

# Using Real Relaxations During Program Specialization

Fabio Fioravanti<sup>1</sup>, Alberto Pettorossi<sup>2</sup>, Maurizio Proietti<sup>3</sup>, and Valerio Senni<sup>2,4</sup>

<sup>1</sup> Dipartimento di Scienze, University 'G. D'Annunzio', Pescara, Italy  
fioravanti@sci.unich.it

<sup>2</sup> DISP, University of Rome Tor Vergata, Rome, Italy  
{pettorossi,senni}@disp.uniroma2.it

<sup>3</sup> CNR-IASI, Rome, Italy  
maurizio.proietti@iasi.cnr.it

<sup>4</sup> LORIA-INRIA, Villers-les-Nancy, France  
valerio.senni@loria.fr

**Abstract.** We propose a program specialization technique for locally stratified  $\text{CLP}(\mathbb{Z})$  programs, that is, logic programs with linear constraints over the set  $\mathbb{Z}$  of the integer numbers. For reasons of efficiency our technique makes use of a relaxation from integers to reals. We reformulate the familiar unfold/fold transformation rules for CLP programs so that: (i) the applicability conditions of the rules are based on the satisfiability or entailment of constraints over the set  $\mathbb{R}$  of the real numbers, and (ii) every application of the rules transforms a given program into a new program with the same perfect model constructed over  $\mathbb{Z}$ . Then, we introduce a strategy which applies the transformation rules for specializing  $\text{CLP}(\mathbb{Z})$  programs with respect to a given query. Finally, we show that our specialization strategy can be applied for verifying properties of infinite state reactive systems specified by constraints over  $\mathbb{Z}$ .

## 1 Introduction

Reactive systems are often composed of processes that make use of possibly unbounded data structures. In order to specify and reason about this type of systems, several formalisms have been proposed, such as unbounded counter automata [27] and vector addition systems [26]. These formalisms are based on linear constraints over variables ranging over the set  $\mathbb{Z}$  of the integer numbers.

Several tools for the verification of properties of systems with unbounded integer variables have been developed in recent years. Among these we would like to mention ALV [36], FAST [6], LASH [25], and TReX [1]. These tools use sophisticated solvers for constraints over the integers which are based on automata-theoretic techniques [22] or techniques for proving formulas of Presburger Arithmetic [32].

Also constraint logic programming is a very powerful formalism for specifying and reasoning about reactive systems [20]. In fact, many properties of counter

automata and vector addition systems, such as *safety* properties and, more generally, temporal properties, can be easily translated into constraint logic programs with linear constraints over the integers, called CLP( $\mathbb{Z}$ ) programs. [21].

Unfortunately, dealing with constraints over the integers is often a source of inefficiency and, in order to overcome this limitation, many verification techniques are based on the interpretation of the constraints over the set  $\mathbb{R}$  of the real numbers, instead of the set  $\mathbb{Z}$  of the integer numbers [7,13]. This extension of the domain of interpretation is sometimes called *relaxation*.

The relaxation from integers to reals, also called the *real relaxation*, has several advantages: (i) many constraint solving problems (in particular, the satisfiability problem) have lower complexity if considered in the reals, rather than in the integers [33], (ii) the class of linear constraints over the reals is closed under *projection*, which is an operation often used during program verification, while the class of linear constraints over the integers is not, and (iii) many highly optimized libraries are actually available for performing various operations on constraints over the reals, such as satisfiability testing, projection, widening, and convex hull, which are often used in the field of static program analysis [10,11] (see, for instance, the Parma Polyhedral Library [3]).

Relaxation techniques can be viewed as *approximation* techniques. Indeed, if a property holds for all real values of a given variable then it holds for all integer values, but not vice versa. This approximation technique can be applied to the verification of reactive systems. For instance, if a safety property  $\varphi =_{def} \forall x \in \mathbb{R} (reachable(x) \rightarrow safe(x))$  holds, then it also holds when replacing the set  $\mathbb{R}$  by the set  $\mathbb{Z}$ . However, if  $\neg\varphi =_{def} \exists x \in \mathbb{R} (reachable(x) \wedge \neg safe(x))$  holds, then we cannot conclude that  $\exists x \in \mathbb{Z} (reachable(x) \wedge \neg safe(x))$  holds.

Now, as indicated in the literature (see, for instance, [17,19,29,30,31]) the verification of infinite state reactive systems can be done via program specialization and, in particular, in [19] we proposed a technique consisting of the following two steps: (*Step 1*) the specialization of the constraint logic program that encodes the given system, with respect to the query that encodes the property to be verified, and (*Step 2*) the construction of the *perfect model* of the specialized program.

In this paper we propose a variant of the verification technique introduced in [19]. This variant is based on the specialization of locally stratified CLP( $\mathbb{Z}$ ) programs and uses a relaxation from the integers to the reals.

In order to do so, we need: (i) a suitable reformulation of the familiar unfold/fold transformation rules for CLP programs [14,18] so that: (i.1) the applicability conditions of the rules are based on the satisfiability or entailment of constraints over the reals  $\mathbb{R}$ , and (i.2) every application of the rules transforms a given program into a new program with the same perfect model constructed over the integers  $\mathbb{Z}$ , called *perfect  $\mathbb{Z}$ -model*, and then (ii) the introduction of a transformation strategy which applies the reformulated transformation rules for specializing a given CLP( $\mathbb{Z}$ ) program with respect to a given query.

There are two advantages of the verification technique we consider here. The first advantage is that, since our specialization strategy manipulates constraints

over the reals, it may exploit efficient techniques for checking satisfiability and entailment, for computing projection, and for more complex constructions, such as the widening and the convex hull operations over sets of constraints. The second advantage is that, since we use equivalence preserving transformation rules, that is, rules which preserve the perfect  $\mathbb{Z}$ -model, the property to be verified holds in the initial program if and only if it holds in the specialized program and, thus, we may apply to the specialized program any other verification technique we wish, including techniques based on constraints over the integers.

The rest of the paper is structured as follows. In Section 2 we introduce some basic notions concerning constraints and CLP programs. In Section 3 we present the rules for transforming CLP( $\mathbb{Z}$ ) programs and prove that they preserve equivalence with respect to the perfect model semantics. In Section 4 we present our specialization strategy and in Section 5 we show its application to the verification of reactive systems. Finally, in Section 6 we discuss related work in the field of program specialization and verification of infinite state systems.

## 2 Constraint Logic Programs over Integers and Reals

We will consider CLP( $\mathbb{Z}$ ) programs, that is, constraint logic programs with linear constraints over the set  $\mathbb{Z}$  of the integer numbers. An *atomic constraint* is either of the form  $r \geq 0$  or of the form  $r > 0$ , where  $r$  is a linear polynomial with integer coefficients. A *constraint* is a conjunction of atomic constraints. The equality  $t_1 = t_2$  stands for the conjunction  $t_1 \geq t_2 \wedge t_2 \geq t_1$ . A clause of a CLP( $\mathbb{Z}$ ) program is of the form  $A \leftarrow c \wedge B$ , where  $A$  is an atom,  $c$  is a constraint, and  $B$  is a conjunction of (positive or negative) literals. For reasons of simplicity and without loss of generality, we also assume that the arguments of all literals are variables, that is, the literals are of the form  $p(X_1, \dots, X_n)$  or  $\neg p(X_1, \dots, X_n)$ , with  $n \geq 0$ , where  $p$  is a predicate symbol not in  $\{>, \geq, =\}$  and  $X_1, \dots, X_n$  are distinct variables ranging over  $\mathbb{Z}$ .

