F. Fioravanti, A. Pettorossi, M. Proietti, V. Senni

A CONSTRAINT-BASED TRANSFORMATION FOR VERIFYING INFINITE STATE SYSTEMS

R. 23, 2011

Fabio Fioravanti – Dipartimento di Scienze, Università ‘G. D’Annunzio’, Viale Pindaro 42, I-65127 Pescara, Italy. Email: fioravanti@sci.unich.it. URL: http://fioravanti.sci.unich.it/fabio.

Alberto Pettorossi – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email: pettorossi@info.uniroma2.it. URL: http://www.iasi.cnr.it/~adp.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email: maurizio.proietti@iasi.cnr.it. URL: http://www.iasi.cnr.it/~proietti.

Valerio Senni – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy. Email: senni@info.uniroma2.it. URL: http://www.disp.uniroma2.it/users/senni.

ISSN: 1128–3378
Collana dei Rapporti dell’Istituto di Analisi dei Sistemi ed Informatica “Antonio Ruberti”, CNR
viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it
Abstract

We address the problem of verifying safety properties of infinite state reactive systems that use unbounded integer variables. We consider systems specified by using linear constraints over the integers and we assume that, for verifying safety properties of these systems, one uses reachability analysis techniques. Our method improves the effectiveness of forward and backward reachability analyses by preprocessing the system specification. For forward reachability our method consists in: (i) transforming the system specification into an equivalent one (with respect to the safety property of interest) by a constraint propagation technique that works backward from the constraints representing the unsafe states, and then (ii) applying to the transformed system specification a reachability analysis that works forward from the constraints representing the initial states. For backward reachability our method works as for forward reachability, by interchanging the roles of the initial states and the unsafe states. We have implemented our method by using the MAP program transformation tool. Our implementation works as a preprocessor for infinite state systems specified in FASTer, a powerful tool for verifying safety properties of infinite state systems with integer variables. Through various experiments performed on several infinite state systems, we have shown that our constraint-based transformation of the system specifications considerably increases the number of successful verifications without a significant degradation of the time performance.

Key words: Program Transformation, Software Verification, Infinite State Systems.
1. Introduction

After the development of very effective techniques for model checking of reactive systems specified by finite state automata [8], the verification community is shifting its interest to classes of more complex system specifications, where one is allowed to model reactive systems with unbounded data structures, and thus, with a possibly infinite set of reachable states.

Counter systems are among the classes of system specifications that have been recently studied: they augment a finite state control structure with unbounded integer variables (see, for instance, [2, 5, 6, 16, 22]). These systems use Presburger formulas to symbolically represent (possibly infinite) sets of states and transitions. For reasons of simplicity, in this paper we consider counter systems defined by linear constraints (that is, equalities and inequalities) over the integers, instead of full Presburger formulas.

It is straightforward to encode as counter systems many classical devices, such as Petri nets (with reset, inhibitor, and transfer arcs) and counter machines (with increment, decrement, and test-if-zero). As a consequence of well-known results in the theory of computation, many properties, like reachability, are undecidable for counter systems. Thus, in order to verify safety properties of counter systems, that is, properties that specify that some given states are not reachable from the initial states, a naive computation of the set of reachable states will not terminate in most cases and, for this reason, several alternative approaches have been proposed.

Several works have identified restricted classes of systems for which the reachability problem is solvable. These classes include various types of Petri nets (see [11] for some results) or, equivalently, vector addition systems [22]. These decidability results are useful when the system to be analysed falls within a specific class (at least in theory, as a tight complexity analysis is still required). However, they do not provide a general method and no action can be performed when the system is outside those decidable classes.

Another approach consists in providing terminating methods for computing a symbolic over-approximation of the set of states that are reachable from the initial states (see, for instance, [1, 7, 17]). Then a safety property is verified if this over-approximated set has an empty intersection with the set of unsafe states. Clearly, methods following this approach are inconclusive in the case where the over-approximated reachability set has a non-empty intersection with the set of unsafe states.

More sophisticated methods for the exact (symbolic) computation of the reachability set of counter systems have been proposed [2, 6, 5, 16]. In particular, these methods use acceleration techniques to improve the convergence of the computation of the reachability set by precomputing new transitions that condense in one step the effect of the transitive closure of a sequence of transitions.

In this paper we focus on methods which allow the exact computation of the reachability set and we propose a transformation technique for improving the convergence of the symbolic computation of that set. The effect of our technique adds on the improvements due to acceleration techniques.

Our starting point is the observation that in some cases during a forward reachability analysis, the computation of the set \( \text{Reach} \) of states that are reachable from the initial states diverges because this set is not representable as a finite set of constraints. However, in order to verify a safety property stating that no element in a set \( \text{Unsafe} \) of unsafe states is reachable, we are actually interested in computing \( \text{Reach} \cap \text{Unsafe} \) and checking whether or not this set is empty. In some cases the computation of this set may terminate even if the computation of \( \text{Reach} \) diverges (consider, for instance, the extreme case where the set \( \text{Unsafe} \) is empty).
The main objective of the technique presented in this paper is to exploit the information about the set $Unsafe$ of the unsafe states during the computation of $Reach$. Our technique does not modify the algorithm for computing the reachability set. On the contrary, it transforms the system specification so that the reachability computation for the new specification incorporates the information about $Unsafe$. By interchanging the roles of the initial states and the unsafe states, our technique can also be applied when performing a backward reachability analysis and it allows us to exploit the information about the set $Init$ of the initial states.

The contributions of this paper are the following ones.

1. We have defined a transformation algorithm that, given a system specification $Spec$ and a set $Unsafe$ of unsafe states, computes a new specification $Spec'$ by propagating the constraints representing $Unsafe$. The propagation process proceeds backwards from $Unsafe$, that is, in opposite direction with respect to the one for computing the set of reachable states from the initial states. Propagation is realized by means of a generalized $Pre$ operator, called $Genpre$, which computes constraints entailed by the states from which a given set of states is reachable in one transition step.

