? There are basically two answers. (1) Give a possibly non-terminating algorithm that terminates for useful examples. This is the approach followed e.g. by [BW98, BGW+97, KMM+97].[1] (2) Give a semi-test that yields the definite answer for useful examples (the other answer being 'don't know'); see e.g. [BGP97, CGL92, LGS+94, Gra94, Dam96, Hal93, HPR97, HH95].

One obtains a semi-test by introducing abstractions that yield a conservative approximation of the original property. In most successful experiments, the abstractions (essentially to finite-state systems) were more or less chosen manually, application-specific. The application of automated, application-independent abstractions that enforce termination (as common in program analysis) to model checking seems difficult, for the reason that such abstractions are often too rough, i.e. abstract away details that are essential to verify the property in question.

In this paper, we consider two kinds of automated, application-independent abstractions that do not enforce termination; instead, their approximation is accurate, i.e. does not loose information wrt. the original property. This way, we carry over the practical advantage of the second approach, namely the acceleration of the model-checking fixpoint computation, to the first approach while still implementing a full test, i.e. maintaining the definiteness of all answers.

To know the accuracy of an abstraction is important conceptually and pragmatically. Note that there seems to be no other way to predict its effect ("too rough?") for a particular application. Obviously, the accuracy is useful for debugging (or finding typos); 'don't know' answers are quite frustating. Finally, it allows one to determine the 'correct' parameters in initial-state specifications.

Technically, our contributions are as follows.

Applying classical results from linear programming [DT97, GN72], we show that the symbolic model checking procedure over reals obtained by relaxation from the one over integers yields a full test of temporal properties for a specific class of concurrent systems; this class seems natural by its definition and contains many examples considered in the literature. The purpose of this first abstraction is to accelerate each single fixpoint iteration. The number of iterations does not decrease. In order to show that it cannot increase, we prove that the relaxation of the fixpoint test is accurate as well.

[1] In this context, see Wolper's statement about the 'practical' absense of a termination guarantee for every model checker on any but the most trivial instances [BW98].

Applying history-dependent widening and narrowing techniques as already foreseen in the abstract interpretation scheme [CC77] and basing our intuition on techniques from Constraint Logic Programming (see [DP98]) and Constraint Data Bases [KKR95, Rev93], we show that a set of abstractions of the model-checking fixpoint operator yields an accurate model checking algorithm (i.e. a full test if terminating). These abstractions are able to drastically decrease the number of iterations or even to enforce termination of an otherwise non-terminating test.

We report on two of our experiments with our CLP(R) implementation of the 'real' relaxation of the symbolic model checking procedure with widening and narrowing. Our examples are typical integer-valued concurrent systems taken from the literature: the ticket algorithm [BGP97] and an elevator program [BW94]. We were able to show safety for the ticket algorithm and liveness for the parameterized elevator program. As far as we know, this is the first time that these two verification problems are solved with a 'full-test' model checker.

## 2 Constraints, Concurrent Systems, Symbolic Model Checking

In this section, we give the preliminaries and fix the formal setup of this paper. We refer to the logical formulas in a given class as 'constraints' if we are interested mainly in the relation that they denote. Our constraints are closed under conjunction but not under disjunction. We use lower case greek letters for a constraint and upper case ones for a set of constraints (which stands for their disjunction). We write $\vec{x}$ for the tuple of variables $\langle x_1, \ldots, x_n \rangle$ and $\vec{d}$ for the tuple of data values $\langle d_1, \ldots, d_n \rangle$ from the interpretation domain $\mathcal{D}$. As usual, $D, \alpha \models \phi$ is the validity of the formula $\phi$ under the valuation $\alpha$, and $\alpha[\vec{x} \mapsto \vec{d}]$ stands for a valuation that maps $x_i$ to $d_i$ for $i = 1, \ldots, n$. We can now formally define the relation denoted by the constraint $\phi$:

$$\left[\, \phi \,\right]_{\mathcal{D}} = \{\vec{d} \mid \mathcal{D}, \alpha[\vec{x} \mapsto \vec{d}] \models \phi\}.$$

(This set is sometimes confused with the set of *solutions* of $\phi$.) Note that we use $\left[\, \phi \,\right]_{\mathcal{D}}$ always with respect to $\vec{x}$; i.e., $x_1, \ldots, x_n$ act as the free variables of $\phi$, and implicitly all other variables are existentially quantified. We write $\phi[\vec{y}]$ for the constraint obtained by alpha-renaming from $\phi$. We define $\left[\, \Phi \,\right]_{\mathcal{D}}$, the relation denoted by the set of constraints $\Phi$ with respect to the variables $x_1, \ldots, x_n$, in the canonical way.

We need the following operations on sets of constraints, besides the two operations $\vee$ and $\wedge$ that implement disjunction and conjunction (both operations return sets of constraints).

- (Satisfiability test) $\mathrm{Sat}_{\mathcal{D}}(\Phi)$ returns 'true' if and only if $\left[\, \Phi \,\right]_{\mathcal{D}} \neq \emptyset$.
- (Variable elimination) $\mathrm{Elim}_{\mathcal{D}}(\vec{y}, \phi)$ returns a constraint $\gamma$ that is equivalent to $\exists \vec{y} \phi$ and whose variables are contained in those of $\phi$ without $y_1, \ldots, y_n$.
- (Entailment test) $\mathrm{Entail}_{\mathcal{D}}(\Psi, \Phi)$ returns 'true' if and only if $\left[\, \Psi \,\right]_{\mathcal{D}} \subseteq \left[\, \Phi \,\right]_{\mathcal{D}}$.

Following [Sha93], we use *concurrent systems* (to which concurrent programs can be directly translated) to specify systems consisting of concurrently executing processes. A concurrent system $\mathcal{S} = \langle \vec{x}, \Theta, \mathcal{E} \rangle$ is given by its control and data variables $x_1, \ldots, x_n$, a initial condition $\Theta$, and a set $\mathcal{E}$ of pairs $\langle \psi, \varphi \rangle$ (the "events"), written also **cond** $\psi$ **action** $\varphi$, where the *guard* $\psi$ is a constraint over $x_1, \ldots, x_n$ and the *action* $\varphi$ is a constraint over the variables $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$. The primed variable $x'$ stands for the value of $x$ in the successor state. The interleaving semantics of $\mathcal{S}$ is defined by a transition system whose states are vectors $\vec{d} = \langle d_1, \ldots, d_n \rangle$ of values for the variables $x_1, \ldots, x_n$. The predecessor function *pre* of $\mathcal{S}$ applied to the set of states $S$ yields the set of all states with at least one successor in $S$. We have that

$$pre(S) = \{\vec{d} \mid \ \mathcal{D}, \alpha[\vec{x} \mapsto \vec{d}, \vec{x'} \mapsto \vec{d'}] \models \psi \wedge \varphi, \ \ \vec{d'} \in S, \ \ \langle \psi, \varphi \rangle \in \mathcal{E}\}.$$

**We will use a set of constraints $\Phi$ to represent a set of states $S$ if $S = \left[\, \Phi \,\right]_{\mathcal{D}}$.** The predecessor states of such a set are represented by the set of the constraints obtained by conjoining the guard $\varphi$ and

the action $\psi$ of each event with each renamed constraint $\phi[\vec{x'}]$ of $\Phi$:[2]

$$pre(\lceil \Phi \rceil_{\mathcal{D}}) = \Big[ \{\psi \wedge \varphi \wedge \phi[\vec{x'}] \mid \langle \psi, \varphi \rangle \in \mathcal{E},\ \ \phi[\vec{x'}] \in \Phi\} \Big]_{\mathcal{D}}.$$

We will next define *symbolic model checking* in terms of the before-mentioned operations on constraints. We define the predecessor operator $pre_{\mathcal{D}}$ over sets of constraints by

$$pre_{\mathcal{D}}(\Phi) = \{\mathrm{Elim}_{\mathcal{D}}(\vec{x'},\ \psi \wedge \varphi \wedge \phi[\vec{x'}]) \mid\ \langle \psi, \varphi \rangle \in \mathcal{E},\ \ \phi \in \Phi,\ \ \mathrm{Sat}_{\mathcal{D}}(\psi \wedge \varphi \wedge \phi[\vec{x'}]) = \mathrm{true}\}.$$

and obtain that it implements the predecessor function $pre$ in the sense that

$$\Big[ pre_{\mathcal{D}}(\Phi) \Big]_{\mathcal{D}} = pre\Big( smallinst\Phi\mathcal{D} \Big).$$

Following the style of [KMM+97], we next formulate (possibly non-terminating) symbolic model checking algorithms for safety and liveness properties. Given a set of constraints $\Phi$ (representing a set of states), the algorithms SYMB-EF and SYMB-EG below 'compute' (if terminating) a set of constraints that represents the sets of states satisfying the properties $EF(\Phi)$ and $EG(\Phi)$, respectively.