Given a constraint  $c$ , by  $vars(c)$  we denote the set of variables occurring in  $c$ . By  $\forall(c)$  we denote the universal closure  $\forall X_1 \dots \forall X_n c$ , where  $vars(c) = \{X_1, \dots, X_n\}$ . Similarly, by  $\exists(c)$  we denote the existential closure  $\exists X_1 \dots \exists X_n c$ . Similar notation will also be used for literals, goals, and clauses.

For the constraints over the integers we assume the usual interpretation which, by abuse of language, we denote by  $\mathbb{Z}$ . A  $\mathbb{Z}$ -*model* of a CLP( $\mathbb{Z}$ ) program  $P$  is defined to be a model of  $P$  which agrees with the interpretation  $\mathbb{Z}$  for the constraints. We assume that programs are *locally stratified* [2] and, similarly to the case of logic programs without constraints, for a locally stratified CLP( $\mathbb{Z}$ ) program  $P$  we can define its unique *perfect  $\mathbb{Z}$ -model* (or, simply, *perfect model*), denoted  $M_{\mathbb{Z}}(P)$  (see [2] for the definition of the perfect model of a logic program).

We say that a constraint  $c$  is  $\mathbb{Z}$ -*satisfiable* if  $\mathbb{Z} \models \exists(c)$ . We also say that a constraint  $c$   $\mathbb{Z}$ -*entails* a constraint  $d$ , denoted  $c \sqsubseteq_{\mathbb{Z}} d$ , if  $\mathbb{Z} \models \forall(c \rightarrow d)$ .

Let  $\mathbb{R}$  be the usual interpretation of the constraints over the set of the real numbers. A constraint  $c$  is  $\mathbb{R}$ -*satisfiable* if  $\mathbb{R} \models \exists(c)$ . A constraint  $c$   $\mathbb{R}$ -*entails* a

constraint  $d$ , denoted  $c \sqsubseteq_{\mathbb{R}} d$ , if  $\mathbb{R} \models \forall(c \rightarrow d)$ . The  $\mathbb{R}$ -projection of a constraint  $c$  onto the set  $X$  of variables is a constraint  $c_p$  such that: (i)  $\text{vars}(c_p) \subseteq X$  and (ii)  $\mathbb{R} \models \forall(c_p \leftrightarrow \exists Y_1 \dots \exists Y_k c)$ , where  $\{Y_1, \dots, Y_k\} = \text{vars}(c) - X$ . Recall that the set of constraints over  $\mathbb{Z}$  is not closed under projection.

The following lemma states some simple relationships between  $\mathbb{Z}$ - and  $\mathbb{R}$ -satisfiability, and between  $\mathbb{Z}$ - and  $\mathbb{R}$ -entailment.

**Lemma 1.** *Let  $c$  and  $d$  be constraints and  $X$  be a set of variables.*

- (i) *If  $c$  is  $\mathbb{Z}$ -satisfiable, then  $c$  is  $\mathbb{R}$ -satisfiable.*
- (ii) *If  $c \sqsubseteq_{\mathbb{R}} d$ , then  $c \sqsubseteq_{\mathbb{Z}} d$ .*
- (iii) *If  $c_p$  is the  $\mathbb{R}$ -projection of  $c$  on  $X$ , then  $c \sqsubseteq_{\mathbb{Z}} c_p$ .*

### 3 Transformation Rules with Real Relaxations

In this section we present a set of transformation rules that can be used for specializing locally stratified CLP( $\mathbb{Z}$ ) programs. The applicability conditions of the rules are given in terms of constraints interpreted over the set  $\mathbb{R}$  and, as shown by Theorem 1, these rules preserve the perfect  $\mathbb{Z}$ -model semantics.

The rules we will consider are those needed for specializing constraint logic programs, as indicated in the Specialization Strategy of Section 4. Note, however, that the correctness result stated in Theorem 1 can be extended to a larger set of rules (including the *negative unfolding* rule [18,34]) or to more powerful rules (such as the *definition* rule with  $m (\geq 1)$  clauses, and the *multiple positive folding* [18]).

Before presenting these rules, we would like to show through an example that, if we consider different domains for the interpretation of the constraints and, in particular, if we apply the relaxation from the integers to the reals, we may derive different programs with different intended semantics.

Let us consider, for instance, the following constraint logic program  $P$ :

- 1.  $p \leftarrow Y > 0 \wedge Y < 1$
- 2.  $q \leftarrow$

If we interpret the constraints over the reals, since  $\mathbb{R} \models \exists Y(Y > 0 \wedge Y < 1)$ , program  $P$  can be transformed into program  $P_{\mathbb{R}}$ :

- 1'.  $p \leftarrow$
- 2.  $q \leftarrow$

If we interpret the constraints over the integers, since  $\mathbb{Z} \models \neg \exists Y(Y > 0 \wedge Y < 1)$ , program  $P$  can be transformed into program  $P_{\mathbb{Z}}$ :

- 2.  $q \leftarrow$

Programs  $P_{\mathbb{R}}$  and  $P_{\mathbb{Z}}$  are not equivalent because they have different perfect  $\mathbb{Z}$ -models (which in this case coincide with their least Herbrand models). Thus, when we apply a relaxation we should proceed with some care. In particular, we will admit a transformation rule only when its applicability conditions interpreted over  $\mathbb{R}$  imply the corresponding applicability conditions interpreted over  $\mathbb{Z}$ .

The transformation rules are used to construct a *transformation sequence*, that is, a sequence  $P_0, \dots, P_n$  of programs. We assume that  $P_0$  is locally stratified. A transformation sequence  $P_0, \dots, P_n$  is constructed as follows. Suppose

that we have constructed a transformation sequence  $P_0, \dots, P_k$ , for  $0 \leq k \leq n-1$ . The next program  $P_{k+1}$  in the transformation sequence is derived from program  $P_k$  by the application of a transformation rule among R1–R5 defined below.

Our first rule is the *Constrained Atomic Definition* rule (or *Definition Rule*, for short), which is applied for introducing a new predicate definition.

**R1. Constrained Atomic Definition.** Let us consider a clause, called a *definition clause*, of the form:

$$\delta: \text{newp}(X_1, \dots, X_h) \leftarrow c \wedge p(X_1, \dots, X_h)$$

where: (i) *newp* does not occur in  $\{P_0, \dots, P_k\}$ , (ii)  $X_1, \dots, X_h$  are distinct variables, (iii)  $c$  is a constraint with  $\text{vars}(c) \subseteq \{X_1, \dots, X_h\}$ , and (iv)  $p$  occurs in  $P_0$ . By *constrained atomic definition* from program  $P_k$  we derive the program  $P_{k+1} = P_k \cup \{\delta\}$ . For  $k \geq 0$ ,  $\text{Defs}_k$  denotes the set of clauses introduced by the definition rule during the transformation sequence  $P_0, \dots, P_k$ . In particular,  $\text{Defs}_0 = \emptyset$ .

