Speeding Up the Constraint-Based Method in Difference Logic ⋆

Lorenzo Candeago[1], Daniel Larraz[2], Albert Oliveras[2], Enric Rodríguez-Carbonell[2], and Albert Rubio[2]

[1] SpazioDati
[2] Universitat Politècnica de Catalunya, Barcelona

Abstract. Over the years the constrained-based method has been successfully applied to a wide range of problems in program analysis, from invariant generation to termination and non-termination proving. Quite often the semantics of the program under study as well as the properties to be generated belong to difference logic, i.e., the fragment of linear arithmetic where atoms are inequalities of the form $u - v \leq k$. However, so far constrained-based techniques have not exploited this fact: in general, Farkas' Lemma is used to produce the constraints over template unknowns, which leads to non-linear SMT problems. Based on classical results of graph theory, in this paper we propose new encodings for generating these constraints when program semantics and templates belong to difference logic. Thanks to this approach, instead of a heavyweight non-linear arithmetic solver, a much cheaper SMT solver for difference logic or linear integer arithmetic can be employed for solving the resulting constraints. We present encouraging experimental results that show the high impact of the proposed techniques on the performance of the VeryMax verification system.

1 Introduction

Since Colón's *et al.* seminal paper [1], the so-called *constrained-based method* has been applied with success to a wide range of problems in system verification, from invariant generation in Petri nets [2], hybrid systems [3] and programs with arrays [4,5], to termination [6,7] and non-termination proving [8]. In most of these applications, one is interested in generating *linear properties*, e.g., linear invariants or linear ranking functions. In these cases, Farkas' Lemma is employed for producing the constraints over the template unknowns. As a result, an SMT *non-linear* formula is obtained, for which a model has to be found. Although great advances have been made in non-linear SMT solvers [9,10,11], the applicability of the approach is still strongly conditioned by the current technology for dealing with this kind of formulas.

A way to circumvent the bottleneck of using non-linear constraint solvers is to exploit the fragment of logics in which the program under study is described.

⋆ Partially supported by Spanish MINECO under grant TIN2015-69175-C4-3-R.

Although this has not been explored so far in the constraint-based method, other more mature approaches for program analysis such as abstract interpretation [12] have profited from this sort of refinements since the early days of their inception. Indeed, there is a wide variety of non-relational and weakly-relational numerical abstract domains which cover different subsets of linear arithmetic, but whose complexity is lower than that of the full language [13]: *intervals* [14], *zones* [15], *octagons* [16] and *octahedra* [17], to name a few. Also in the model checking community, it is a common practice to focus on particular subclasses of linear inequalities as a means to improve efficiency. In particular, *potential constraints* have been employed in the verification of several kinds of timed and concurrent systems [18,19,20].

In this paper we restrict our attention to *difference logic* over the integers, in which atoms are inequalities of the form $u - v \leq k$, where u and v are integer variables and $k \in \mathbb{Z}$. This fragment of linear arithmetic corresponds to the aforementioned zone abstract domain in abstract interpretation, and to the potential constraints in model checking.

Our contributions in this work are the following:

- we propose an encoding for satisfiability and unsatisfiability of sets of inequalities in difference logic including templates, which results in formulas of difference logic. This is noteworthy since current approaches to equivalent problems in general full linear arithmetic lead to non-linear formulas.
- for the problem of, given a set of inequalities with free independent terms, choosing an invariant subset that proves an assertion, we present two encodings, one for full linear arithmetic and another specialized one for difference logic. While the former leads to non-linear formulas, again the latter falls into a more tractable fragment, in this case linear arithmetic.
- we present an experimental evaluation with the constraint-based verification system VeryMax [21]. We consider the problem of proving the absence of out-of-bounds array accesses in a benchmark suite of numerical programs, and our results show that the expressiveness of difference logic is sufficient to succeed in the majority of the cases, while a remarkable boost in performance is obtained thanks to the proposed techniques.