2. We have proved that, under some assumptions on the $Genpre$ operator, the transformation algorithm terminates and produces a new specification $Spec'$ which is equivalent to the given $Spec$ with respect to $Unsafe$. That is, $Spec$ is safe if and only if $Spec'$ is safe. This preservation of equivalence is a notable difference with respect to the techniques that use approximations, such as the ones cited above [1, 7, 17].

3. We have provided several definitions of the $Genpre$ operator that combine (variants of) operators and relations introduced in the field of program analysis and transformation, such as the widening and convex hull operators and the well-quasi order relations on constraints. We have proved that these definitions guarantee termination and correctness of the transformation algorithm.

4. Finally, we have implemented our transformation algorithm on the MAP transformation system [23], an experimental transformation tool written in constraint logic programming (CLP), and we have performed experiments on several infinite state systems by using the FASTer verification tool for counter systems [4]. These experiments show that our transformation determines an increase of the number of successful verifications without a significant degradation of the time performance (actually, in some cases the verification time is highly improved).

Our paper is structured as follows. In Section 2 we present a language based on linear constraints over the integers for specifying counter systems and their safety properties. In Sections 3 and 4 we present our transformation algorithm for counter systems and we prove its termination and correctness. In Section 5 we present some experimental results using FASTer. Finally, in Section 6 we discuss related work in the field of program transformation and analysis.

2. Specifying Counter Systems

In order to specify counter systems and their safety properties we use a simplified version of the languages considered in [3, 4, 21, 25]. Our language allows us to specify systems and properties by using linear constraints over the set $\mathbb{Z}$ of the integers.

A system specification is a 4-tuple $(Var, Init, Trans, Unsafe)$ defined as follows.

(i) $Var$ is a variable declaration which is a sequence of declarations of (distinct) variables, each of which may be either: (i.1) an enumerated variable, or (i.2) an integer variable. (i.1) An enumerated variable $x$ is declared by the statement: $\text{enumerated } x D$, meaning that $x$ ranges over a finite set $D$ of constants. The set $D$ is said to be the type of $x$ and it is also said to be the
type of every constant in $D$. (i.2) An integer variable $x$ is declared by the statement: integer $x$, meaning that $x$ is a variable ranging over the set $\mathbb{Z}$ of the integers.

(ii) $Init$ denotes the set of initial states. It is a set of constraints of the form \{init$_1(X), \ldots, \text{init}_K(X)\}$.

(iii) $Trans$ denotes the transition relation between states. It is a set of constraints of the form \{t$_1(X, X'), \ldots, t_M(X, X')\}$.

(iv) $Unsafe$ denotes the set of unsafe states. It is a set of constraints of the form \{u$_1(X), \ldots, u_N(X)\}$.

Each set of constraints of Points (ii)–(iv) has to be interpreted as a disjunction of constraints. By $X$ we denote the tuple $\langle x_1, \ldots, x_k, x_{k+1}, \ldots, x_n \rangle$ of variables declared in Var, where: (i) for $i = 1, \ldots, k$, $x_i$ is an enumerated variable of type $D_i$, for some finite set $D_i$ of constants, and (ii) for $i = k + 1, \ldots, n$, $x_i$ is an integer variable. By $X'$ we denote the tuple $\langle x'_1, \ldots, x'_k, x'_{k+1}, \ldots, x'_n \rangle$ of the primed variables.

Constraints are defined as follows. If $e_1$ and $e_2$ are enumerated variables or constants of the same type, then $e_1 = e_2$ and $e_1 \neq e_2$ are atomic constraints. If $p_1$ and $p_2$ are linear polynomials with integer coefficients, then $p_1 = p_2$, $p_1 \geq p_2$, and $p_1 > p_2$ are atomic constraints. A constraint is either true, or false, or an atomic constraint, or a conjunction of constraints. By $c(X)$ we denote a constraint on the tuple $X$ of variables. By $C[X]$ we denote the set of all constraints on $X$.

Example 1. In Figure 1 we show a reactive system and its specification.

(1.1) \[
x'_1 = x_1 + x_2 \\
x'_2 = x_2 + 1
\]

(1.2) \[
Var: \quad \text{integer } x_1, \text{ integer } x_2;
Init: \quad \{x_1 \geq 0 \land x_2 \geq 0 \land x_2 \leq 1\};
Trans: \quad \{x'_1 = x_1 + x_2 \land x'_2 = x_2 + 1\};
Unsafe: \quad \{(x_1 < 0 \land x_2 \leq 1), (x_2 \leq -1 \land x_2 \geq -2)\}
\]

Figure 1: A reactive system (1.1) and its specification $Spec$ (1.2). A set of constraints is interpreted as a disjunction of constraints.

Now we define the notion of reachability associated with a system specification.

A state is an $n$-tuple $\langle r_1, \ldots, r_k, z_{k+1}, \ldots, z_n \rangle$ of constants in $D_1 \times \ldots \times D_k \times \mathbb{Z}^{n-k}$. By $\Sigma$ we denote the set of all states. A state $s$ will be called an initial (resp., unsafe) state if there exists init$_i(X) \in Init$ (resp., $u_i(X) \in Unsafe$) such that init$_i(s)$ (resp. $u_i(s)$) holds. A constraint $c(X)$ is less general than a constraint $d(X)$, or $d(X)$ is more general than $c(X)$, denoted $c(X) \sqsubseteq d(X)$, if for all $s \in \Sigma$, if $c(s)$ holds then also $d(s)$ holds. $c(X)$ is said to be equivalent to $d(X)$, denoted $c(X) \equiv d(X)$, if $c(X) \sqsubseteq d(X)$ and $d(X) \sqsubseteq c(X)$.

A computation sequence is a sequence of states $s_0, \ldots, s_m$, with $m \geq 0$, such that, for $i = 0, \ldots, m-1$, $t(s_i, s_{i+1})$ holds, for some $t(X, X') \in Trans$. State $s_m$ is reachable from state $s_0$ if there exists a computation sequence $s_0, \ldots, s_m$. A system specification is safe if there is no unsafe state which is reachable from an initial state. Two system specifications $Spec_1$ and $Spec_2$ are said to be equivalent if $Spec_1$ is safe iff $Spec_2$ is safe.