**R2. (Positive) Unfolding.** Let  $\gamma: H \leftarrow c \wedge G_L \wedge A \wedge G_R$  be a clause in program  $P_k$  and let

$$\gamma_1: K_1 \leftarrow c_1 \wedge B_1 \quad \dots \quad \gamma_m: K_m \leftarrow c_m \wedge B_m \quad (m \geq 0)$$

be all clauses of (a renamed apart variant of) program  $P_k$  such that, for  $i=1, \dots, m$ , the constraint  $c \wedge c_i \rho_i$  is  $\mathbb{R}$ -satisfiable, where  $\rho_i$  is a renaming substitution such that  $A = K_i \rho_i$  (recall that all atoms in a  $\text{CLP}(\mathbb{Z})$  program have distinct variables as arguments).

By *unfolding clause  $\gamma$  w.r.t. the atom  $A$*  we derive the clauses

$$\eta_1: H \leftarrow c \wedge c_1 \rho_1 \wedge G_L \wedge B_1 \rho_1 \wedge G_R$$

...

$$\eta_m: H \leftarrow c \wedge c_m \rho_m \wedge G_L \wedge B_m \rho_m \wedge G_R$$

and from program  $P_k$  we derive the program  $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta_1, \dots, \eta_m\}$ .

Note that if  $m=0$  then, by unfolding, clause  $\gamma$  is deleted from  $P_k$ .

*Example 1.* Let  $P_k$  be the following  $\text{CLP}(\mathbb{Z})$  program:

1.  $p(X) \leftarrow X > 1 \wedge q(X)$
2.  $q(Y) \leftarrow Y > 2 \wedge Z = Y - 1 \wedge q(Z)$
3.  $q(Y) \leftarrow Y < 2 \wedge 5Z = Y \wedge q(Z)$
4.  $q(Y) \leftarrow Y = 0$

Let us unfold clause 1 w.r.t. the atom  $q(X)$ . We have the renaming substitution  $\rho = \{Y/X\}$ , which unifies the atoms  $q(X)$  and  $q(Y)$ , and the following three constraints:

- (a)  $X > 1 \wedge X > 2 \wedge Z = X - 1$ , derived from clauses 1 and 2,
- (b)  $X > 1 \wedge X < 2 \wedge 5Z = X$ , derived from clauses 1 and 3,
- (c)  $X > 1 \wedge X = 0$ , derived from clauses 1 and 4.

Only (a) and (b) are  $\mathbb{R}$ -satisfiable, and only (a) is  $\mathbb{Z}$ -satisfiable. By unfolding clause 1 w.r.t.  $q(X)$  we derive the following clauses:

- 1.a  $p(X) \leftarrow X > 1 \wedge X > 2 \wedge Z = X - 1 \wedge q(Z)$
- 1.b  $p(X) \leftarrow X > 1 \wedge X < 2 \wedge 5Z = X \wedge q(Z)$

Now we introduce two versions of the folding rule: *positive folding* and *negative folding*, depending on whether folding is applied to positive or negative literals in the body of a clause.

**R3. Positive Folding.** Let  $\gamma: H \leftarrow c \wedge G_L \wedge A \wedge G_R$  be a clause in  $P_k$  and let  $\delta: K \leftarrow d \wedge B$  be a clause in (a renamed apart variant of)  $Defs_k$ . Suppose that there exists a renaming substitution  $\rho$  such that: (i)  $A = B\rho$ , and (ii)  $c \sqsubseteq_{\mathbb{R}} d\rho$ . By *folding  $\gamma$  using  $\delta$*  we derive the clause  $\eta: H \leftarrow c \wedge G_L \wedge K\rho \wedge G_R$  and from program  $P_k$  we derive the program  $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$ .

The following example illustrates an application of Rule R3.

*Example 2.* Suppose that the following clause belongs to  $P_k$ :

$$\gamma: h(X) \leftarrow X \geq 1 \wedge 2Y = 3X + 2 \wedge p(X, Y)$$

and suppose that the following clause is a definition clause in  $Defs_k$ :

$$\delta: new(V, Z) \leftarrow Z > 2 \wedge p(V, Z)$$

We have that the substitution  $\rho = \{V/X, Z/Y\}$  satisfies Conditions (i) and (ii) of the positive folding rule because  $X \geq 1 \wedge 2Y = 3X + 2 \sqsubseteq_{\mathbb{R}} (Z > 2)\rho$ . Thus, by folding clause  $\gamma$  using clause  $\delta$ , we derive:

$$\eta: h(X) \leftarrow X \geq 1 \wedge 2Y = 3X + 2 \wedge new(X, Y)$$

**R4. Negative Folding.** Let  $\gamma: H \leftarrow c \wedge G_L \wedge \neg A \wedge G_R$  be a clause in  $P_k$  and let  $\delta: K \leftarrow d \wedge B$  be a clause in (a renamed apart variant of)  $Defs_k$ . Suppose that there exists a renaming substitution  $\rho$  such that: (i)  $A = B\rho$ , and (ii)  $c \sqsubseteq_{\mathbb{R}} d\rho$ . By *folding  $\gamma$  using  $\delta$*  we derive the clause  $\eta: H \leftarrow c \wedge G_L \wedge \neg K\rho \wedge G_R$  and from program  $P_k$  we derive the program  $P_{k+1} = (P_k - \{\gamma\}) \cup \{\eta\}$ .

The following notion will be used for introducing the *clause removal* rule. Given two clauses of the form  $\gamma: H \leftarrow c \wedge B$  and  $\delta: H \leftarrow d$ , respectively, we say that  $\gamma$  is  $\mathbb{Z}$ -*subsumed* by  $\delta$ , if  $c \sqsubseteq_{\mathbb{Z}} d$ . Similarly, we say that  $\gamma$  is  $\mathbb{R}$ -*subsumed* by  $\delta$ , if  $c \sqsubseteq_{\mathbb{R}} d$ .

By Lemma 1, if  $\gamma$  is  $\mathbb{R}$ -subsumed by  $\delta$ , then  $\gamma$  is  $\mathbb{Z}$ -subsumed by  $\delta$ .

**R5. Clause Removal.** Let  $\gamma$  be a clause in  $P_k$ . By *clause removal* we derive the program  $P_{k+1} = P_k - \{\gamma\}$  if clause  $\gamma$  is  $\mathbb{R}$ -subsumed by a clause occurring in  $P_k - \{\gamma\}$ .

The following Theorem 1 states that the transformation rules R1–R5 preserve the perfect  $\mathbb{Z}$ -model semantics.