2 Background

2.1 Programs, Invariants and Safety

Let us fix a set of (integer) program *variables* $\mathcal{X} = \{x_1, \ldots, x_n\}$, and denote by $\mathcal{F}(\mathcal{X})$ the formulas consisting of conjunctions of linear inequalities[3] over the variables $\mathcal{X}$. Let $\mathcal{L}$ be the set of program *locations*, which contains a set $\mathcal{L}_0$ of *initial* locations. Program *transitions* $\mathcal{T}$ are tuples (ℓ_S, τ, ℓ_T), where ℓ_S and $\ell_T \in \mathcal{L}$ represent the *source* and *target* locations respectively, and $\tau \in \mathcal{F}(\mathcal{X} \cup \mathcal{X}')$ describes the transition relation. Here $\mathcal{X}' = \{x'_1, \ldots, x'_n\}$ represent the values of

[3] Note that equalities can be considered as conjunctions of inequalities.

the variables after the transition.[4] A transition is *initial* if its source location is initial. The set of initial transitions is denoted by $\mathcal{T}_0$. A *program* is a pair $\mathcal{P} = (\mathcal{L}, \mathcal{T})$, which can be viewed as a directed graph where nodes are the locations $\mathcal{L}$, and edges are the transitions $\mathcal{T}$.

A *state* $s = (\ell, \boldsymbol{x})$ consists of a location $\ell \in \mathcal{L}$ and a *valuation* $\boldsymbol{x} : \mathcal{X} \to \mathbb{Z}$. A state is *initial* if its location is initial. We denote a *computation step* with transition $t = (\ell_S, \tau, \ell_T)$ by $(\ell_S, \boldsymbol{x}) \to_t (\ell_T, \boldsymbol{x}')$ when the valuations $\boldsymbol{x}$, $\boldsymbol{x}'$ satisfy the transition relation τ of t. We use $\to_{\mathcal{P}}$ if we do not care about the executed transition, and $\to^*_{\mathcal{P}}$ to denote the transitive-reflexive closure of $\to_{\mathcal{P}}$. We say that a state s is *reachable* if there exists an initial state s_0 such that $s_0 \to^*_{\mathcal{P}} s$.

An *assertion* (ℓ, φ) is a pair of a location $\ell \in \mathcal{L}$ and a formula φ with free variables $\mathcal{X}$. A program is *safe* with respect to the assertion (ℓ, φ) if for every reachable state $(\ell, \boldsymbol{x})$, we have that $\boldsymbol{x} \models \varphi$ holds.

A map $\mathcal{I} : \mathcal{L} \to \mathcal{F}(\mathcal{X})$ is an *invariant* if for every $\ell \in \mathcal{L}$, the program is safe with respect to $(\ell, \mathcal{I}(\ell))$. An important class of invariants are inductive invariants. A map $\mathcal{I}$ is an *inductive invariant* if the following two conditions hold:

Initiation: For $(\ell_S, \tau, \ell_T) \in \mathcal{T}_0$: $\quad \tau \models \mathcal{I}(\ell_T)'$
Consecution: For $(\ell_S, \tau, \ell_T) \in \mathcal{T} - \mathcal{T}_0$: $\mathcal{I}(\ell_S) \wedge \tau \models \mathcal{I}(\ell_T)'$

If only the condition **Consecution** is fulfilled, the map $\mathcal{I}$ is called a *conditional* inductive invariant.

One of the key problems in program analysis is to determine whether a program is safe with respect to a given assertion (ℓ, φ). This is typically proved by computing an (inductive) invariant $\mathcal{I}$ such that the following condition holds:

Safety: $\mathcal{I}(\ell) \models \varphi$

In this case we say that the invariant $\mathcal{I}$ *proves* the assertion (ℓ, φ).