In order to compute the possibly infinite set of states which are (forward) reachable from an initial state, available verification tools make use of symbolic reachability algorithms which represent set of states as Presburger formulas, or equivalent automata-theoretic notions [3, 4, 5].
Input: A specification $Spec$.
Output: A new specification $Spec'$ equivalent to $Spec$.

**Phase 1:** Construction of the tree $CTree$ of constraints.

Let $Genpre(\Lambda)$ be the set $\{g_1(X), \ldots, g_R(X)\}$.

$$CTree := \{\Lambda \leftarrow \emptyset, g_1(X), \ldots, \Lambda \leftarrow g_R(X)\}.$$  

while there exists a non-recurrent constraint $c(X)$ labelling a leaf of $CTree$ do

let $Genpre(c(X))$ be the set $\{g_1(X), \ldots, g_S(X)\}$;

for $j = 1, \ldots, S$, let $T_j$ be $\{t_i(X, X') \in Trans \mid \pi(i) = j\}$;

$$CTree := CTree \cup \{c(X) \leftarrow T_1, g_1(X), \ldots, c(X) \leftarrow T_S, g_S(X)\};$$

od;

**Phase 2:** Extraction of the new specification $Spec'$ from the tree $CTree$ of constraints.

$SpecExtract(\text{Spec, } CTree, \text{Spec'})$

Figure 2: The Specification Transformer algorithm.

21, 25]. Similar symbolic techniques are used for computing the set of states from which an unsafe state is (backward) reachable. In this paper we will not need to take into consideration any specific algorithm for computing reachability sets, as our technique transforms a system specification into an equivalent one, independently of any such algorithm.

### 3. An Algorithm for Transforming Specifications

In this section we present our method for improving the reachability analysis of counter systems by applying constraint-based transformations of system specifications. Here we consider the forward reachability, but our method can also be applied to the case of backward reachability by interchanging the roles of initial states and unsafe states.

Our method is based on an algorithm that transforms a specification $Spec = \langle Var, Init, Trans, Unsafe \rangle$ into an equivalent specification $Spec' = \langle Var', Init', Trans', Unsafe' \rangle$ for which the reachability analysis is hopefully more effective. The objective of our transformation is to derive, from the transition relation $Trans$ and the set $Unsafe$ of the unsafe states, a new transition relation $Trans'$ such that by using the relation $Trans'$, instead of $Trans$, the unsafe states are implicitly taken into consideration when constructing the set of the reachable states.

Our Specification Transformer algorithm (see Figure 2) consists of two phases:

(Phase 1) the construction of a tree of constraints, called $CTree$, by using the procedure $Genpre$ from the given specification $Spec$, and

(Phase 2) the extraction of a new specification $Spec'$ from the tree $CTree$ by using the procedure $SpecExtract$.

In the tree $CTree$, (i) the root node is labelled by a distinguished symbol $\Lambda$ (which stands for the constraint $false$), (ii) each non-root node is labelled by a constraint in $C[X]$, and (iii) each arc is labelled by a set of transitions. When no confusion arises, we will identify a node with the constraint which labels it. An arc from a node (or constraint) $c_1(X)$ to a node (or constraint) $c_2(X)$ labelled by a set $T \subseteq Trans$ of transitions is denoted by $c_1(X) \leftarrow T \rightarrow c_2(X)$. For simplicity, $CTree$ will be represented as the set of its labelled arcs.

In the tree $CTree$ the following hold: (i) the disjunction of all constraints in the non-root nodes is an over-approximation of the set of states from which an unsafe state is reachable, and
(ii) for every arc \( c_1(X) \xrightarrow{T} c_2(X) \), we have that \( c_2(X) \) is an over-approximation of the set of states from which a state \( X \) satisfying \( c_1(X) \) can be reached in one step by using one of the transitions in \( T \).

In order to construct \( CTree \) we will use the following operator.

**Definition 3.1.** A generalized \( \text{pre} \) operator, called \( \text{Genpre} \), is a function which, for any \( a \in \mathcal{C}[X] \cup \{ \Lambda \} \),

- if \( a = \Lambda \), then it returns a set \( \{ g_1(X), \ldots, g_R(X) \} \) of constraints in \( \mathcal{C}[X] \) and a surjective mapping \( \lambda : \{ 1, \ldots, N \} \rightarrow \{ 1, \ldots, R \} \), with \( R \leq N \) (where \( N \) is the cardinality of \( \text{Unsafe} \)), such that, for \( i = 1, \ldots, N \), \( u_i(X) \subseteq g_{\lambda(i)}(X) \);
- else if \( a = c(X) \) for some \( c(X) \in \mathcal{C}[X] \), then it returns a set \( \{ g_1(X), \ldots, g_S(X) \} \) of constraints in \( \mathcal{C}[X] \) and a surjective mapping \( \pi : \{ 1, \ldots, M \} \rightarrow \{ 1, \ldots, S \} \), with \( S \leq M \) (where \( M \) is the cardinality of \( \text{Trans} \) and both \( \pi \) and \( S \) may depend on \( c(X) \)), such that, for \( i = 1, \ldots, M \), \( c(X) \triangleq t_i(Y, X) \subseteq g_{\pi(i)}(Y) \).

Note that the \( \text{Genpre} \) operator depends on the input \( a \), but also on: (i) the set \( \text{Trans} \) of transitions, and (ii) the upper portion of the tree \( CTree \) constructed so far. However, for reasons of simplicity, we do not explicitly indicate \( \text{Trans} \) and \( CTree \) among the arguments of the operator \( \text{Genpre} \) and we indicate the argument \( a \) only.