**Proc** SYMB-EF($\Phi$)
  $\Phi_0 := \Phi$;
  **repeat**
    $\Phi_{i+1} = \Phi_i \ \vee\ pre_{\mathcal{D}}(\Phi_i)$;
  **until** $\mathrm{Entail}_{\mathcal{D}}(\Phi_{i+1}, \Phi_i) = \mathrm{true}$;
  **return** $\Phi_i$;
**end**

**Proc** SYMB-EG($\Phi$)
  $\Phi_0 := \{true\}$;
  **repeat**
    $\Phi_{i+1} = \Phi \wedge pre_{\mathcal{D}}(\Phi_i)$;
  **until** $\mathrm{Entail}_{\mathcal{D}}(\Phi_i, \Phi_{i+1}) = \mathrm{true}$;
  **return** $\Phi_i$;
**end**

For testing the safety property $AG(\neg\Phi)$, we add the instruction (recall that $\Theta$ is the initial condition):

**check** $\Theta \wedge$ SYMB-EF($\Phi$) unsatisfiable,

and for the liveness (reactiveness) property $AG(\Phi \rightarrow AF(\neg\Psi))$:

**check** $\Theta \wedge$ SYMB-EF($\Phi \wedge$ SYMB-EG($\Psi$)) unsatisfiable.

# 3 Relaxation

In this section, we investigate the int-real relaxation of the symbolic model checking procedures SYMB-EF and SYMB-EG for a large class of concurrent systems with unbounded positive integer values (which we call 'simple' for the lack of a better name).[3]

The relaxation from integers to reals stems from linear programming [DT97, GN72]. The motivation there is the same as here: the manipulation of linear arithmetic constraints is less costly over reals than over integers, theoretically (e.g. polynomial vs. NP-hard for the satisfiability test) as well as practically (e.g., the variable elimination is less involved); see [DT97]. Even if the complexity for integers is the same as for reals for a particular application (it can hardly be better), there seems to be no reason to build a solver over integers (other than a potential loss of precision). There exist many highly optimized constraint systems over reals, general-purpose such as CLP(R) (see [DP98]) and special purpose such as Uppaal [BLL+98] or Hytech [HHW+97], which one would like to exploit for model checking concurrent systems over integers.

[2] Note that $[\cdot]_{\mathcal{D}}$ is defined wrt. the variables $x_1, \ldots, x_n$. I.e., the primed variables in the conjunction $\psi \wedge \varphi \wedge \phi[\vec{x'}]$ are implicitly existentially quantified. Recall that $\phi[\vec{x'}]$ is a renaming of $\phi$.

[3] Note that this abstraction is *not* an embedding of the verification problem for a system over integers into one for a system over reals.

*Simple* concurrent systems model programs that contain comparisons between variables, assignments between variables, increments and decrements. Vector Addition Systems (a.k.a. Petri Nets) and Integral Relational Automata are two well-known subclasses of simple concurrent systems (see e.g. [KM69] and [Čer94]). The reachability problem is decidable for these classes (see e.g [Čer94, Lam92]). 2-counter machines [Min67] can be directly encoded by simple concurrent systems. Other examples of simple concurrent systems are multi-clock automata [CJ98] and gap-order automata [FR96]. The above-mentioned decidability results are related to the general results for verification problems of infinite-state systems in [AJK+96, FS98]. The communication protocols considered in [BGP97, BGP98, SKR98] are examples of simple concurrent systems that do not seem to belong to a known decidable subclass.

**Definition 3.1 (Simple concurrent systems)** In a *simple* concurrent system, the data variables range over positive integers and the initial condition and all guards and actions are formulas $\phi$ (called *simple constraints*) built up according to the following grammar (where $c$ is a constant, and $x$ and $y$ are (primed or unprimed) variables).

$$\phi \ ::= \ x \leq y + c \mid c \leq x \mid x \leq c \mid \text{true} \mid \text{false} \mid \phi_1 \wedge \phi_2.$$

Note that the assignment action $x' = y + c$ is expressed by the simple constraint $x' \leq y + c \wedge y \leq x' - c$. Over the domain of integers, the constraint $x < y$ is equivalent to the simple constraint $x \leq y - 1$.

We will interpret simple constraints over the positive subset of both, the domains $\mathbf{Z}$ and $\mathbf{R}$ of integers and reals, respectively. We will next compare the two interpretations wrt. the operators used in SYMB-EF and SYMB-EG.

**Proposition 3.1 (Relaxation of constraint operators)**

i) The relaxation of the tests of satisfiability and entailment[4] and of the variable elimination is accurate; i.e., the operators $\text{Sat}_{\mathcal{D}}(\cdot)$, $\text{Entail}_{\mathcal{D}}(\cdot, \cdot)$ and $\text{Elim}_{\mathcal{D}}(\vec{y}, \cdot)$ (over simple constraints or over sets of simple constraints) yields the same results for $\mathcal{D} = \mathbf{Z}$ and for $\mathcal{D} = \mathbf{R}$.

ii) The application of the symbolic predecessor operator over integers $pre_{\mathbf{Z}}$ and of its real relaxation $pre_{\mathbf{R}}$ to a set of simple constraints $\Phi$ yield two sets of simple constraints with the same integers solutions, i.e. denoting the same relation over integers. That relation is obtained also by applying the predecessor function $pre$ to the relation denoted by $\Phi$. Formally,

$$\Big[pre_{\mathbf{R}}(\Phi)\Big]_{\mathbf{Z}} \ = \ \Big[pre_{\mathbf{Z}}(\Phi)\Big]_{\mathbf{Z}} \ = \ pre(\big[\,\Phi\,\big]_{\mathbf{Z}})$$

The proof is based on the classical results from linear programming and the fact that simple constraints are closed under the application of symbolic predecessor operators; see Appendix A. The iteration of (ii) yields that for all $k \geq 0$, $\Big[pre^k_{\mathbf{R}}(\Phi)\Big]_{\mathbf{Z}} = \Big[pre^k_{\mathbf{Z}}(\Phi)\Big]_{\mathbf{Z}}$.

This means that the relaxations of the procedures SYMB-EF and SYMB-EG from integers to reals 'compute' (if terminating) the same set of states of concurrent systems over integers. Moreover, since the satisfiability test is invariant under the relaxation, we obtain the following result.

**Theorem 3.1 (Relaxation)** The relaxation of the symbolic model checking procedures for safety and liveness properties of simple concurrent systems is accurate.

[4] In our implementation of the symbolic model checking procedures, we use the *local* entailment test that succeeds if every $\psi \in \Psi$ entails some $\phi \in \Phi$ (we thus trade efficiency with incompleteness for the fixpoint test); the accuracy of the relaxation of the local entailment test holds as well.