**Theorem 1 (Correctness of the Transformation Rules).** *Let  $P_0$  be a locally stratified program and let  $P_0, \dots, P_n$  be a transformation sequence obtained by applying rules R1–R5. Let us assume that for every  $k$ , with  $0 < k < n - 1$ , if  $P_{k+1}$  is derived by applying positive folding to a clause in  $P_k$  using a clause  $\delta$  in  $Defs_k$ , then there exists  $j$ , with  $0 < j < n - 1$ , such that: (i)  $\delta$  belongs to  $P_j$ , and (ii)  $P_{j+1}$  is derived by unfolding  $\delta$  w.r.t. the only atom in its body. Then  $P_n$  is locally stratified and for every ground atom  $A$  whose predicate occurs in  $P_0$ , we have that  $A \in M_{\mathbb{Z}}(P_0)$  iff  $A \in M_{\mathbb{Z}}(P_n)$ .*

*Proof.* (Sketch) Let us consider variants of Rules R1–R5 where the applicability conditions are obtained from those for R1–R5 by replacing  $\mathbb{R}$  by  $\mathbb{Z}$ . Let us denote R1 $_{\mathbb{Z}}$ –R5 $_{\mathbb{Z}}$  these variants of the rules. Rules R1 $_{\mathbb{Z}}$ , R2 $_{\mathbb{Z}}$ , R3 $_{\mathbb{Z}}$ , R4 $_{\mathbb{Z}}$ , and R5 $_{\mathbb{Z}}$  can be viewed as instances (for  $\mathcal{D} = \mathbb{Z}$ ) of the rules R1, R2p, R3(P), R3(N), and R4s, respectively, for specializing CLP( $\mathcal{D}$ ) programs presented in [15]. By Theorem 3.3.10 of [15] we have that  $P_n$  is locally stratified and for every ground atom  $A$  whose predicate occurs in  $P_0$ , we have that  $A \in M_{\mathbb{Z}}(P_0)$  iff  $A \in M_{\mathbb{Z}}(P_n)$ . Since, by Lemma 1 we have that the applicability conditions of R1–R5 imply the applicability conditions of R1 $_{\mathbb{Z}}$ –R5 $_{\mathbb{Z}}$ , we get the thesis.  $\square$

## 4 The Specialization Strategy

Now we present a strategy for specializing a program  $P_0$  with respect to a query of the form  $c \wedge p(X_1, \dots, X_h)$ , where  $c$  is a constraint and  $p$  is a predicate occurring in  $P_0$ . Our strategy constructs a transformation sequence  $P_0, \dots, P_n$  by using the rules R1–R5 defined in Section 3. The last program  $P_n$  is the specialized version of  $P_0$  with respect to  $c \wedge p(X_1, \dots, X_h)$ .  $P_n$  is the output program  $P_{sp}$  of the specialization strategy below.

The Specialization Strategy makes use of two auxiliary operators: an *unfolding operator* and a *generalization operator* that tell us how to apply the unfolding rule R2 and the constrained atomic definition rule R1, respectively. The problem of designing suitable unfolding and generalization operators has been addressed in many papers and various solutions have been proposed in the literature (see, for instance, [16,19,31] and [28] for a survey in the case of logic programs). In this paper we will not focus on this aspect and we will simply assume that we are given: (i) an operator  $Unfold(\delta, P)$  which, for every clause  $\delta$  occurring in a program  $P$ , returns a set of clauses derived from  $\delta$  by applying  $n$  ( $\geq 1$ ) times the unfolding rule R2, and (ii) an operator  $Gen(c \wedge A, Defs)$  which, for every constraint  $c$ , atom  $A$  with  $vars(c) \subseteq vars(A)$ , and set  $Defs$  of the definition clauses introduced so far by Rule R1 during the Specialization Strategy, returns a constraint  $g$  that is *more general than*  $c$ , that is: (i)  $vars(g) \subseteq vars(c)$  and (ii)  $c \sqsubseteq_{\mathbb{R}} g$ . An example of the generalization operator  $Gen$  will be presented in Section 5.

---

### The Specialization Strategy

*Input:* A program  $P_0$  and a query  $c \wedge p(X_1, \dots, X_h)$  where: (i)  $c$  is a constraint with  $vars(c) \subseteq \{X_1, \dots, X_h\}$ , and (ii)  $p$  occurs in  $P_0$ .

*Output:* A program  $P_{sp}$  such that for every tuple  $\langle n_1, \dots, n_h \rangle \in \mathbb{Z}^h$ ,

$$p_{sp}(n_1, \dots, n_h) \in M_{\mathbb{Z}}(P_0 \cup \{\delta_0\}) \text{ iff } p_{sp}(n_1, \dots, n_h) \in M_{\mathbb{Z}}(P_{sp}),$$

where: (i)  $\delta_0$  is the definition clause  $p_{sp}(X_1, \dots, X_h) \leftarrow c \wedge p(X_1, \dots, X_h)$  and (ii)  $p_{sp}$  is a predicate not occurring in  $P_0$ .

INITIALIZATION:

$$P_{sp} := P_0 \cup \{\delta_0\}; \text{ Defs} := \{\delta_0\};$$

*while* there exists a clause  $\delta$  in  $P_{sp} \cap Defs$  *do*

UNFOLDING:  $\Gamma := Unfold(\delta, P_{sp})$ ;

CLAUSE REMOVAL:

*while* in  $\Gamma$  there exist two distinct clauses  $\gamma_1$  and  $\gamma_2$  such that  $\gamma_1$  is  $\mathbb{R}$ -subsumed by  $\gamma_2$  *do*  $\Gamma := \Gamma - \{\gamma_1\}$  *end-while*;

DEFINITION & FOLDING:

*while* in  $\Gamma$  there exists a clause  $\gamma: H \leftarrow c \wedge G_1 \wedge L \wedge G_2$ , where  $L$  is a literal whose predicate occurs in  $P_0$  *do*

let  $c_p$  be the  $\mathbb{R}$ -projection of  $c$  on  $vars(L)$  and let  $A$  be the atom such that  $L$  is either  $A$  or  $\neg A$ ;

*if* in  $Defs$  there exists a clause  $K \leftarrow d \wedge B$  and a renaming substitution  $\rho$  such that: (i)  $A = B\rho$  and (ii)  $c_p \sqsubseteq_{\mathbb{R}} d\rho$

*then*  $\Gamma := (\Gamma - \{\gamma\}) \cup \{H \leftarrow c \wedge G_1 \wedge M \wedge G_2\}$   
where  $M$  is  $K\rho$  if  $L$  is  $A$ , and  $M$  is  $\neg K\rho$  if  $L$  is  $\neg A$ ;

*else*  $P_{sp} := P_{sp} \cup \{K \leftarrow g \wedge A\}$ ;  $Defs := Defs \cup \{K \leftarrow g \wedge A\}$   
where: (i)  $K = newp(Y_1, \dots, Y_m)$ , (ii)  $newp$  is a predicate symbol not occurring in  $P_0 \cup Defs$ , (iii)  $\{Y_1, \dots, Y_m\} = vars(A)$ , and  
(iv)  $g = Gen(c_p \wedge A, Defs)$ ;

$\Gamma := (\Gamma - \{\gamma\}) \cup \{H \leftarrow c \wedge G_1 \wedge M \wedge G_2\}$   
where  $M$  is  $K$  if  $L$  is  $A$ , and  $M$  is  $\neg K$  if  $L$  is  $\neg A$ ;