The tree \( CTree \) is incrementally constructed as follows. Initially, \( CTree \) is given by the \( R \) arcs:

\[
\Lambda \xrightarrow{\hat{1}} g_1(X), \ldots, \Lambda \xrightarrow{\hat{1}} g_R(X), \quad \text{where} \quad \{ g_1(X), \ldots, g_R(X) \} = \text{Genpre}(\Lambda).
\]

Suppose that we have constructed an upper portion \( U \) of \( CTree \). Let us introduce the following terminology.

A leaf constraint \( c(X) \) of the tree \( U \) is said to be recurrent if there exists a non-leaf constraint \( d(X) \) in \( U \) such that \( c(X) \subseteq d(X) \).

Now, if all leaf constraints in \( U \) are recurrent, then the construction of \( CTree \) terminates. Otherwise, if there exists a non-recurrent leaf constraint \( c(X) \) in \( U \), then the current upper portion \( U \) of \( CTree \) is expanded by adding the \( S \) arcs: \( c(X) \xrightarrow{T_1} g_1(X), \ldots, c(X) \xrightarrow{T_S} g_S(X) \), where:

- \( \{ g_1(X), \ldots, g_S(X) \} = \text{Genpre}(c(X)) \), and
- for \( j = 1, \ldots, S \), \( T_j = \{ t_i(X, X') \in \text{Trans} \mid \pi(i) = j \} \). Recall that the mapping \( \pi \) is provided by \( \text{Genpre}(c(X)) \).

In order to guarantee the termination of the construction of \( CTree \) we assume that our \( \text{Genpre} \) operator is terminating, in the sense specified by the following definition.

**Definition 3.2.** A \( \text{Genpre} \) operator is said to be terminating if, for every infinite sequence \( c_1(X), c_2(X), \ldots \) of constraints such that: (i) \( c_1(X) \in \text{Genpre}(\Lambda) \), and (ii) for every \( i > 0 \), \( c_{i+1}(X) \in \text{Genpre}(c_i(X)) \), there exist \( k \) and \( n \), with \( 0 < k < n \), such that \( c_n(X) \subseteq c_k(X) \).

In the next section we define the three terminating \( \text{Genpre} \) operators we have used in our experiments.

**Theorem 3.3.** For any specification \( \text{Spec} \) and any terminating \( \text{Genpre} \) operator, the construction of \( CTree \) terminates.

From \( CTree \) a new system specification \( \text{Spec}' = \{ \text{Var}', \text{Init}', \text{Trans}', \text{Unsafe}' \} \) can be extracted by the \text{SpecExtract} procedure described below. In particular, this procedure adds new control states, by introducing an enumerated variable \( s \).
Procedure SpecExtract(Spec, CTree, Spec').

Input: A specification Spec = (Var, Init, Trans, Unsafe) and the tree CTree constructed in Phase 1 of the Specification Transformer.

Output: A new specification Spec' that is equivalent to Spec.

Step 1. We introduce a set CS of new constants (the name CS stands for ‘control states’) starting from the nodes in the tree CTree and Unsafe. We associate a constant \( k_\Lambda \in CS \) with the root symbol Λ. With each constraint \( c(X) \) occurring in a node of CTree or in Unsafe, we associate a constant \( k_c \in CS \), such that for all constraints \( c(X), d(X) \) occurring in a node of CTree or in Unsafe, \( k_c = k_d \) iff \( c(X) \equiv d(X) \).

We also introduce a folding function \( \varphi : (CS - \{k_\Lambda\}) \rightarrow CS \) such that, for all constraints \( c(X) \) and \( g(X) \) associated with the constants \( k_c \) and \( k_g \), respectively, if \( \varphi(k_c) = k_g \), then \( c(X) \equiv g(X) \), that is, \( g(X) \) is more general than \( c(X) \).

Step 2. We introduce a new enumerated variable \( s \) (ranging over a subset of CS and declared as indicated at Step 4 below) and the corresponding primed variable \( s' \). We define the set \( \text{Trans} \) of constraints to be

\[
\{ s = \varphi(k_d) \land t_i(X, X') \land c(X) \land s' = k_c \mid c(X) \xrightarrow{T} d(X) \in \text{CTree} \text{ and } t_i(X, X') \in T \text{ and } (t_i(X, X') \land c(X')) \text{ is satisfiable} \}.
\]

Note that every arc arriving at \( k_\Lambda \) having the empty set as label, does not contribute to \( \text{Trans} \).

Step 3. We define on the set CS the ‘depends on’ relation which is the transitive closure of the ‘immediately depends on’ binary relation defined as follows.

(i) \( k_\Lambda \) immediately depends on \( \varphi(k_c) \) for every constraint \( c(X) \) labelling a child node of \( \Lambda \) in the tree CTree.

(ii) For any two constants \( k, k' \in CS \), we say that \( k' \) immediately depends on \( k \) if there exists a constraint \( u(s, X, s', X') \in \text{Trans} \), where the two equalities \( s = k \) and \( s' = k' \) occur.

Step 4. We have that Spec' is \( \langle \text{Var}', \text{Init}', \text{Trans}', \text{Unsafe}' \rangle \), where:

- \( \text{Var}' \) is obtained by adding to Var the variable declaration enumerated \( s \) CS', where CS' = \{ \( k \in CS \mid k_\Lambda \) depends on \( k \) \};
- \( \text{Init}' \) is \( \{ s = k_c \land c(X) \land \text{init}(X) \mid k_\Lambda \) depends on \( k_c \) and \( \text{init}(X) \in \text{Init} \) and \( (c(X) \land \text{init}(X)) \) is satisfiable \};
- \( \text{Trans}' \) is \( \{ u(s, X, s', X') \in \text{Trans} \mid \) equality \( s' = k \) occurs in \( u(s, X, s', X') \) and \( k_\Lambda \) depends on \( k \} \);
- \( \text{Unsafe}' \) is \( \{ s = \varphi(k_d) \land u(X) \mid u(X) \in \text{Unsafe} \} \).