# 4 Widening

In this section, we consider how one can achieve (or just speed up) the termination of the least fixpoint iteration needed in SYMB-EF, the symbolic model checking algorithm for safety properties. We will first give some intuition through an artificial example, then define the algorithm SYMB-EF-W with the acceleration by widening formally and then show how it applies to the ticket algorithm. Consider the concurrent system with the event **cond** true **action** $x' = x \wedge y' = y + 1$ and the property $EF(x \leq y)$. The procedure SYMB-EF generates an infinite sequence of strictly increasing sets of constraints,

$$\Phi_0 = \{x \leq y\}, \quad \Phi_1 = \Phi_0 \cup \{x \leq y + 1\}, \quad \Phi_2 = \Phi_1 \cup \{x \leq y + 2\}, \quad \ldots$$

whose infinite union is equivalent to the constraint 'true'. Our widening operator will add the constraint 'true' (instead of $\{x \leq y + 1\}$) in the second iteration, after having gone through the following four steps: (1) compare the constraints $\gamma \equiv x \leq y$ and $\phi \equiv x \leq y + 1$ and observe that this might be the start of an infinite sequence, (2) check that '$\phi$ depends on $\gamma$', i.e. the two constraints are related by an event $e$ wrt. backward analysis, (3) check that the event $e$ is of a given form for which the sequence of constraints generated by iteration is known, and for which a constraint $\eta$ equivalent (or, stronger than) the disjunction of all those constraints can be inferred, (4) add the constraint $\eta$ to the set of constraints obtained in the second iteration step.

In general, the event $e$ that relates the constraint $\gamma$ with the constraint $\phi$ occurring in some later iteration is not an original event of the concurrent system but is constructed as a composition of events; we then write $e = event(\gamma, \phi)$.

We will next specify the predicate '$\phi$ depends on $\gamma$' and the construction of $event(\gamma, \phi)$. Let us first define the restriction $pre_{\mathcal{D}|e}$ of $pre_{\mathcal{D}}$ wrt. (a concurrent system consisting of) the single event $e$. Thus, if $e = \langle \psi, \varphi \rangle$, then $pre_{\mathcal{D}|e}(\phi) = \mathrm{Elim}_{\mathcal{D}}(\vec{x'}, \psi \wedge \varphi \wedge \phi[\vec{x'}])$.

We define that '$\phi$ depends on $\gamma$' holds if there exists a sequence of events $e_1, \ldots, e_m$ such that

$$\phi = pre_{\mathcal{D}|e_m}(\ldots pre_{\mathcal{D}|e_1}(\gamma) \ldots).$$

Let $e_1 = \langle \psi_1, \varphi_1 \rangle$. Then, we define the event $event(\gamma, \phi)$ as the event $\langle \mathrm{true}, \varphi \rangle$ such that[5]

$$\varphi = pre_{\mathcal{D}|e_m}(pre_{\mathcal{D}|e_{m-1}} \ \ldots \ (pre_{\mathcal{D}|e_2}(\psi_1 \wedge \varphi_1)) \ldots)).$$

We can now define SYMB-EF-W, the symbolic model checking algorithm with widening; see Figure 1. In the definition of WIDEN function, a third disjunct can be added to the **if** expression. In that disjunct, $\gamma$ decomposes into $\gamma \equiv \eta \wedge x \geq c$ and the event $e$ contains a conjunct $x' = x - c_x$ (i.e. a decrement instead of an increment). If the condition in the WIDEN function applies to several decompositions of $\gamma$ simultaneously, the corresponding widenings are effectuated in several successive iterations.

Before we analyze the accelerating effect of the widening for the verification of the ticket algorithm, we consider its accuracy.

**Theorem 4.1 (Widening)** The algorithm SYMB-EF-W obtained by abstracting the least fixpoint operator in the symbolic model checking algorithm SYMB-EF with the widening defined in Figure 1 yields (if terminating) a full test of safety properties for concurrent systems over integers or reals.

The proof of the theorem works by showing that $\left[\,\mathrm{WIDEN}(\gamma, \phi)\,\right]_{\mathcal{D}} \subseteq \left[\bigcup_{j \geq 0} \Phi_j\right]_{\mathcal{D}}$ holds for constraints $\gamma \in \Phi_i$ and $\phi \in pre_{\mathcal{D}}(\Phi_i)$, where $\Phi_i$ is the $i$-th set of constraints computed by the algorithm SYMB-EF; see Appendix B.

[5] We here assume suitable $\alpha$-renamings of all variables that are implicitly existentially quantied; i.e., when $pre_{\mathcal{D}}$ is applied to $\phi[\vec{x}, \vec{x'}]$ we consider a renaming $\vec{x''}$ of the variables $\vec{x'}$ etc.. Note that we can construct an event with an empty guard since the events $\langle \psi, \varphi \rangle$ and $\langle \mathrm{true}, \psi \wedge \varphi \rangle$ are equivalent wrt. model checking

**Procedure** SYMB-EF-W($\Phi$)
  $\Phi_0 := \Phi$;
  **repeat**
    $\Phi_{i+1} = \Phi_i \cup$ WIDEN$(\Phi_i, pre_{\mathcal{D}}(\Phi_i))$;
  **until** Entail$_{\mathcal{D}}(\Phi_{i+1}, \Phi_i) = \text{true}$;
**end**

**Function** WIDEN$(\Gamma, \Phi) = \{\text{WIDEN}(\gamma, \phi) \mid \gamma \in \Gamma,\ \phi \in \Phi\}$

**Function** WIDEN$(\gamma, \phi)$

**if** $\gamma \equiv \eta \wedge x \leq y + c$; $\gamma$ entails $\phi$; $\phi$ depends on $\gamma$ with $e = event(\gamma, \phi)$; $e$ contains $x' = x + c_x,\ y' = y + c_y$ where $c_y - c_x > 0$; $\eta$ entails $pre_{\mathcal{D}|e}(\eta)$

**or if** $\gamma \equiv \eta \wedge x \leq c$; $\gamma$ entails $\phi$; $\phi$ depends on $\gamma$ with $e = event(\gamma, \phi)$; $e$ contains $x' = x + c_x$ where $c_x > 0$; $\eta$ entails $pre_{\mathcal{D}|e}(\eta)$

**then return** $\eta$;
**else return** $\phi$;
**end**

Figure 1: Symbolic model checking procedure with widening.

In the *ticket algorithm* [BGP97, DP98] (see Figure D in Appendix), the variables are $p_1, p_2, a, b, t, s$. Here, $p_1$ and $p_2$ are the control variables of two concurrent processes trying to access a shared resource (ranging over the constants *think*, *wait* and *use*); the other variables range over unbounded positive integers. The incoming processes are assigned increasing numbers ("tickets") that are generated through the variable $t$ and stored internally by the two processes with the variables $a$ and $b$, respectively; the variable $s$, which stores the number of the ticket to be served next, is updated each time a process leaves the critical section.

This concurrent system does not belong to any of the known classes for which reachability is decidable. The unsafe states ("both processes are in the critical section") are represented by the simple constraint $p_1 = use \wedge p_2 = use$. The procedure SYMB-EF here yields an infinite sequence of constraints; see Figure D.1 in the appendix. We take two constraints in this sequence:

$$\gamma \equiv p_1 = use \wedge p_2 = wait \wedge \mathbf{b} \leq \mathbf{s} + \mathbf{1} \wedge t \leq s + 1,$$
$$\phi \equiv p_1 = use \wedge p_2 = wait \wedge \mathbf{b} \leq \mathbf{s} + \mathbf{2} \wedge t \leq s + 1.$$

Clearly, $\gamma$ entails $\phi$. Furthermore, $\phi$ 'depends on' $\gamma$ (it is produced by the application of the sequence of events called $e_2, e_1, e_3$ in Figure D.1) via the constructed event $e$ that we can write as:

$$\textbf{cond}\ \ p_1 = use \wedge t \leq s + 1\ \ \textbf{action}\ \ a' = t \wedge t' = t + 1 \wedge s' = s + 1.$$

We decompose $\gamma$ into $\gamma \equiv \eta \wedge b \leq s + 1$, where

$$\eta \equiv p_1 = use \wedge p_2 = wait \wedge t \leq s + 1.$$

Then, we note that $\eta$ entails $pre_{\mathcal{D}|e}(\eta)$, since

$$pre_{\mathcal{D}|e}(\eta) \equiv \exists p_1', p_2', a', b', t', s' \left( \begin{array}{l} p_1 = use \wedge p_2 = wait \wedge p_1' = use \wedge\ p_2' = wait\ \wedge\ t \leq s + 1 \wedge \\ a' = t\ \wedge\ b' = b \wedge t' = t + 1\ \wedge\ s' = s + 1 \wedge t' \leq s' + 1 \end{array} \right)$$

As a consequence, we can add the constraint $\eta$ to the current set of constraints (in the appendix, see Figure D.2). Adding this constraint *blocks* the addition of new constraints containing $p_1 = use, p_2 = wait$ (because they will all be subsumed).