*end-while*;

$P_{sp} := (P_{sp} - \{\delta\}) \cup \Gamma$ ;

*end-while*

In the Specialization Strategy we use real relaxations at several points: (i) when we apply the *Unfold* operator (because for applying rule R2 we check  $\mathbb{R}$ -satisfiability of constraints); (ii) when we check  $\mathbb{R}$ -subsumption during clause removal; (iii) when we compute the  $\mathbb{R}$ -projection  $c_p$  of the constraint  $c$  occurring in a clause  $\gamma$  of the form  $H \leftarrow c \wedge G_1 \wedge L \wedge G_2$ , (iv) when we check whether or not  $c_p \sqsubseteq_{\mathbb{R}} d\rho$ , and (v) when we compute the constraint  $g = Gen(c_p \wedge A, Defs)$  such that  $c_p \sqsubseteq_{\mathbb{R}} g$ . (Note that the condition  $c_p \sqsubseteq_{\mathbb{R}} g$  ensures that clause  $\gamma$  can be folded using the new clause  $K \leftarrow g \wedge A$ , as it can be checked by looking at Rules R3 and R4 and recalling that, by Lemma 1,  $c \sqsubseteq_{\mathbb{R}} c_p$ .)

The correctness of the Specialization Strategy derives from the correctness of the transformation rules (see Theorem 1). Indeed, the sequence of values assigned to  $P_{sp}$  during the strategy can be viewed as (a subsequence of) a transformation sequence satisfying the hypotheses of Theorem 1.

We assume that the unfolding and the generalization operators guarantee that the Specialization Strategy terminates. In particular, we assume that: (i) the *Unfold* operator performs a finite number of unfolding steps, and (ii) the set *Defs* stabilizes after a finite number of applications of the *Gen* operator, that is, there exist two consecutive values, say  $Defs_k$  and  $Defs_{k+1}$ , of *Defs* such that  $Defs_k = Defs_{k+1}$ . This stabilization property can be enforced by defining the

generalization operator similarly to the *widening* operator on polyhedra, which is often used in static analysis of programs [10].

## 5 Application to the Verification of Reactive Systems

In this section we show how our Specialization Strategy based on real relaxations can be used for the verification of properties of infinite state reactive systems.

Suppose that we are given an infinite state reactive system such that: (i) the set of *states* is a subset of  $\mathbb{Z}^k$ , and (ii) the state *transition relation* is a binary relation on  $\mathbb{Z}^k$  specified as a set of constraints over  $\mathbb{Z}^k \times \mathbb{Z}^k$ . In this section we will take into consideration *safety* properties, but our technique can be applied to more complex properties, such as CTL temporal properties [9,19]. A reactive system is said to be *safe* if from every *initial state* it is not possible to reach, by zero or more applications of the transition relation, a state, called an *unsafe state*, satisfying an undesired property. Let *Unsafe* be the set of all unsafe states. A standard method to verify whether or not the system is safe consists in: (i) computing (backward from *Unsafe*) the set *BR* of the states from which it is possible to reach an unsafe state, and (ii) checking whether or not  $BR \cap \text{Init} = \emptyset$ , where *Init* denotes the set of initial states.

In order to compute the set *BR* of backward reachable states, we introduce a CLP( $\mathbb{Z}$ ) program  $P_{BR}$  defining a predicate *br* such that  $\langle n_1, \dots, n_k \rangle \in BR$  iff  $br(n_1, \dots, n_k) \in M_{\mathbb{Z}}(P_{BR})$ . Then we can show that the reactive system is safe, by showing that there is no atom  $br(n_1, \dots, n_k) \in M_{\mathbb{Z}}(P_{BR})$  such that  $init(n_1, \dots, n_k)$  holds, where  $init(X_1, \dots, X_k)$  is a constraint that represents the set *Init* of states. Unfortunately, the computation of the perfect  $\mathbb{Z}$ -model  $M_{\mathbb{Z}}(P_{BR})$  by a bottom-up evaluation of the immediate consequence operator, may not terminate and in that case we are unable to check whether or not the system is safe.

It has been shown in [19] that the termination of the bottom-up construction of the perfect model of a program can be improved by first specializing the program with respect to the query of interest. In this paper, we use a variant of the specialization-based method presented in [19] which is tailored to the verification of safety properties.

Our specialization-based method for verification consists of two steps. In Step 1 we apply the Specialization Strategy of Section 4 and specialize program  $P_{BR}$  with respect to the initial states of the system, that is, w.r.t. the query  $init(X_1, \dots, X_k) \wedge br(X_1, \dots, X_k)$ . In Step 2 we compute the perfect  $\mathbb{Z}$ -model of the specialized program by a bottom-up evaluation of the immediate consequence operator associated with the program.

Before presenting an example of application of our verification method, let us introduce the generalization operator we will use in the Specialization Strategy. We will define our generalization operator by using the widening operator [10], but we could have made other choices by using suitable combinations of the widening operator, the *convex hull* operator, and *thin well-quasi orderings* based on the coefficients of the polynomials (see [11,19,31] for details).

First, we need to structure the set  $Defs$  of definition clauses as a tree, also called  $Defs$  (a similar approach is followed in [19]): (i) the root clause of that tree is  $\delta_0$ , and (ii) the children of a definition clause  $\delta$  are the new definition clauses added to  $Defs$  (see the *else* branch in the body of the inner while-loop of the Specialization Strategy) during the execution relative to  $\delta$  (see the test ' $\delta$  in  $P_{sp} \cap Defs$ ') of the body of outer while-loop of the Specialization Strategy.

Given a constraint  $c_p$  and an atom  $A$  obtained from a clause  $\delta$  as described in the Specialization Strategy,  $Gen(c_p \wedge A, Defs)$  is the constraint  $g$  defined as follows. If in  $Defs$  there exists a (most recent) ancestor clause  $K \leftarrow d \wedge B$  of  $\delta$  (possibly  $\delta$  itself) such that: (i)  $A = B\rho$  for some renaming substitution  $\rho$ , and (ii)  $d\rho = a_1 \wedge \dots \wedge a_m$  then  $g = \bigwedge_{i=1}^m \{a_i \mid c_p \sqsubseteq_{\mathbb{R}} a_i\}$ . Otherwise, if no such ancestor of  $\delta$  exists in  $Defs$ , then  $g = c_p$ .