Note that at Step 1 of the SpecExtract procedure we may choose \( \varphi \) to be a function such that \( \varphi(k_c) = k_g \) for some constraint \( g(X) \) which is maximally general, that is, for all \( d(X) \) labelling a node of CTree, if \( g(X) \equiv d(X) \) then \( g(X) \equiv d(X) \). The function \( \varphi \) will be called a maximally general folding.

Theorem 3.4 (Correctness of the Specification Transformer) For any specification Spec and terminating GENPRE operator, the Specification Transformer algorithm terminates and returns a specification Spec' which is equivalent to Spec.

Example 2. In Figures 3 and 4 we show the two-phase construction of the new specification Spec' starting from the system specification of Figure 1. In the left part of Figure 3 we show
Figure 3: On the left we show the constraint tree $CTree$ rooted in $k_\Lambda$ for the system specification of Figure 1. Every arc $k_m \xrightarrow{T} k_n$ is obtained by applying the GENPRE operator with the options Singleton and WidenSum. In the middle we show the correspondence between the constraints and the associated constants $k_1, \ldots, k_5$. On the right we show the maximally general folding function $\varphi$.

Figure 4: On the left we show the relation $Trans'$: the arc $k_m \xrightarrow{t} k_n$ indicates that $k_m$ depends on $k_n$ and $t \in Trans'$ (the dashed arcs show that $k_\Lambda$ depends on $k_1$ and $k_5$). On the right we show the specification $Spec'$ derived from the system specification of Figure 1. In $Trans'$ and $Unsafe'$ comma is interpreted as disjunction.

the $CTree$ constructed at the end of Phase 1 of the Specification Transformer algorithm. For reasons of simplicity, in the nodes of the tree $CTree$ in Figure 3, instead of the constraints, we have indicated their associated constants $k_\Lambda, k_1, \ldots, k_5$ (to be introduced at Step 1 of the SPECExtract procedure). The correspondence between the constraints and the constants is given in the middle part of Figure 3. Let $t$ be an abbreviation for $x'_1=x_2+x_1 \land x'_2=x_2+1$, which is the only constraint in $Trans$. In what follows, when no confusion arises, we will identify the constraints with their associated constants.

The specific GENPRE operator we consider, uses the option Singleton for partitioning and WidenSum for generalizing (see Section 4).

At the beginning of Phase 1 of the derivation of the new system specification, $\text{GENPRE}(\Lambda)$ returns the two constraints $k_1$ (that is, $x_1<0 \land x_2\leq 1$) and $k_2$ (that is, $x_2\leq-1 \land x_2\geq-2$) and introduces the two arcs $k_\Lambda \xrightarrow{\{} k_1$ and $k_\Lambda \xrightarrow{\{} k_2$. Note that $k_1$ and $k_2$ are exactly the two constraints in $Unsafe$ and, with reference to the Specification Transformer algorithm of Figure 2, this means that: (i) $R=N=2$, (ii) the function $\lambda$ returned by $\text{GENPRE}$ is the identity function,
Then, at Step 3, from the set 

\{\text{not arrive at Unsafe}(i)\} the enumerated variable ranges over the set 

\{k_1, k_2\}, which return, respectively, the two constraints \(k_3\) (that is, \(x_1 + x_2 < 0 \land x_2 \leq 0\)) and \(k_4\) (that is, \(x_2 \leq -2\)) and introduces the two arcs \(k_3 \xrightarrow{(t)} k_3\) and \(k_2 \xrightarrow{(t)} k_4\). Now, a further application of GENPRE to \(k_4\) returns the same constraint and introduces the arc \(k_4 \xrightarrow{(t)} k_4\), which indicates that \(k_4\) is recurrent. The construction continues by applying GENPRE to \(k_3\), which returns \(k_5\) (that is, \(x_2 \leq 0\)) and introduces the arc \(k_3 \xrightarrow{(t)} k_5\). One more application of GENPRE to \(k_5\) returns the same constraint and introduces the arc \(k_5 \xrightarrow{(t)} k_5\). Now, since also \(k_5\) is recurrent, the construction of the tree \(CTree\) is completed.

We proceed with Phase 2 by considering the maximally general folding function \(\varphi\) whose definition is as follows: \(\varphi(k_1) = k_1\) and \(\varphi(k_2) = \varphi(k_3) = \varphi(k_4) = \varphi(k_5) = k_5\) (see the right part of Figure 3). The function \(\varphi\) satisfies the condition of Step 1 because \(k_5\) is more general than \(k_2, k_3, k_4,\) and \(k_5\) itself, while it is incomparable with \(k_1\).

At Step 2 we compute the following set \(\text{Trans}\) of constraints (every arc in \(CTree\) does not arrive at \(k_\Lambda\), generates an element of \(\text{Trans}\)):

\[
\{s = \varphi(k_3) \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_1' < 0 \land x_2' \leq 1 \land s' = k_1, \\
s = \varphi(k_4) \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_1' \leq 1 \land x_2' \leq 2 \land s' = k_2, \\
s = \varphi(k_5) \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_1' + x_2' < 0 \land x_2' \leq 0 \land s' = k_3, \\
s = \varphi(k_4) \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_2' \leq 2 \land s' = k_4, \\
s = \varphi(k_5) \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_2' \leq 0 \land s' = k_5\}
\]

Then, at Step 3, from the set \(\text{Trans}\) we compute the \textit{immediately depends on} relation which is the set \(\{(k_\Lambda, k_1), (k_\Lambda, k_5), (k_1, k_5), (k_2, k_5), (k_3, k_5), (k_4, k_5), (k_5, k_5)\}\). We have that \(k_\Lambda\) \text{ depends on } \(k_1\) and \(k_5\) and only on those constraints. The constraints on which \(k_\Lambda\) depends are required in the following Step 4 for constructing \(\text{Trans}'\) starting from \(\text{Trans}\).