Finally, we say a transition $t = (\ell_S, \tau, \ell_T)$ is *disabled* if it can never be executed, i.e., if for any reachable state $(\ell_S, \boldsymbol{x})$, there does not exist any $\boldsymbol{x}'$ such that $(\boldsymbol{x}, \boldsymbol{x}')$ satisfies τ. One can prove this by computing an invariant $\mathcal{I}$ such that $\mathcal{I}(\ell_S) \models \neg\tau$. Disabled transitions allow one to simplify the program under analysis, since they can be soundly removed from the program. In general, if $\mathcal{I}$ is an invariant map, then any transition $t = (\ell_S, \tau, \ell_T)$ can be soundly strengthened by replacing the transition relation τ by $\mathcal{I}(\ell_S) \wedge \tau$.

2.2 Constraint-Based Invariant Generation

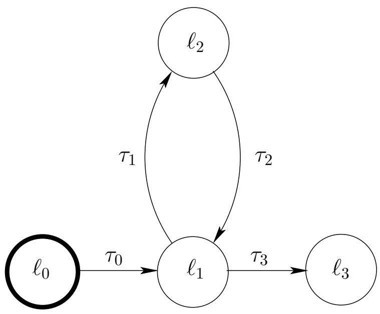
Invariants can be generated using the *constraint-based* (also called *template-based*) method [1]. The idea is to consider *templates* for candidate invariant properties. These templates involve both the program variables as well as fresh template variables whose values have to be determined to ensure invariance. To this end, conditions **Initiation** and **Consecution** are enforced by means of *constraints*. Any solution to these constraints then yields an invariant. If templates are meant to represent linear inequalities, Farkas' Lemma [22] is used to express the constraints in terms of the template variables:

[4] For $\varphi \in \mathcal{F}(\mathcal{X})$, the formula $\varphi' \in \mathcal{F}(\mathcal{X}')$ is the version of φ using primed variables.

Theorem 1 (Farkas' Lemma). *Let S be a system of linear inequalities $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ $(\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m)$ over real variables $\boldsymbol{x}^T = (x_1, \ldots, x_n)$. Then S has no solution iff there is $\boldsymbol{\lambda} \in \mathbb{R}^m$ (called the* multipliers*) such that $\boldsymbol{\lambda} \geq \boldsymbol{0}$, $\boldsymbol{\lambda}^T \boldsymbol{A} = \boldsymbol{0}$ and $\boldsymbol{\lambda}^T \boldsymbol{b} \leq -1$.*

In general, an SMT formula over non-linear arithmetic is obtained. By assigning weights to the different conditions, invariant generation can be cast as an optimization problem in the Max-SMT framework [7,8,21].

Example 1. Let us consider the program in Figure 1.



$$\begin{aligned}
&\tau_0 : n - x_0 \geq 1, x_0' = x_0, n' = n, i' = x_0\\
&\tau_1 : i \leq n - 2,\ \ x_0' = x_0, n' = n, i' = i\\
&\tau_2 : \qquad\qquad\ \ x_0' = x_0, n' = n, i' = i + 1\\
&\tau_3 : i \geq n - 1,\ \ x_0' = x_0, n' = n, i' = i
\end{aligned}$$

Fig. 1. Program with a single initial location ℓ_0.

Let us take the following 3 templates expressing general linear inequalities, one for each non-initial location:

$$T_j := c_{0j}\, x_0 \,+\, c_{1j}\, n \,+\, c_{2j}\, i \leq d_j \qquad \text{for all } j = 1 \ldots 3\,.$$

By imposing that these templates yield an invariant, we obtain the conditions [5]:

Initiation:	$\tau_0 \models T_1'$,	i.e.,	$\tau_0 \wedge \neg T_1'$	unsatisfiable
Consecution:	$T_1 \wedge \tau_1 \models T_2'$,	i.e.,	$T_1 \wedge \tau_1 \wedge \neg T_2'$	unsatisfiable
	$T_2 \wedge \tau_2 \models T_1'$,	i.e.,	$T_2 \wedge \tau_2 \wedge \neg T_1'$	unsatisfiable
	$T_1 \wedge \tau_3 \models T_3'$,	i.e.,	$T_1 \wedge \tau_3 \wedge \neg T_3'$	unsatisfiable.