The widening procedure has the same kind of effect on the remaining 'sources of non-termination' (namely, the constraints containing $p_1 = wait, p_2 = wait$, or $p_1 = think, p_2 = wait$, or $p_1 = wait, p_2 = think$). Thus, the algorithm SYMB-EF-W terminates for this problem. In its result, the only constraint with $p_1 = think \wedge p_2 = think$ is such that $t \leq s - 1$, which is not satisfiable in conjunction with the initial condition $\Theta$ (see Figure D). Since we have a full test, the result determines the 'correct' parameters in the initial condition $\Theta$. E.g., the initial values for the variables $a$, $b$ and $t$ can be left unconstrained if and only if we set $s = 0$.

# 5 Narrowing

We will next consider the acceleration of the greatest fixpoint iteration for SYMB-EG, the algorithm for liveness properties. Again, we will first consider a trivial example. This example, a dual to the introductory example in the previous section, will not be suitable, however, to explain the extra complication of narrowing wrt. widening for symbolic model checking based on constraints. Intuitively, the duality between narrowing and widening stops at the fact that we represent states as disjunctions of constraints (i.e. conjunctions of atomic formulas) and not as conjunctions of disjunctions; the intersection operator between sets of constraints is more involved than the union operator; the function NARROW on sets of constraints cannot be defined as a canonical extension of a function on constraints, in contrast to the function WIDEN. Take the concurrent system with the event **cond** true **action** $x' = x \wedge y' = y - 1$ and the property $EG(x \leq y)$. The procedure SYMB-EG generates a strictly decreasing (wrt. the denoted relation) infinite sequence of sets of constraints,

$$\Phi_0 = \{\, true \,\}, \quad \Phi_1 \equiv \{\, x \leq y \,\}, \quad \Phi_2 \equiv \{\, x \leq y - 1 \,\}, \quad \Phi_3 \equiv \{\, x \leq y - 2 \,\}, \ldots$$

whose infinite intersection is equivalent to the constraint 'false'. Thus, in a series of four steps as in the introduction of the previous section, we come to intersect $\Phi_1$ with 'false', which corresponds to removing the (here only) constraint $x \leq y - 1$ from $pre_{\mathcal{D}}(\Phi_1)$. What would we do, however, if $pre_{\mathcal{D}}(\Phi_0)$ contained more constraints (obtained e.g. from other events), possibly one that subsumes $x \leq y$? The narrowing step that we define (see Figure 2) first checks a sufficient condition that ensures that the above-mentioned overlap between 'removable' and 'unremovable' constraints does not arise. The condition requires that $\Phi_i$ can be *partitioned* into the constraint $\gamma$ and the set of constraints $\Psi$ (i.e., the conjunction of $\gamma$ and $\Psi$ is unsatisfiable). If the constraint $\phi$ arising in $\Phi_{i+1}$ 'depends on' $\gamma$ via the event $e$, then the following must hold: the predecessors wrt. the event $e$ of states represented by $\gamma$ are represented by $\phi$, and all predecessors of states represented by $\Psi$ are represented by $\Phi_{i+1} \setminus \{\phi\}$; formally

$$\left[\, pre_{\mathcal{D}|e}(\gamma) \,\right]_{\mathcal{D}} \subseteq \left[\, \gamma \,\right]_{\mathcal{D}} \quad \text{and} \quad \left[\, pre_{\mathcal{D}}(\Psi) \,\right]_{\mathcal{D}} \subseteq \left[\, \Psi \,\right]_{\mathcal{D}}.$$

Furthermore, the condition requires that the event $e$ can be applied only to the constraint $\gamma$ (and not to any other constraint in $\Phi_i$). In the appendix we show that this condition ensures that $\gamma$ and $\Psi$ give rise to two disjoint (wrt. solutions) decreasing chains: $pre_{\mathcal{D}}(\Psi_i) = pre_{\mathcal{D}|e}(\gamma) \uplus pre_{\mathcal{D}}(\Psi)$, etc.. Before we apply the narrowing operator to the elevator example, we consider its accuracy.

**Theorem 5.1 (Narrowing)** The algorithm SYMB-EG-N obtained by abstracting the greatest fixpoint operator in the symbolic model checking algorithm SYMB-EG with the narrowing defined in Figure 2 yields (if terminating) a full test of liveness properties for concurrent systems over integers or reals.

The proof of the theorem works by showing that $\bigcap_{j \geq 0} \left[\, \Phi_j \,\right]_{\mathcal{D}} = \bigcap_{j \geq 0} \left[\, \Phi_j^k \,\right]_{\mathcal{D}}$ where the sequence $\{\Phi_i^k\}_{i \geq 0}$ is obtained after $k$-applications of the narrowing operator; see Appendix C.

**Procedure** SYMB-EG-N($\Phi$)
  $\Phi_0 := \{true\}$;
  **repeat**
    $\Phi_{i+1} = \Phi \ \wedge$ NARROW$(\Phi_i, pre_{\mathcal{D}}(\Phi_i))$;
  **until** Entail$_{\mathcal{D}}(\Phi_{i+1}, \Phi_i) =$ true;
**end**

**Function** NARROW$(\{\gamma\} \cup \Psi, \{\phi\} \cup \Phi)$

**if** { $\Phi$ entails $\Psi$ ; $\gamma \equiv \eta \wedge x \leq y + c$ ; $\gamma \wedge \Psi$ is unsatisfiable ; $\phi$ entails $\gamma$ ; $\phi$ depends on $\gamma$ with $e = event(\gamma, \phi)$ ; $e$ is the only event applicable to $\gamma$ ; $e$ contains $x' = x + c_x,\ y' = y + c_y$ where $c_y - c_x < 0$ ; $pre_{\mathcal{D}|e}(\eta)$ entails $\eta$ }

**or if** { $\Phi$ entails $\Psi$ ; $\gamma \equiv \eta \wedge x \geq c$ ; $\gamma \wedge \Psi$ is unsatisfiable ; $\phi$ entails $\gamma$ ; $\phi$ depends on $\gamma$ with $e = event(\gamma, \phi)$ ; $e$ is the only event applicable to $\gamma$ ; $e$ contains $x' = x + c_x$ where $c_x < 0$ ; $pre_{\mathcal{D}|e}(\eta)$ entails $\eta$ }

**then return** $\Phi$;
**else return** $\{\phi\} \cup \Phi$;
**end**

Figure 2: Symbolic model checking procedure with narrowing.

The *elevator program* (see Appendix E) is taken from [BW94]. It is the composition of two processes (the motor and the control panel). The variables are $el$ (the internal state of the elevator), $c$ (the current floor of the elevator), $g$ (the goal, i.e. the current request), and $n$ (the number of floors). When a request is submitted to the control panel, the elevator enters the state *move* and starts moving until the target floor $g$ is reached. The system is parameterized in the number of floors $n$.

Consider the liveness property $AG(el = choose \rightarrow AF(el = ok))$. The call of SYMB-EG$(\{el = move, el = choose\})$ yields an infinite decreasing sequence of sets of constraints $\Phi_0$, $\Phi_1$, $\Phi_2$, ... (see Appendix E.1), where

$$\Phi_2 : \quad \{el = move \wedge c \geq g + 1 \wedge c \geq 0,\ \ el = move \wedge c \leq g - 1\},$$
$$\Phi_3 : \quad \{el = move \wedge c \geq g + 2 \wedge c \geq 1,\ \ el = move \wedge c \leq g - 2\}.$$

We will analyze the effect of the algorithm SYMB-EG-N in the third iteration.