Finally, at Step 4 we extract the new specification \(Spec'\) (see Figure 4) as follows:

(i) the enumerated variable \(s\) ranges over the set \(\{k_1, k_5\}\) of constraints on which \(k_\Lambda\) depends,

(ii) \(Init'\) is \(s = k_5 \land x_2 \leq 0 \land Init\), where \(Init\) is the constraint \(x_1 \geq 0 \land x_2 \geq 0 \land x_2 \leq 1\) (note that \(s = k_1 \land x_1 \leq 0 \land x_2 \leq 1 \land Init\) is unsatisfiable), and thus, \(Init'\) can also be rewritten as \(s = k_5 \land x_1 \leq 0 \land x_2 = 0\),

(iii) \(Trans'\) is built by selecting from \(\text{Trans}\) every constraint that assigns to the variable \(s'\) either the value \(k_1\) or the value \(k_5\); there are two of them

\[
(s = k_5 \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_1' < 0 \land x_2' \leq 1 \land s' = k_1), \quad (t_{15})
\]

\[
(s = k_5 \land x_1' = x_2 + x_1 \land x_2' = x_2 + 1 \land x_2' \leq 0 \land s' = k_5), \quad (t_{55})
\]

(iv) \(Unsafe'\) is obtained from \(Unsafe = \{u_1(X), u_2(X)\}\), where \(u_1(X)\) is \((x_1 < 0 \land x_2 \leq 1)\) and \(u_2(X)\) is \((x_2 \leq -1 \land x_2 \geq 2)\). Now, \(k_1\) is the constant associated with \(u_1\) and \(k_2\) is the constant associated with \(u_2\) (see Figure 3). Thus, \(Unsafe' = \{s = \varphi(k_1) \land u_1(X), s = \varphi(k_2) \land u_2(X)\}\), that is, \(Unsafe' = \{(s = k_1 \land x_1 < 0 \land x_2 \leq 1), (s = k_5 \land x_2 \leq -1 \land x_2 \geq 2)\}\).


In this section we present the definitions of various GENPRE operators that are used in the construction of the constraint tree \(CTree\). We will use variants of generalization operators, such
as the convex-hull operator and the widening operator, which are introduced in the field of static program analysis. In our construction of the tree CTree these operators should act on the integers \( \mathbb{Z} \) and, unfortunately, in this case they are not as efficient as in the case in which they act on the reals \( \mathbb{R} \). Thus, in order to overcome this difficulty and have an efficient implementation of our generalization operators, we use a relaxation from the integers \( \mathbb{Z} \) to the reals \( \mathbb{R} \), that is, we use solvers on the reals \( \mathbb{R} \) for performing the unsatisfiability tests, the entailment tests, and the projection operations which are required by our generalization operators.

In [15] it is shown that, for any given finite domains \( D_1, \ldots, D_k \), the tree CTree derived by using a relaxation from \( D_1 \times \ldots \times D_k \times \mathbb{Z}^{n-k} \) to \( D_1 \times \ldots \times D_k \times \mathbb{R}^{n-k} \) allows us to get a specification which is equivalent to the given specification. Details on this point can be found in [15].

In the definition of the GENPRE operators we distinguish between: (i) atomic constraints on enumerated variables and (ii) atomic constraints on variables ranging over the integers. Thus, any constraint \( c \) can be partitioned into: (i) a (possibly empty) conjunction of equalities on enumerated variables, denoted \( \text{fd}(c) \) (\( \text{fd} \) stands for ‘finite domain’), and (ii) a (possibly empty) conjunction of linear inequalities on the integers, denoted \( \text{it}(c) \) (it stands for ‘integer type’). Note that the set of all conjunctions of equalities on enumerated variables can be viewed as a finite lattice whose underlining partial order is defined by the entailment relation \( \square \). Given the set of constraints \( \{c_1, \ldots, c_n\} \), we define their most specific generalization, denoted \( \gamma(\{c_1, \ldots, c_n\}) \), to be the conjunction of: (i) the least upper bound of the conjunctions \( \text{fd}(c_1), \ldots, \text{fd}(c_n) \) of equalities on enumerated variables, and (ii) the convex hull [9] of the constraints \( \text{it}(c_1), \ldots, \text{it}(c_n) \) on the reals \( \mathbb{R} \). For \( i = 1, \ldots, n \), we have that \( c_i \sqsubseteq \gamma(\{c_1, \ldots, c_n\}) \).

Given a set of constraints \( Cs = \{c_1, \ldots, c_n\} \), we introduce the equivalence relation \( \simeq_{\text{fd}} \) on \( Cs \) such that, for every \( c_1, c_2 \in Cs \), \( c_1 \simeq_{\text{fd}} c_2 \) iff \( \text{fd}(c_1) \) is equivalent to \( \text{fd}(c_2) \). We also define the equivalence relation \( \simeq_{\text{it}} \) on \( Cs \) as the reflexive, transitive closure of the relation \( \sqsubseteq_{\mathbb{R}} \) on \( Cs \) such that, for every \( c_1, c_2 \in Cs \), \( c_1 \sqsubseteq_{\mathbb{R}} c_2 \) iff \( \text{it}(c_1) \sqsubseteq \text{it}(c_2) \) is satisfiable in \( \mathbb{R} \).

For example, let the constraint \( c_1 \) be \( x_1 = a \land x_2 > 0 \), for some \( a \) in a finite domain \( D \), and the constraint \( c_2 \) be \( x_1 = a \land x_2 < 0 \). We have that \( c_1 \simeq_{\text{fd}} c_2 \) on \( \{c_1, c_2\} \). Let \( c_3 \) be the constraint \( x_1 > 0 \land x_1 < 2 \), \( c_4 \) be the constraint \( x_1 > 1 \land x_1 < 3 \), and \( c_5 \) be the constraint \( x_1 > 2 \land x_1 < 4 \). Since \( c_3 \sqsubseteq_{\mathbb{R}} c_4 \) and \( c_4 \sqsubseteq_{\mathbb{R}} c_5 \), we have \( c_3 \simeq_{\text{it}} c_5 \) on \( \{c_3, c_4, c_5\} \). Note that \( c_3 \not\simeq_{\text{it}} c_5 \) on \( \{c_3, c_5\} \) because \( c_3 \land c_5 \) is not satisfiable in \( \mathbb{R} \).