Now let us present an example of application of our verification technique based on the Specialization Strategy of Section 4. The states of the infinite state reactive system we consider are pairs of integers and the transitions from states to states, denoted by  $\longrightarrow$ , are the following ones: for all  $X, Y \in \mathbb{Z}$ ,

- (1)  $\langle X, Y \rangle \longrightarrow \langle X, Y-1 \rangle$  if  $X \geq 1$
- (2)  $\langle X, Y \rangle \longrightarrow \langle X, Y+2 \rangle$  if  $X \leq 2$
- (3)  $\langle X, Y \rangle \longrightarrow \langle X, -1 \rangle$  if  $\exists Z \in \mathbb{Z} (Y = 2Z + 1)$

(Thus, transition (3) is applicable only if  $Y$  is a positive or negative odd number.) The initial state is  $\langle 0, 0 \rangle$  and, thus,  $Init$  is the singleton  $\{\langle 0, 0 \rangle\}$ . We want to prove that the system is safe in the sense that from the initial state we cannot reach any state  $\langle X, Y \rangle$  with  $Y < 0$ . As mentioned above, we define the set  $BR = \{\langle m, n \rangle \in \mathbb{Z}^2 \mid \exists \langle x, y \rangle \in \mathbb{Z}^2 (\langle m, n \rangle \longrightarrow^* \langle x, y \rangle \wedge y < 0)\}$ , where  $\longrightarrow^*$  is the reflexive, transitive closure of the transition relation  $\longrightarrow$ . Thus,  $BR$  is the set of states from which an unsafe state is reachable. We have to prove that  $Init \cap BR = \emptyset$ .

We proceed as follows. First, we introduce the following program  $P_{BR}$ :

1.  $br(X, Y) \leftarrow X \geq 1 \wedge X' = X \wedge Y' = Y - 1 \wedge br(X', Y')$
2.  $br(X, Y) \leftarrow X \leq 2 \wedge X' = X \wedge Y' = Y + 2 \wedge br(X', Y')$
3.  $br(X, Y) \leftarrow Y = 2Z + 1 \wedge X' = X \wedge Y' = -1 \wedge br(X', Y')$
4.  $br(X, Y) \leftarrow Y < 0$

The predicate  $br$  computes the set  $BR$  of states, in the sense that: for all  $\langle m, n \rangle \in \mathbb{Z}^2$ ,  $\langle m, n \rangle \in BR$  iff  $br(m, n) \in M_{\mathbb{Z}}(P_{BR})$ . Thus, in order to prove the safety of the system it is enough to show that  $br(0, 0) \notin M_{\mathbb{Z}}(P_{BR})$ . Unfortunately, the construction of  $M_{\mathbb{Z}}(P_{BR})$  performed by means of the bottom-up evaluation of the immediate consequence operator does not terminate.

Note that the use of a *tabled* logic programming system [8], augmented with a solver for constraints on the integers, would not overcome this difficulty. Indeed, a top-down evaluation of the query  $br(0, 0)$  generates infinitely many calls of the form  $br(0, 2n)$ , for  $n \geq 1$ .

Now we show that our two step verification method successfully terminates. *Step 1.* We apply the Specialization Strategy which takes as input the program  $P_{BR}$  and the query  $X = 0 \wedge Y = 0 \wedge br(X, Y)$ . Thus, the clause  $\delta_0$  is:

$$\delta_0. \text{ br}_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge \text{br}(X, Y)$$

By applying the *Unfold* operator we obtain the two clauses:

5.  $\text{br}_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge X'=0 \wedge Y'=2 \wedge \text{br}(X', Y')$
6.  $\text{br}_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge \text{br}(X', Y')$

Since clause  $\delta_0$  cannot be used for folding clause 5, we apply the generalization operator and we compute  $\text{Gen}((X'=0 \wedge Y'=2 \wedge \text{br}(X', Y')), \{\delta_0\})$  as follows. We consider the definition clause  $\delta_0$  to be an ancestor clause of itself, and we generalize the constraint  $d\rho \equiv (X' \geq 0 \wedge X' \leq 0 \wedge Y' \geq 0 \wedge Y' \leq 0)$ , using  $c_p \equiv (X'=0 \wedge Y'=2)$ , thereby introducing the following definition (modulo variable renaming):

$$\delta_1. \text{ new1}(X, Y) \leftarrow X=0 \wedge Y \geq 0 \wedge \text{br}(X, Y)$$

Similarly, in order to fold clause 6, we introduce the following definition:

$$\delta_2. \text{ new2}(X, Y) \leftarrow X=0 \wedge Y \leq 0 \wedge \text{br}(X, Y)$$

By folding clauses 5 and 6 by using definitions  $\delta_1$  and  $\delta_2$ , respectively, we derive the following clauses:

7.  $\text{br}_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge X'=0 \wedge Y'=2 \wedge \text{new1}(X', Y')$
8.  $\text{br}_{sp}(X, Y) \leftarrow X=0 \wedge Y=0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge \text{new2}(X', Y')$

Then, we proceed with the next iterations of the body of the outermost while-loop of the Specialization Strategy, and we process first clause  $\delta_1$  and then clause  $\delta_2$ . By using clauses  $\delta_0$ ,  $\delta_1$ , and  $\delta_2$ , we cannot fold all the clauses which are obtained by unfolding  $\delta_1$  and  $\delta_2$  w.r.t. the atom  $\text{br}(X, Y)$ . Thus, we again apply the generalization operator and we introduce the following definition (modulo variable renaming):

$$\delta_3. \text{ new3}(X, Y) \leftarrow X=0 \wedge \text{br}(X, Y)$$

After processing also this clause  $\delta_3$  and performing the unfolding and folding steps as indicated by the Specialization Strategy, we obtain the clauses:

9.  $\text{new1}(X, Y) \leftarrow X=0 \wedge Y \geq 0 \wedge X'=0 \wedge Y'=Y+2 \wedge \text{new1}(X', Y')$
10.  $\text{new1}(X, Y) \leftarrow X=0 \wedge Y \geq 0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge \text{new2}(X', Y')$
11.  $\text{new2}(X, Y) \leftarrow X=0 \wedge Y \leq 0 \wedge X'=0 \wedge Y'=Y+2 \wedge \text{new3}(X', Y')$
12.  $\text{new2}(X, Y) \leftarrow X=0 \wedge Y \leq 0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge \text{new2}(X', Y')$
13.  $\text{new2}(X, Y) \leftarrow X=0 \wedge Y < 0$
14.  $\text{new3}(X, Y) \leftarrow X=0 \wedge X'=0 \wedge Y'=Y+2 \wedge \text{new3}(X', Y')$
15.  $\text{new3}(X, Y) \leftarrow X=0 \wedge Y=2Z+1 \wedge X'=0 \wedge Y'=-1 \wedge \text{new3}(X', Y')$
16.  $\text{new3}(X, Y) \leftarrow X=0 \wedge Y < 0$

The final program  $P_{sp}$  consists of clauses 7–16.

*Step 2.* Now we construct the perfect  $\mathbb{Z}$ -model of  $P_{sp}$  by computing the least fixpoint of the immediate consequence operator associated with  $P_{sp}$  (note that in our case the least fixpoint exists, because the program is definite, and is reached after a finite number of iterations) and we have that:

$$M_{\mathbb{Z}}(P_{sp}) = \{new1(X, Y) \mid X=0 \wedge Y \geq 0 \wedge Y = 2Z+1\} \cup \\ \{new2(X, Y) \mid X=0 \wedge (Y < 0 \vee Y = 2Z+1)\} \cup \\ \{new3(X, Y) \mid X=0 \wedge (Y < 0 \vee Y = 2Z+1)\}.$$

By inspection, we immediately get that  $br_{sp}(0, 0) \notin M_{\mathbb{Z}}(P_{sp})$  and, thus, the safety property has been proved.