We consider the partion of the set $\Phi_2$ into $\Phi_2 = \{\gamma\} \uplus \Phi$ where $\gamma \equiv el = move \wedge c \geq g + 1$ and $\Phi \equiv \{el = move \wedge c \leq g - 1\}$. We set $\phi \equiv el = move \wedge c \geq g + 2 \wedge c \geq 1$. Observe that the condition of the narrowing rule holds; here, $\eta = c \geq 1$. Thus, we can remove the constraint $el = move \wedge c \geq g + 2 \wedge c \geq 1$ from $pre_{\mathcal{D}}(\Phi_2)$.

At the next iteration, we apply the narrowing rule again, now setting $\gamma \equiv el = move \wedge c \leq g - 2$ and $\Phi = \emptyset$. We derive the empty set of constraints as the fixpoint in the fourth iteration step of SYMB-EG-N. The liveness property is then immediately verified.

# 6 Conclusion and Related Work

Previous approaches to symbolic model checking for integer-valued concurrent systems are based mainly on Presburger arithmetic formulas or on automata as representations of sets of states (see [BGP97, BGP98, Pug92, BW98, SKR98]). Our result about relaxation says that constraints over reals are a potential third alternative with the same precision, at least for the many examples described in Section 3: here, the abstraction of the meaning of integer-valued constraints to the reals is accurate. This means that we can use also the already existing tools based on real arithmetic [BLL+98, HHW+97, DP98]) as full tests on these examples. A comparison of the three approaches wrt. to the performance of the corresponding tools (in the style of [SKR98]) is now in order.

Our widening operator is related to Boigelot and Wolper's loop-first technique [BW94] for deriving 'periodic sets' as representation of infinite sets of integer-valued states for reachability analysis (although they do not use abstract-interpretation terms explicitely). As a difference, Boigelot and Wolper analyze cycles and nested cycles in the control graph to detect meta-transitions *before* and independently of their (forward) model checking procedure, whereas we construct new events (which roughly are meta-transitions) *during* our model checking procedure and consider them only if we detect that they possibly lead to an infinite loop. It will be interesting to formulate their 'widening' in our setup and possibly extend it; note that a set is 'periodic' if it can be represented by an equational constraint with existential variables, e.g. $\exists y\, x = 2y$.

The application of widening techniques to the verification of systems with huge or infinite state spaces has proven useful in several examples (it seems that narrowing for proving liveness properties has not been investigated as much until now). Halbwachs [Hal93], using linear relational analysis [CH78] to prove properties involving integer-valued delay counters of synchronous programs, defines a widening operator over convex polyhedra: unions of polyhedra are approximated by their convex hull before the widening step. This technique is also applied to *linear* hybrid systems, see e.g. [HPR97]. Approximation techniques for more general classes of hybrid systems are studied in [HHW+97, HH95]. Specifically, Henzinger and Ho [HH95] apply an extrapolation operator which gives better approximations than Halbwachs' convex widening operator for their examples. In [BGP97, BGP98], Bultan, Gerber and Pugh generalize Halbwachs' widening operator: their multi-polyhedra widening can be applied to unions of convex polyhedra without the preliminary computation of their convex hull (here, termination is no longer guaranteed). They were thus able to prove the safety property of the ticket algorithm. In [BGP98], they explicitely mention the difficulty that often the abstraction is too rough.

In this paper we have shown that it is possible to achieve (or just accelerate) termination with abstractions by widening or narrowing that are, as we prove, accurate. The above-mentioned widenings loose precision in general. It will be interesting to investigate in which cases these widenings can be made accurate by adding operations as described in this paper. The general goal will be a whole library of useful, accurate widening and narrowing rules for a variety of verification problems.

# References


[AJK+96] P. A. Abdulla, K. Čerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. 10th IEEE Int. Symp. on Logic in Computer Science*, 1995.

[BLL+98] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi and C. Weise. New Generation of UPPAAL. In *Proceedings of the International Workshop on Software Tools for Technology Transfer.* Aalborg, Denmark, 12 - 13 July, 1998.

[BGW+97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDD's. In *Proc. of Int. Static Analysis Symposium*, LNCS 1302, pages 172–186. Springer, Paris, September 1997.

[BW94] B. Boigelot, P. Wolper. *Symbolic verification with periodic sets.* In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV '94)*, D.Dill ed., LCNS 818, pages 55–67. Springer, Stanford, June 1994.

[BW98] B. Boigelot, P. Wolper. Verifying systems with infinite but regular state space. In Alan J.Hu and Moshe Y. Vardi, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '98)*, LNCS 1427, pages 88–97. Springer, Vancouver, June/July 1998.

[Bul98] T. Bultan. *Automated Symbolic Analysis of Reactive Systems*, Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, August 1998.

[BGP97] T. Bultan, R. Gerber and W. Pugh. Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetics. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97)*, O. Grumberg ed, LNCS 1254, pages 400-411. Springer, Haifa, July, 1997.

[BGP98] T. Bultan, R. Gerber, and W. Pugh. Model Checking Concurrent Systems with Unbounded Integer Variables: Symbolic Representations, Approximations and Experimental Results. To appear in ACM Transactions on Programming Languages and Systems. Also available as Technical Report CS-TR-3870, UMIACS-TR-98-07. Department of Computer Science, University of Maryland, College Park, February 1998.

[Čer94] K. Čerāns. Deciding Properties of Integral Relational Automata. In *Proceedings of ICALP 94*, LNCS 820, pages 35-46. 1994.

[CGL92] E.M. Clarke, O. Grümberg and D.E. Long. Model checking and abstraction, In *Proceedings of the 19th Annual Symposium on Principles of Programming Languages (POPL 92)*, pages 343–354. ACM Press, 1992.

[CJ98] H. Comon, Y. Jurski. Multiple Counters Automata, Safety Analysis, and Presburger Arithmetic. In Alan J.Hu and Moshe Y. Vardi, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '98)*, LNCS 1427, pages 268–279. Springer, Vancouver, June/July 1998.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of Fourth the ACM Symposium on Principles of Programmming Languages (POPL '77)* pages 238-252. Los Angeles, January 1977.

*Journal of Logic and Computation*, 2(4):511–547, August 1992.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 1978.

[Dam96] D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD Thesis, Eindhoven University of Technology, 1996.

[DT97] G. B. Dantzig and M. N. Thapa. Linear Programming 1 : Introduction. Springer Series in Operations Research, Springer, Berlin;Heidelberg;New York, 1997.

[DP98] G. Delzanno and A. Podelski. Model Checking in CLP. To appear in *Proceedings of TACAS 99*. Available also as technical report MPI-I-98-2-012 at the URL "www.mpi-sb.mpg.de/ delzanno/clp.html", Max-Planck-Institut für Informatik, Saarbrücken, Germany, July 1998,

[FS98] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere!, April 1998. Technical Report LSV-98-4, Laboratoire Spécification et Vérification, Ecole Normale Supérieure de Cachan.

[FR96] L. Fribourg, J. Richardson. Symbolic verification with gap-order constraints, 1996. *Technical Report LIENS-93-3, Laboratoire d'Informatique, Ecole Normale Superieure, Paris.*

[GN72] R. S. Garfinkel and G. L. Nemhauser. Integer Programming. John Wiley & Sons, 1972.

[Gra94] S. Graf. Verification of a distributed cache memory by using abstractions. In D.L. Dill editor, *Proceedings of Computer-aided Verification (CAV 94)*, LNCS 818, pages 207–219. Springer, 1994.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proceedings of the 5th Conference on Computer-Aided Verification (CAV '93)*, Elounda (Greece), LNCS 697, pages 333-346. Springer, 1993.

[HPR97] N. Halbwachs, Y-E. Proy, and P. Roumanoff. Verification of Real-Time Systems using Linear Relation Analysis. In *Formal Methods in System Design*, 11(2), pp.157-185. August 1997.

[HH95] T. A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, LNCS 999, pages 252–264. Springer, 1995.