Given a specification \( \text{Spec} = (\text{Var}, \text{Init}, \text{Trans}, \text{Unsafe}) \) and a leaf node \( L \) of the constraint tree CTree, the GENPRE operator constructs the child nodes \( g_1(X), \ldots, g_S(X) \) of \( L \) in the following three steps: (1) Unfold, (2) Partition, and (3) Generalize.

(Step 1). (Unfold) If \( L = \Lambda \), we define \( U \) to be the set \( \{u_1(Y), \ldots, u_N(Y)\} \) of the constraints in \( \text{Unsafe} \). If \( L \) is a constraint \( c(X) \), we define \( U \) to be the set \( \{d_1(Y), \ldots, d_M(Y)\} \) of \( \mathbb{R} \)-satisfiable constraints obtained as follows.

First, we use the transitions in the set \( \text{Trans} \) arriving at the node \( c(X) \) and we obtain the set \( \text{Temp}U = \{c(X) \land t_1(Y, X), \ldots, c(X) \land t_M(Y, X)\} \) of constraints. Then we compute the \( \mathbb{R} \)-projection of each element in \( \text{Temp}U \) on the tuple of variables \( Y \), thereby deriving the set \( \{d_1(Y), \ldots, d_M(Y)\} \) of \( \mathbb{R} \)-satisfiable constraints.

(Step 2). (Partition) The partition of the set \( U \) consists of the equivalence classes \( E_1, \ldots, E_S \), for \( S \geq 1 \), induced on \( U \) by one of the following equivalence relations:

(Singleton) no two constraints are equivalent (every block is a singleton);

(Chain) for \( i, j = 1, \ldots, M \), two constraints \( d_i(Y) \) and \( d_j(Y) \) are equivalent iff \( d_i(Y) \simeq_{\text{fd}} d_j(Y) \) and \( d_i(Y) \simeq_{\text{it}} d_j(Y) \) on \( \{d_1(Y), \ldots, d_M(Y)\} \);
constructed as follows. Let $E$ replaced by 3.1 Let

Step 3.1 Let $b(X)$ denote the most specific generalization $\gamma(\{d_1(X), \ldots, d_K(X)\})$.

if there exists a nearest ancestor $a_1(X)$ of $c(X)$ (possibly $c(X)$ itself) in $CTree$ such that

\[ a_1(X) \preceq_d c(X) \]

then $b_{anc}(X) = \gamma(\{a_1(X), b(X)\})$ else $b_{anc}(X) = b(X)$;

Step 3.2 Let us consider a generalization operator $\ominus$ (for instance, the operator WidenSum defined below).

if in $CTree$ there exists a constraint $d(X)$ such that $b_{anc}(X) \subseteq d(X)$ in $\mathbb{R}$

then $g_i(X) = d(X)$

else if there exists a nearest ancestor $a_2(X)$ of $c(X)$ (possibly $c(X)$ itself) in $CTree$ such that

\[ a_2(X) \preceq_d b_{anc}(X) \]

then $g_i(X) = a_2(X) \ominus b_{anc}(X)$ else $g_i(X) = b_{anc}(X)$.

The generalization operator we have considered in our experiments is the WidenSum operator, denoted $\ominus_{WS}$, and it has been defined and studied in [13]. As shown in [13], it performs quite well in practice with respect to other generalization operators introduced in the literature. For convenience of the reader we now recall its definition.

We first introduce the thin well-quasi ordering $\preceq_S$. For any atomic constraint $a$ on $\mathbb{R}$ of the form $q_0 + q_1 x_1 + \ldots + q_k x_k < 0$, where $<$ is either $<$ or $\leq$, we define $\text{sumcoeff}(a)$ to be $\sum_{j=0}^{k} |q_j|$. Given two atomic constraints $a_1$ of the form $p_1 < 0$ and $a_2$ of the form $p_2 < 0$, we have that $a_1 \preceq_S a_2$ iff $\text{sumcoeff}(a_1) \leq \text{sumcoeff}(a_2)$. Similarly, if we are given the atomic constraints $a_1$ of the form $p_1 \leq 0$ and $a_2$ of the form $p_2 \leq 0$. Given any two constraints $c = a_1 \land \ldots \land a_m$ and $d = b_1 \land \ldots \land b_n$, where the $a_i$’s and the $b_i$’s are atomic constraints, the operator WidenSum returns the constraint $c \ominus_{WS} d$ which is the conjunction of the constraints in the set $\{a_h \mid 1 \leq h \leq m \land d \subseteq a_h\} \cup \{b_k \mid b_k \text{ occurs in } it(d) \land \exists a_i \text{ occuring in } it(c), b_k \preceq_S a_i\}$. Note that it is the case that $fd(d)$ is a subconjunction of $c \ominus_{WS} d$.

5. Experimental Evaluation

In this section we present the results of the experiments we have performed for verifying safety properties of some infinite state systems taken from the literature [4, 10, 25].

We have performed our experiments by using the FASTer tool, which is designed to prove safety properties of linear counter systems [4]. FASTer performs forward reachability analysis by an exact computation of the least fixpoint of the transition relation of an accelerated version of the system, which is obtained by adding transitions which realize the effect of sequences of transitions.

We have run FASTer using its best options, that is, the options which make it terminate more often, among those reported on the FASTer home page. In particular, we have used the MONA system for manipulating constraints, and we have used a fixed value as the seed of the random number generator.

The constraint manipulation techniques presented in this paper were implemented on MAP [23], a tool for transforming CLP programs which uses the SICStus Prolog clp fr library to operate on constraints on the reals. (Recall that the Genpre operators defined in Section 4 manipulate constraints on the reals.)
All experiments were performed on an Intel Core 2 Duo E7300 2.66 GHz with 4GB of memory, under Linux. The results of our experiments are reported in Table 1, where we have indicated, for each counter system, the following times expressed in seconds: (i) the time taken by FASTer for verifying the given system (column System), and (ii) for each Genpre operator, the sum of the time taken by MAP for transforming the system and the time taken by FASTer for verifying the transformed system (columns Singleton, Chain, and All).