Our Specialization Strategy has been implemented on the MAP transformation system (available at <http://www.iasi.cnr.it/~proietti/system.html>) by suitably modifying the specialization strategy presented in [19], so as to use the transformation rules based on real relaxations we have presented in this paper. We have tested our implementation on the set of infinite state systems used for the experimental evaluation in [19] and we managed to prove the same properties. However, the technique proposed in [19] encodes the temporal properties of the reactive systems we consider as CLP( $\mathbb{Q}$ ) programs, where  $\mathbb{Q}$  is the set of rational numbers. Thus, a proof of correctness of the encoding is needed for each system, to show that the properties of interest hold in the CLP( $\mathbb{Q}$ ) encoding iff they hold in the CLP( $\mathbb{Z}$ ) one. In contrast, the method presented in this paper makes use of constraint solvers over the real numbers, but it preserves *equivalence* with respect to the perfect  $\mathbb{Z}$ -model, thereby avoiding the need for *ad hoc* proofs of the correctness of the encoding.

Finally, note that the example presented in this section cannot be worked out by first applying the relaxation from integers to reals to the initial program and then applying polyhedral approximations, such as those considered in static program analysis [10]. Indeed, we have that  $br(0, 0) \notin M_{\mathbb{Z}}(P_{BR})$ , but if the system is interpreted over the reals, instead of the integers, we have that  $br(0, 0) \in M_{\mathbb{R}}(P_{BR})$  (where  $M_{\mathbb{R}}$  denotes the perfect model constructed over  $\mathbb{R}$ ). This is due to the fact that  $\exists Z(0=2Z+1)$  holds on the reals (but it does not hold on the integers) and, hence, we derive  $br(0, 0)$  from clauses 3 and 4 of program  $P_{BR}$ . Thus,  $br(0, 0)$  is a member of every over-approximation of  $M_{\mathbb{R}}(P_{BR})$  and the safety property cannot be proved.

## 6 Related Work and Conclusions

We have presented a technique for specializing a CLP( $\mathbb{Z}$ ) program with respect to a query of interest. Our technique is based on the unfold/fold transformation rules and its main novelty is that it makes use of the relaxation from the integers to the reals, that is, during specialization the constraints are interpreted over the set  $\mathbb{R}$  of the real numbers, instead of  $\mathbb{Z}$ . The most interesting feature of our specialization technique is that, despite the relaxation, the initial program and the derived, specialized program are equivalent with respect to the perfect model constructed over  $\mathbb{Z}$  (restricted to the query of interest). In essence, the reason for this equivalence is that, if the unsatisfiability or entailment between constraints that are present in the applicability conditions of the transformation rules hold in  $\mathbb{R}$ , then they hold also in  $\mathbb{Z}$ .

The main practical advantage of our specialization technique is that, during transformation, we can use tools for manipulating constraints over the reals, such

as the libraries for constraint solving, usually available within  $\text{CLP}(\mathbb{R})$  systems and, in particular, the Parma Polyhedral Library [3]. These tools are significantly more efficient than constraint solvers over the integers and, moreover, they implement operators which are often used during program specialization and program analysis, such as, the widening and convex hull operators. The price we pay, at least in principle, for the efficiency improvement, is that the result of program specialization may be sub-optimal with respect to the one which can be achieved by manipulating integer constraints. Indeed, our specialization strategy might fail to exploit properties which hold for the integers and not for the reals, while transforming the input program. For example, it may be unable to detect that a clause could be removed because it contains constraints which are unsatisfiable on the integers. However, we have checked that, for the significant set of examples taken from [19], this sub-optimality never occurs.

The main application of our specialization technique is the verification of infinite state reactive systems by following the approach presented in [19]. Those systems are often specified by using constraints over integer variables, and their properties (for instance, reachability, safety and liveness) can be specified by using  $\text{CLP}(\mathbb{Z})$  programs [20,21]. It has been shown in [19] that properties of infinite state reactive systems can be verified by first (1) specializing the program that encodes the properties of the system with respect to the property of interest, and then (2) constructing the perfect model of the specialized program by the standard bottom-up procedure based on the evaluation of the immediate consequence operator. However, in [19] the reactive systems and their properties were encoded by using CLP programs over the rational numbers (or, equivalently for linear constraints, real numbers), instead of integer numbers. Thus, a proof of correctness of the encoding is needed for each system (or for classes of systems, as in [7,13]). In contrast, our specialization technique makes use of constraint solvers over the real numbers, but preserves equivalence with respect to the perfect model constructed over the integer numbers, thereby avoiding the need for *ad hoc* proofs of the correctness of the encoding.

Specialization techniques for constraint logic programs have been presented in several papers [12,16,23,31,35]. However, those techniques consider  $\text{CLP}(\mathcal{D})$  programs, where  $\mathcal{D}$  is either a generic domain or the domain of the rational numbers or the domain of the real numbers. None of those papers proposes techniques for specializing  $\text{CLP}(\mathbb{Z})$  programs by manipulating constraints interpreted over the real numbers, as we do here.

Also the application of program specialization to the verification of infinite state systems is not a novel idea [17,19,29,30,31] and, indeed, the technique outlined in Section 5 is a variant of the one proposed in [19]. The partial deduction techniques presented in [29,30] do not make use of constraints. The papers [17,19,31] propose verification techniques for reactive systems which are based on the specialization of constraint logic programs, where the constraints are linear equations and inequations over the rational or real numbers. When applying these specialization techniques to reactive systems whose native specifications are given by using constraints over the integers, we need to prove the

correctness of the encoding. Indeed, as shown by our example in Section 3, if we specify a system by using constraints over the integers and then we interpret those constraints over the reals (or the rationals), we may get an incorrect result. The approach investigated in this paper avoids extra correctness proofs, and allows us to do the specialization by interpreting constraints over the reals.

The verification of program properties based on real convex polyhedral approximations (that is, linear inequations over the reals) has been first proposed in the field of static program analysis [10,11] and then applied in many contexts. In particular, [4,5,13] consider  $\text{CLP}(\mathbb{R})$  encodings of infinite state reactive systems. In the case where a reactive system is specified by constraints over the integers and we want to prove a property of a set of reachable states, these encodings determine an (over-)approximation of that set. Thus, by static analysis a further (over-)approximation is computed, besides the one due the interpretation over the reals, instead of the integers, and the property of interest is checked on the approximated set of reachable states. (Clearly this method can only be applied to prove that certain states are *not* reachable.)

A relevant difference between our approach and the program analysis techniques based on polyhedral approximations is that we apply equivalence preserving transformations and, therefore, the property to be verified holds in the initial  $\text{CLP}(\mathbb{Z})$  program *if and only if* it holds in the specialized  $\text{CLP}(\mathbb{Z})$  program. In some cases this equivalence preservation is an advantage of the specialization-based verification techniques over the approximation-based techniques. For instance, if we want to prove that a given state is not reachable and this property does not hold in the  $\text{CLP}(\mathbb{R})$  encoding (even if it holds in the  $\text{CLP}(\mathbb{Z})$  encoding), then we will not be able to prove the unreachability property of interest by computing any further approximation (see our example in Section 5).