[HHW+97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 97)*, O. Grumberg, ed., LNCS 1254, pages 460–463. Springer, 1997.

[KKR95] P. C. Kanellakis and G. M. Kuper and P. Z. Revesz. Constraint Query Languages. Journal of Computer and System Sciences, 51, pages 6–52, 1995.

[KM69] R. M. Karp and R. E. Miller. Parallel Program Schemata. Journal of Computer and System Sciences, 3, pages 147-195, 1969.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97)*, LNCS 1254, pages 424-435. Springer, 1997.

[Lam92] J. Lambert. A structure to decide reachability in Petri nets. Theoretical Computer Science, 99(1), pages 79–1044, 1992.

[LLPW97] K. G. Larsen, F. Larsson, P. Pettersson and W. Yi. Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 14-24. San Francisco, 3-5 December 1997.

[LGS+94] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajiani, S. Bensalem. Property Preserving Abstractions for the Verification of Concurrent Systems. Formal Methods in System Design, pages 1–35. Kluwer Academic, Boston 1994.

[Min67] N. M. Minsky. Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, N.Y., 1967.

[Pug92] W. Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. Communications of the ACM, 8, pages 102-114, August 1992.

[Rev93] P. Z. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1), pages 117–149, 1993.

[Sha93] U. A. Shankar. An introduction to assertional reasoning for concurrent systems. *ACM Computing Surveys*, 25(3),pages 225–262, 1993.

[SKR98] T. R. Shiple, J. H. Kukula, R. K. Ranjan. A Comparison of Presburger Engines for EFSM Reachability. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '98)*, Alan J.Hu and Moshe Y. Vardi, ed., LNCS 1427, pages 280–292. Springer, Vancouver, June/July 1998.

[Sri92] D. Srivistava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. In *2nd International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale*, 1992.

# Appendix to the Extended Abstract

## A Proof of Proposition 3.1

The proof of Proposition 3.1 is based on results from *integer linear programming*. We will first give some preliminaries in the next section.

### A.1 Linear and Integer Programming

An *integer linear problem* consists of finding a solution for a system of inequations having the form $\Sigma_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i = 1, \ldots, m,\ x_j \geq 0,\ j = 1, \ldots, n$, also written as $A\vec{x} \leq \vec{b},\ \vec{x} \geq \vec{0}$ *integer*, where $\vec{x}$ is a $n$-vector of variables, $A$ is a $m \times n$ matrix ($n$ is the number of variables and $m$ is the number of equations) consisting of integer coefficients, and $\vec{b}$ is a $n$-vector of integers. Let us reformulate a problem $A\vec{x} \leq \vec{b}$ as $B\vec{x}_b + N\vec{x}_n \leq \vec{b}$ where $B$ is a non-singular square submatrix of $A$. A solution to $A\vec{x} \leq \vec{b}$ can be obtained by setting $\vec{x}_n = 0$ and by solving the resulting problem wrt $\vec{x}_b$. Such a solution is called *basic* and corresponds to a solution of the problem $\vec{x}_b \leq B^{-1}\vec{b}$.

A fundamental theorem of linear programming says that the solutions of a given problem $A\vec{x} \leq b, \vec{x} \geq 0$ are determined by the *basic* solutions only. Below we recall the results under which the basic solutions are integral. The following definitions and results are taken from [GN72].

**Definition A.1** *A square matrix $A$ is unimodular if its determinant is $= 1$. A matrix $B$ is* totally unimodular *if every square, nonsingular submatrix of $B$ is unimodular.*

**Theorem A.1** *If $A$ is totally unimodular, the extreme points of the set of solutions to $A\vec{x} \leq \vec{b}$ are integer for* arbitrary *integer vector $\vec{b}$.*

As a consequence, if $A$ is totally unimodular the set of real solutions of $A\vec{x} \leq \vec{b}, \vec{x} \geq 0$ is empty if and only if the set of integer solutions is empty (in fact, the extreme points are integer solutions). This result allows one to consider the following *relaxation* in the reals of the previous system: $A\vec{x} \leq \vec{b}, \vec{x} \geq 0$ *real*. A number of interesting properties can be used to identify *totally unimodular* matrices. We listed them below.

**Proposition A.1** *Let $A$ be a $m \times n$ matrix.*

- *If $A$ is totally unimodular than $a_{ij} = 0, 1, -1$, for all $i, j$.*
- *$A$ is totally modular if and only if $A^t$ (the transposed matrix) is totally unimodular.*
- *Let $I_k$ be the $k \times k$ identity matrix. If $A$ is totally unimodular then $(A, I_n)$ and $\binom{A}{I_m}$ are totally unimodular.*

Furthermore, we have the following.

**Theorem A.2** *An integer matrix $A$ ($a_{ij} = 0, 1, -1$ for all $i, j$) is totally unimodular, if*

- *No more than two nonzero elements appear in each column.*
- *The rows can be partitioned into two subsets $Q_1$ and $Q_2$ such that: if a column has two nonzero values with* same sign*, one element is in* each of the partitions*; if a column has two nonzero values with* different sign*, both elements are in the* same partition*.*

## A.2 Simple Constraints and Integer Programming

Let us reformulate a simple constraint in terms of a integer linear programming problem. We first order the variables and the conjuncts occurring in a simple constraint $\phi$, say $x_1, \ldots, x_m$ and $\phi_1, \ldots, \phi_n$. Then, we define the matrix $A$ of the coefficients as the $m \times n$ matrix such that: if $\phi_i = x_p \leq x_q + c_i$ then $a_{ip} = 1$, $a_{iq} = -1$, $b_i = c_i$, and the remaining elements of the $i$-th row are all zero; if $\phi_i = (-)x_p \leq c_i$ then $a_{ip} = (-)1$, $b_i = c_i$, and the remaining elements of the $i$-th row are all zero.

Simple constraints satisfy the following important property.

**Theorem A.3** *Let $\phi$ be a simple constraint and let $A\vec{x} \leq \vec{b}$ be its corresponding matrix form. Then, $A$ is totally unimodular.*

**Proof A.1** *Let $A^t$ be the transpose of $A$. By construction, each column of $A^t$ has* at most *two nonzero elements. Furthermore, if two nonzero elements are in the same column, they have different sign. Now, let $Q_1$ be the set of all the rows and $Q_2$ be the empty set. By Theorem A.2, $A^t$ is totally unimodular. To conclude, we apply Proposition A.1 and obtain that $A$ is totally unimodular.* □

The following corollary easily follows.

**Corollary A.4** *A simple constraint $\phi$ is satisfiable in* **Z** *if and only if it is satisfiable in* **R**.

The previous result allows us to use *linear programming* techniques to handle simple constraints, In particular, we can employ the Fourier-Motzkin Variable Elimination (FMVE) algorithm to compute the projection wrt to a given variable.

**Algorithm FMVE [DT97].** Assume that $x$ is the variable to eliminate in the simple constraint $\psi$. The algorithm consists of the following steps. Firstly, the conjuncts in $\psi$ containing $x$ are partitioned in two sets, say $Q_L$ and $Q_R$, such that $Q_L$ containts the atomic constraints of the form $x \leq y + c$ (i.e. such that $x$ occurs on the left of $\leq$) and $Q_R$ contains the atomic constraints of the form $z - d \leq x$ (i.e. such that $x$ occurs on the right of $\leq$). The variable $x$ is then eliminated by introducing the new constraints $z \leq y + c + d$ (i.e. all the possible combinations of constraints in $Q_R$ and $Q_L$) and by simpliflying redundant ones (e.g. $y \leq d$ and $y \leq c$ can be simplified in $y \leq min\{c, d\}$).

The following result holds.