<table>
<thead>
<tr>
<th>EXAMPLES</th>
<th>System (= Spec)</th>
<th>Transformed System (= Spec')</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Singleton</td>
<td>Chain</td>
</tr>
<tr>
<td>Bakery2</td>
<td>24.08</td>
<td>0.09</td>
</tr>
<tr>
<td>Bakery3</td>
<td>$\infty$</td>
<td>2.25</td>
</tr>
<tr>
<td>Mutast</td>
<td>0.48</td>
<td>1.32</td>
</tr>
<tr>
<td>Ticket</td>
<td>0.27</td>
<td>0.94</td>
</tr>
<tr>
<td>Bounded Buffer</td>
<td>$\infty$</td>
<td>0.21</td>
</tr>
<tr>
<td>Unbounded Buffer</td>
<td>0.06</td>
<td>0.19</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$\infty$</td>
<td>0.78</td>
</tr>
<tr>
<td>Scheduler2</td>
<td>0.20</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Train</td>
<td>8.86</td>
<td>1.06</td>
</tr>
<tr>
<td>TTP</td>
<td>112.88</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Number of verified properties</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 1: Verification times (in seconds) using FASTer [4]. ‘$\infty$’ means ‘No answer’ within 5 minutes or memory limit exceeded.

The experiments show that the overall precision, that is, the number of verified properties, achieved by FASTer on the systems obtained by applying our transformation technique, is never worse than the precision of FASTer on the original systems.

By comparing the precision of the Genpre operators, we observe that: (i) the Chain operator is the most precise, being able to prove 9 out of 10 safety properties (all, except Selection Sort); (ii) the Singleton operator is able to improve the precision for the three systems whose safety property could not be proved by FASTer (Bakery3, Bounded Buffer and Selection Sort), but it fails on two other systems (Scheduler and TTP); (iii) the All operator, with 7 properties out of 10, is the least precise (actually, it has the same precision of FASTer on the original systems).

If we consider the verification times, we have that the time in column System and the time in the other columns are of the same order of magnitude in almost all cases. Sometimes (and, in particular, in the case of the Bakery2 system) our method substantially reduces the total verification time.

Moreover, we have compared the total verification times of the given systems and the derived systems for which FASTer is able to prove the safety property, that is, we computed the total time of each column by summing up only the times occurring in the rows which do not have any $\infty$ symbol in any of the columns.

The smallest total verification time thus obtained, is the one of column Singleton, that is, the one relative to the systems derived by the Singleton operator (3.6 seconds), followed by the times relative to the systems derived by using the All and Chain operators (10.09 and 13.91 seconds, respectively). The largest overall verification time of FASTer is the one relative to the original systems (33.75 seconds).
Similar considerations also apply if we compare the time taken by FASTer on the original systems with that it takes on the systems derived by using Chain, which is the most precise operator (146.83 and 55.89 seconds, respectively).

Thus, the increase of precision due to the constraint-based transformation technique we have proposed, does not determine any significant degradation of the time performance.

6. Related Work and Conclusions

The transformation method presented in this paper is related to program specialization, which is a program transformation technique that, given a program and a specific context of use, derives a specialized program that is more effective in the given context [20]. Indeed, the improvement of precision (and performance) of forward reachability analysis is due to the fact that the transformed systems incorporate information about the specific properties holding in the unsafe states.

The use of specialization techniques for the verification of infinite state systems has been first proposed in the context of constraint logic programming (see, for instance, [12, 14, 24]). In [12, 24] infinite state systems and their properties are encoded as constraint logic programs (following an approach similar to [10, 16]) and program specialization is used as a pre-processing technique for a more effective computation of the least model (or an over-approximation thereof) of those programs. In [14] constraint logic programs provide an intermediate representation of the systems to be verified. However, the final result of the specialization is not a constraint logic program, but a new infinite state system which is then analyzed by using the ALV tool [25].

Unlike [12, 14, 24], in this paper we define a transformation method that works directly on the system specification, without the need of any constraint logic programming representation. Our new approach has the advantage of allowing the use of efficient representation and manipulation techniques for constraints, without the need for providing the representation and the manipulation of clauses, which we, thus, prove to be redundant. We have used our technique as a pre-processing of FASTer [4], instead of ALV, thereby demonstrating that our transformation method works well also in combination with flat acceleration.

Our transformation method is also related to some techniques for abstract interpretation [9] and, in particular, to those proposed in the field of verification of infinite state systems [1, 8]. Indeed, in Section 4 we have introduced several generalized PRE operators by combining various operators previously considered in program analysis and program specialization, such as, widening, convex hull, and well-quasi orders on the integers [9, 12, 13, 24].

The main difference between our transformation technique and abstract interpretation is that, when applied to a given system specification, the former produces an equivalent specification, while the latter produces a more abstract (possibly, finite state) model whose semantics is an approximation of the semantics of the given specification. Moreover, since our transformation method returns a new system specification which is written in the same language of the given specification, after transformation we may also apply abstract interpretation techniques for proving system properties.

A notable feature of our approach is that we use operators that manipulate linear constraints on real numbers, and yet, as already mentioned, we are able to preserve the equivalence of counter systems defined on the integers (obviously, this equivalence is with respect to the class of properties we consider). This feature allows us to apply at transformation time more efficient constraint solvers and more efficient manipulation techniques.

Finally, we would like to note that the transformation algorithm proposed in Section 3 is
independent of the specific interpretation of the constraints (in particular, on the integers or on the reals). Thus, our technique can easily be extended to other classes of reactive systems such as linear hybrid systems [18].

Acknowledgements

This work has been partially supported by PRIN-MIUR and by a joint project between CNR (Italy) and CNRS (France). Thanks to Laurent Fribourg and John Gallagher for many stimulating conversations.

References