Another difference between specialization-based verification techniques and static program analysis techniques is that program specialization allows *polyvariance* [24], that is, it can produce several specialized versions for the same predicate (see our example in Section 5), while static program analysis produces one approximation for each predicate. Polyvariance is a potential advantage, as it could be exploited for a more precise analysis, but, at the same time, it requires a suitable control to avoid the explosion in size of the specialized program. The issue of controlling polyvariance is left for future work.

In this paper we have considered constraints consisting of conjunctions of linear inequations. In the case of non-linear inequations the relaxation from integer to real numbers is even more advantageous, as the satisfiability of non-linear inequations is undecidable on the integers and decidable on the reals. Our techniques smoothly extend to the non-linear case which, for reasons of simplicity, we have not considered.

Finally, in this paper we have considered transformation rules and strategies for program specialization. An issue for future research is the extension of relaxation techniques to more general unfold/fold rules, including, for instance: (i) negative unfolding, and (ii) folding using multiple clauses with multiple literals in their bodies (see, for instance, [18]).

## Acknowledgements

This work has been partially supported by PRIN-MIUR. The last author has been supported by an ERCIM grant during his visit at LORIA-INRIA, France.

## References

1. A. Annichini, A. Bouajjani, and M. Sighireanu. TRex: A tool for reachability analysis of complex systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the CAV 2001, Paris, France, Lecture Notes in Computer Science* 2102, pages 368–372. Springer, 2001.
2. K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19, 20:9–71, 1994.
3. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1-2):3–21, 2008.
4. G. Banda and J. P. Gallagher. Analysis of linear hybrid systems in CLP. In Michael Hanus, editor, *LOPSTR 2008, Valencia, Spain, Lecture Notes in Computer Science* 5438, pages 55–70. Springer, 2009.
5. G. Banda and J. P. Gallagher. Constraint-based abstract semantics for temporal logic: A direct approach to design and implementation. In E. M. Clarke and A. Voronkov, editors, *LPAR 2010, Lecture Notes in Artificial Intelligence* 6355, pages 27–45. Springer, 2010.
6. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Acceleration from theory to practice. *International Journal on Software Tools for Technology Transfer*, 10(5):401–424, 2008.
7. B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In *Proceedings of CONCUR '99, Lecture Notes in Computer Science* 1664, pages 178–193. Springer, 1999.
8. W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *JACM*, 43(1), 1996.
9. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
10. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *Proceedings of the 4th ACM-SIGPLAN POPL '77*, pages 238–252. ACM Press, 1977.
11. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth ACM Symposium on Principles of Programming Languages (POPL '78)*, pages 84–96. ACM Press, 1978.
12. S.-J. Craig and M. Leuschel. A compiler generator for constraint logic programs. In M. Broy and A. V. Zamulin, editors, *5th Ershov Memorial Conference, PSI 2003, Lecture Notes in Computer Science* 2890, pages 148–161. Springer, 2003.
13. G. Delzanno and A. Podelski. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer*, 3(3):250–270, 2001.
14. S. Etalle and M. Gabbriellini. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
15. F. Fioravanti. *Transformation of Constraint Logic Programs for Software Specialization and Verification*. PhD thesis, Università di Roma “La Sapienza”, Italy, 2002.
16. F. Fioravanti, A. Pettorossi, and M. Proietti. Automated strategies for specializing constraint logic programs. In K.-K. Lau, editor, *Proceedings of LOPSTR '00, London, UK, Lecture Notes in Computer Science* 2042, pages 125–146. Springer-Verlag, 2001.
17. F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite state systems by specializing constraint logic programs. In *Proceedings of the ACM SIGPLAN Workshop on Verification and Computational Logic VCL'01, Florence (Italy)*, Technical Report DSSE-TR-2001-3, pages 85–96. University of Southampton, UK, 2001.

18. F. Fioravanti, A. Pettorossi, and M. Proietti. Transformation rules for locally stratified constraint logic programs. In K.-K. Lau and M. Bruynooghe, editors, *Program Development in Computational Logic*, Lecture Notes in Computer Science 3049, pages 292–340. Springer-Verlag, 2004.
19. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Program specialization for verifying infinite state systems: An experimental evaluation. In M. Alpuente, editor, *Proceedings of LOPSTR '10, Hagenberg, Austria*, Lecture Notes in Computer Science 6564, pages 164–183. Springer, 2011.
20. L. Fribourg. Constraint logic programming applied to model checking. In A. Bossi, editor, *Proceedings of LOPSTR '99, Venezia, Italy*, Lecture Notes in Computer Science 1817, pages 31–42. Springer-Verlag, 2000.
21. L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints*, 2(3/4):305–335, 1997.
22. J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS '95, Aarhus, Denmark*, Lecture Notes in Computer Science 1019, pages 89–110. Springer, 1996.
23. T. J. Hickey and D. A. Smith. Towards the partial evaluation of CLP languages. In *Proceedings of the 1991 ACM Symposium PEPM '91, New Haven, CT, USA*, SIGPLAN Notices, 26, 9, pages 43–51. ACM Press, 1991.
24. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
25. LASH. homepage: <http://www.montefiore.ulg.ac.be/~boigelot/research/lash>.
26. J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 307–316. ACM, 2011.
27. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proceedings of the Third ATVA 2005, Taipei, Taiwan*, Lecture Notes in Computer Science 3707, pages 489–503. Springer, 2005.
28. M. Leuschel and M. Bruynooghe. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4&5):461–515, 2002.
29. M. Leuschel and H. Lehmann. Coverability of reset Petri nets and other well-structured transition systems by partial deduction. In J. W. Lloyd, editor, *Proceedings of the First International Conference on Computational Logic (CL 2000), London, UK, 24-28 July*, Lecture Notes in Artificial Intelligence 1861, pages 101–115. Springer-Verlag, 2000.
30. M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialization. In A. Bossi, editor, *Proceedings LOPSTR '99, Venezia, Italy*, Lecture Notes in Computer Science 1817, pages 63–82. Springer, 2000.
31. J. C. Peralta and J. P. Gallagher. Convex hull abstractions in specialization of CLP programs. In M. Leuschel, ed., *Proceedings of LOPSTR 2002, Lecture Notes in Computer Science 2664*, pages 90–108, 2003.
32. W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.
33. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.
34. H. Seki. On negative unfolding in the answer set semantics. In *Proceed. LOPSTR 2008*, Lecture Notes in Computer Science 5438, pages 168–184. Springer, 2009.
35. A. Wrzós-Kaminska. Partial evaluation in constraint logic programming. In Z.W. Ras and M. Michalewicz, editors, *Proceedings of the 9th International Symposium on Foundations of Intelligent Systems, Zakopane, Poland*, Lecture Notes in Computer Science 1079, pages 98–107. Springer-Verlag, 1996.
36. T. Yavuz-Kahveci and T. Bultan. Action Language Verifier: An infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.