**Proposition A.2 (Simple constraints are closed under projection)** *Given a simple constraint $\psi$, the FMVE algorithm yields a simple constraint $\varphi$ equivalent to $\psi$.*

**Proof A.2** *The proof that $\varphi$ is a simple constraint is by a case analysis, whereas the equivalence of $\varphi$ and $\psi$ is a consequence of the Fourier-Motzkin's Theorem [DT97].* □

We are now in condition to prove Proposition 3.1. Concerning point *i*), Cor. A.4 and Proposition A.2 proves that the functions $\mathrm{Sat}_{\mathcal{D}}(\cdot)$ and $\mathrm{Elim}_{\mathcal{D}}(\cdot)$ are equivalent when $\mathcal{D} = \mathbf{Z}$ and $\mathcal{D} = \mathbf{R}$. Furthermore, following [Sri92], the entailment test reduces to a number of satisfiability tests. More precisely, given two sets of constraints $\Phi = \{\phi_1, \ldots, \phi_n\}$ and $\Psi = \{\psi_1, \ldots, \psi_m\}$, $\mathrm{Entail}_{\mathcal{D}}(\Phi, \Psi)$ is true if and only if for each $\phi_i$ and for each choice of 'atomic' contraints $\mathcal{A}_{1_{j_1}} \in \psi_1, \ldots, \mathcal{A}_{m_{j_m}} \in \psi_m$, the constraint $\phi_i \wedge \neg\mathcal{A}_{1_{j_1}} \wedge \ldots \neg\mathcal{A}_{m_{j_m}}$ is unsatisfiable. To conclude, we notice that since $\mathcal{A}_i$ is atomic, $\neg\mathcal{A}_i$ is still a simple constraint. Thus, for each satisfiability problem, we can apply again Cor. A.4.

Finally, point *ii*) follows as a corollary of point *i*). □

# B Proof of Theorem 4.1

We prove the theorem for one of the two subcases considered in the narrowing operator (i.e. the one applied to the ticket algorithm). Let us first mention the following properties:

i) $pre, pre_{\mathcal{D}}$ are monotonic wrt. set inclusion, and continuous wrt. set union.

ii) $\bigcup_{i\geq 0} Pre^k(\lceil \Phi_0 \rceil_{\mathcal{D}}) = \bigcup_{i\geq 0} \left\lceil pre^k_{\mathcal{D}}(\Phi_0) \right\rceil_{\mathcal{D}} = \left\lceil \bigcup_{i\geq 0} pre^k_{\mathcal{D}}(\Phi_0) \right\rceil_{\mathcal{D}}$.

iii) $pre_{|e}\Big(\Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}}\Big) = \Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}}$ if $e = event(\gamma, \phi)$, $\gamma$ and $\phi$ as in the conditions of the widening.

In order to prove the accuracy of the widening operator, we first prove that, under the hypothesis considered in Section 4, $\Big\lceil \eta \wedge x \leq y + c + q * (c_y - c_x) \Big\rceil_{\mathcal{D}} \subseteq \Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}}$ for each $q \geq 0$.

The proof is by induction on $q$. The base case immediately follows by assumption. To prove the inductive step let us first notice that, by properties $i$) and $iii$),

$$pre_{|e}\Big(\Big\lceil \eta \wedge x \leq y + c + q * (c_y - c_x) \Big\rceil_{\mathcal{D}}\Big) \subseteq pre_{|e}\Big(\Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}}\Big) \equiv \Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}},$$

and by property of $pre_{\mathcal{D}}$,

$$pre_{|e}\Big(\Big\lceil \eta \wedge x \leq y + c + q * (c_y - c_x) \Big\rceil_{\mathcal{D}}\Big) = \Big\lceil pre_{\mathcal{D}|e}\Big(\eta \wedge x \leq y + c + q * (c_y - c_x)\Big) \Big\rceil_{\mathcal{D}}.$$

Now, by definition of $pre_{\mathcal{D}}$,

$$pre_{\mathcal{D}|e}(\eta \wedge x \leq y + c + q * (c_y - c_x)) \equiv \exists \vec{x'}.\psi \wedge \varphi \wedge \eta[\vec{x'}] \wedge x' \leq y' + c + q(c_y - c_x).$$

By applying the substitution $x' = x + c_x,\ y' = y + c_y$ (which is part of $\varphi$) to $x' \leq y' + c + q(c_y - c_x)$ we obtain the equivalent constraint $\exists \vec{x'}.\Big(\psi \wedge \varphi \wedge \eta[\vec{x'}]\Big) \wedge x \leq y + c + (q+1)(c_y - c_x)$. By hypothesis. $\eta[\vec{x}]$ entails $\exists \vec{x'}.(\psi \wedge \varphi \wedge \eta[\vec{x'}])$, thus we conclude that

$$\Big\lceil \eta \wedge x \leq y + c + (q+1)(c_y - c_x) \Big\rceil_{\mathcal{D}} \in \Big\lceil \bigcup_{i\geq 0} \Phi_i \Big\rceil_{\mathcal{D}}.\Box$$

To conclude the proof, we notice that $\bigvee_{q\geq 0} \eta \wedge x \leq y + c + q(c_y - c_x)$ is equivalent to $\eta \wedge \bigvee_{q\geq 0} x \leq y + c + q(c_y - c_x)$ which in turn is equivalent to true. $\Box$

# C Proof of Theorem 5.1

We first notice that, given a system $\mathcal{S}$ and a set of constraints $\Phi$, by conjoining the constraints in $\Phi$ with the guard of the events in $\mathcal{S}$, we can build a system $\mathcal{S}'$ such that

$$pre_{\mathcal{D}|\mathcal{S}'}(\Psi) = \Phi \wedge pre_{\mathcal{D}|\mathcal{S}}(\Psi).$$

In the following we will assume that $pre_{\mathcal{D}}$ is defined over the modified system $\mathcal{S}'$.

Let us define the chain $\{\Phi_i\}_{i\geq 0}$ as follows: $\Phi_0 = \{true\}$ and $\Phi_{i+1} = pre_{\mathcal{D}}(\Phi_i)$. Furthermore, let us define the family of chains $\{\Phi_i^k\}_{i\geq 0}$ for $k \geq 0$ as follows: $\{\Phi_i^0\}_{i\geq 0} = \{\Phi_i\}_{i\geq 0}$, whereas $\{\Phi_i^{k+1}\}_{i\geq 0}$ is such that $\Phi_0^{k+1} = \text{NARROW}(\Phi_{m_k}^k, pre_{\mathcal{D}}(\Phi_{m_k}^k))$ where $\Phi_{m_k}^k$, $m_k \geq 0$, is the first step of the $k$-th chain for which the conditions of the NARROW operator are all satisfied. Note that, by the assumptions of NARROW, $\left[\,\Phi_1^k\,\right]_{\mathcal{D}} \subseteq \left[\,\Phi_0^k\,\right]_{\mathcal{D}}$ for all $k$. Intuitively, $\{\Phi_i^k\}_{i\geq 0}$ is obtained after $k$-application of the narrowing operator. We now show that the application of the narrowing operator is accurate, i.e.,

$$\bigcap_{j\geq 0} \left[\,\Phi_j\,\right]_{\mathcal{D}} \equiv \bigcap_{j\geq 0} \left[\,\Phi_j^k\,\right]_{\mathcal{D}} \text{ for each k.}$$

The proof is by induction on $k$. The base case immediately follows by definition. Let us assume that the asserts hold for $k$. Let $m_k$ be (the first index) such that $\Phi_{m_k}^k = \{\gamma\} \cup \Psi$ and $\gamma$ and $\Psi$ satisfy the condition of the operator NARROW. We first notice that

$$\bigcap_{j\geq 0} \left[\,\Phi_j^k\,\right]_{\mathcal{D}} = \bigcap_{j> m_k} \left[\,\Phi_j^k\,\right]_{\mathcal{D}} =$$

(as a consequence of the conditions of NARROW)

$$= \bigcap_{j> m_k} \left[\,pre^j_{\mathcal{D}|e}(\gamma)\,\right]_{\mathcal{D}} \cup \left[\,pre^j_{\mathcal{D}}(\Psi)\,\right]_{\mathcal{D}} = \bigcap_{j> m_k} \left[\,pre^j_{\mathcal{D}|e}(\gamma)\,\right]_{\mathcal{D}} \cup \bigcap_{j> m_k} \left[\,pre^j_{\mathcal{D}}(\Psi)\,\right]_{\mathcal{D}},$$

i.e., since the chains produced by $\gamma$ and $\psi$ have empty intersection, we can distribute $\cap$ over $\cup$. Let now us assume that $\gamma = \eta \ \wedge \ x \leq y + c$ and that $e$ contains $x' = x + c_x, y' = y + c_y$ with $c_y - c_x < 0$. By arguments similar to the ones used in the proof of the accuracy of the widening operator, we notice that $\bigcap_{j>m_k} \left[\,pre^j_{\mathcal{D}|e}(\gamma)\,\right]_{\mathcal{D}}$ is represented by the constraint

$$\bigwedge_{q>0} pre^q_{\mathcal{D}|e}(\eta) \wedge v \leq w + c + q(c_y - c_x),$$

which has no solutions, i.e., $\bigcap_{j>m_k} \left[\,pre^j_{\mathcal{D}|e}(\gamma)\,\right]_{\mathcal{D}} = \emptyset$.

Finally, we note that, by definition, $\bigcap_{j>m_k} \left[\,pre^j_{\mathcal{D}}(\Psi)\,\right]_{\mathcal{D}}$ corresponds to $\bigcap_{j\geq 0} \left[\,\Phi_j^{k+1}\,\right]_{\mathcal{D}}$, where the 'seed' of the the $k+1$-th chain is set to $\text{NARROW}(\Phi_{m_k}^k, pre_{\mathcal{D}}(\Phi_{m_k}^k))$. To conclude, we apply the inductive hypothesis and obtain that $\bigcap_{j\geq 0} \left[\,\Phi_j^{k+1}\,\right]_{\mathcal{D}}$ corresponds to $\bigcap_{j\geq 0} \left[\,\Phi_j\,\right]_{\mathcal{D}}$. □

# D Ticket Agorithm

**init** $p_1 = think \wedge p_1 = think \wedge a = 0 \wedge b = 0 \wedge t = 0 \wedge s = 0.$

$e_1$ : **cond** $p_1 = think$ **action** $p_1' = wait \wedge a' = t \wedge t' = t + 1.$

$e_2$ : **cond** $p_1 = wait \wedge a \leq s$ **action** $p_1' = use.$

$e_3$ : **cond** $p_1 = use$ **action** $p_1' = think \wedge s' = s + 1.$

$\ldots$ and similarly for $p_2$ and $b$

## D.1 Least Fixpoint Iterations of SYMB-EF

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $p_1 = use,$ | $p_2 = use$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0,$ | $s \geq 0$ | | |
| $p_1 = wait,$ | $p_2 = use$ | $a \leq s,$ | $b \geq 0,$ | $a \geq 0,$ | $t \geq 0$ | | |
| $p_1 = use,$ | $p_2 = wait$ | $b \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | | |
| $p_1 = think,$ | $p_2 = use$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | | |
| $p_1 = use,$ | $p_2 = think$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | | |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s,$ | $a \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = think,$ | $p_2 = wait$ | $b \leq s,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = think$ | $a \leq s,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = think,$ | $p_2 = think$ | $t \leq s - 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | | |
| $p_1 = use,$ | $p_2 = wait$ | $b \leq s + 1,$ | $t \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | ◀ |
| $p_1 = wait,$ | $p_2 = use$ | $a \leq s + 1,$ | $t \leq s + 1,$ | $b \geq 0,$ | $a \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s + 1,$ | $t \leq s + 1,$ | $a \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s,$ | $t \leq s + 1,$ | $a \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = think,$ | $p_2 = wait$ | $b \leq s + 1,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = think$ | $a \leq s + 1,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = use,$ | $p_2 = wait$ | $b \leq s + 2,$ | $t \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | ◀ |
| $p_1 = wait,$ | $p_2 = use$ | $a \leq s + 2,$ | $t \leq s + 1,$ | $b \geq 0,$ | $a \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s + 2,$ | $t \leq s + 1,$ | $a \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s,$ | $a \leq s + 2,$ | $t \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = think,$ | $p_2 = wait$ | $b \leq s + 2,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = think$ | $a \leq s + 2,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = use,$ | $p_2 = wait$ | $b \leq s + 3,$ | $t \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | ◀ |
| $p_1 = wait,$ | $p_2 = use$ | $a \leq s + 3,$ | $t \leq s + 1,$ | $b \geq 0,$ | $a \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s + 3,$ | $t \leq s + 1,$ | $a \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = wait,$ | $p_2 = wait$ | $b \leq s,$ | $a \leq s + 3,$ | $t \leq s + 1,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ |
| $p_1 = think,$ | $p_2 = wait$ | $b \leq s + 3,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |
| $p_1 = wait,$ | $p_2 = think$ | $a \leq s + 3,$ | $t \leq s,$ | $a \geq 0,$ | $b \geq 0,$ | $t \geq 0$ | |

$\ldots$ non-terminating

## D.2 Least Fixpoint obtained with SYMB-EF-W (Accurate Widening)

$p_1 = use,$ $p_2 = use$ $a \geq 0,$ $b \geq 0,$ $t \geq 0,$ $s \geq 0$
$p_1 = wait,$ $p_2 = use$ $a \leq s,$ $b \geq 0,$ $a \geq 0,$ $t \geq 0$
$p_1 = use,$ $p_2 = wait$ $b \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = think,$ $p_2 = use$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = use,$ $p_2 = think$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = wait$ $b \leq s,$ $a \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = think,$ $p_2 = wait$ $b \leq s,$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = think$ $a \leq s,$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = think,$ $p_2 = think$ $t \leq s - 1,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$

$p_1 = use,$ $p_2 = wait$ $b \leq s + 1,$ $t \leq s + 1,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$ ◀

$p_1 = wait,$ $p_2 = use$ $a \leq s + 1,$ $t \leq s + 1,$ $b \geq 0,$ $a \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = wait$ $b \leq s + 1,$ $t \leq s + 1,$ $a \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = wait$ $b \leq s,$ $t \leq s + 1,$ $a \leq s + 1,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = think,$ $p_2 = wait$ $b \leq s + 1,$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = think$ $a \leq s + 1,$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$

$p_1 = use,$ $p_2 = wait$ $t \leq s + 1,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$ ◀

$p_1 = wait,$ $p_2 = use$ $t \leq s + 1,$ $b \geq 0,$ $a \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = wait$ $t \leq s + 1,$ $a \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = wait$ $b \leq s,$ $t \leq s + 1,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = think,$ $p_2 = wait$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$
$p_1 = wait,$ $p_2 = think$ $t \leq s,$ $a \geq 0,$ $b \geq 0,$ $t \geq 0$

# E Elevator

**init** $el = idle \wedge c = 1 \wedge g = 1 \wedge n \geq 2$.

| | | |
|---|---|---|
| $e_1$ : | **cond** $el = choose$ | **action** $el' = move \wedge g' \leq n-1 \ \wedge \ g' \geq 1$. |
| $e_2$ : | **cond** $el = idle$ | **action** $el' = choose$. |
| $e_3$ : | **cond** $el = move \wedge c = g$ | **action** $el' = idle$. |
| $e_4$ : | **cond** $el = move \wedge c \geq g+1$ | **action** $c' = c-1$. |
| $e_5$ : | **cond** $el = move \wedge c \leq g-1$ | **action** $c' = c+1$. |

## E.1 Greatest Fixpoint Iterations with SYMB-EG

Note that the constraint in $\Phi_2$ contains the conjunct $c \geq 0$ because all variables are interpretet as positive numbers (which is sometimes left implicit).

$\Phi_0$ : $true$.

$\Phi_1$ : $el = move$.

$\Phi_2$ : $el = move \wedge c \geq g+1 \wedge c \geq 0$.
$el = move \wedge c \leq g-1$.

$\Phi_3$ : $el = move \wedge c \geq g+2 \wedge c \geq 1$.
$el = move \wedge c \leq g-2$.

$\Phi_4$ : $el = move \wedge c \geq g+3 \wedge c \geq 2$.
$el = move \wedge c \leq g-3$.

... non-terminating