By fleshing out the transition relations, expanding the templates and simplifying, these four formulas are equivalent to

(1) $x_0 - n \leq -1 \wedge -(c_{01} \,+\, c_{21})\, x_0 \,-\, c_{11}\, n \leq -d_1 - 1$
(2) $c_{01}\, x_0 \,+\, c_{11}\, n \,+\, c_{21}\, i \leq d_1 \wedge i - n \leq -2 \wedge -c_{02}\, x_0 \,-\, c_{12}\, n \,-\, c_{22}\, i \leq -d_2 - 1$
(3) $c_{02}\, x_0 \,+\, c_{12}\, n \,+\, c_{22}\, i \leq d_2 \wedge \wedge - c_{01}\, x_0 \,-\, c_{11}\, n \,-\, c_{21}\, i \leq -d_1 - 1 + c_{21}$
(4) $c_{01}\, x_0 \,+\, c_{11}\, n \,+\, c_{21}\, i \leq d_1 \wedge n - i \leq 1 \wedge -c_{03}\, x_0 \,-\, c_{13}\, n \,-\, c_{23}\, i \leq -d_3 - 1$

[5] For simplicity, no assertion and thus no **Safety** condition is considered here.

respectively. Now Farkas' Lemma is applied to express unsatisfiability. Namely, for (1) we consider non-negative multipliers $\lambda_{11}, \lambda_{12}$ such that the linear combination that consists in multiplying the first inequality by λ_{11} and the second inequality by λ_{12} results in a trivially false inequality. For that, we need the coefficients of x_0 to cancel out, i.e., $\lambda_{11} - \lambda_{12}(c_{01} + c_{21}) = 0$, and the same for n, i.e., $-\lambda_{11} - \lambda_{12}c_{11} = 0$. With respect to the independent term, we force that it is smaller than or equal to -1, i.e., $-\lambda_{11} + \lambda_{12}(-d_1 - 1) \leq -1$, which will create a trivially false inequality. All in all, we get the non-linear formula

$$\begin{aligned} \exists \lambda_{11}\lambda_{12} \ \big(&\lambda_{11}, \lambda_{12} \geq 0 \ \wedge \\ &\lambda_{11} - \lambda_{12}(c_{01} + c_{21}) = -\lambda_{11} - \lambda_{12}c_{11} = 0 \ \wedge \\ &-\lambda_{11} + \lambda_{12}(-d_1 - 1) \leq -1\big) \end{aligned} \tag{1}$$

Similar constraints are obtained for (2)-(4). □

2.3 Difference Logic and Graph Theory

Given variables u and v and a numeric constant k, henceforth we will refer to an inequality of the form $u - v \leq k$ as a *difference inequality*. The fragment of (quantifier-free) first-order logic where atoms are difference inequalities is called *difference logic*.

Sets (conjunctions) of difference inequalities, also called *difference systems*, have long been studied in the literature [23]. For instance, they can be represented as graphs as follows. Given a difference system S defined over variables $v_1, v_2, \ldots, v_n$, we consider the weighted graph G with vertices $(v_1, v_2, \ldots, v_n)$ and an edge $v_i \xrightarrow{k} v_j$ for each inequality $v_i - v_j \leq k \in S$. This graph is called the *constraint graph* of S.

It is well-known that a constraint graph has interesting properties as regards to the solutions of the corresponding difference system:

Theorem 2. *Let S be a difference system, and G its constraint graph. Then S has no solution iff G has a negative cycle.*

This result is a particular case of Farkas' Lemma. It essentially ensures that, for difference systems, the multipliers of Farkas' Lemma are either 1 or 0 (the difference inequality belongs to the negative cycle or it does not, respectively).

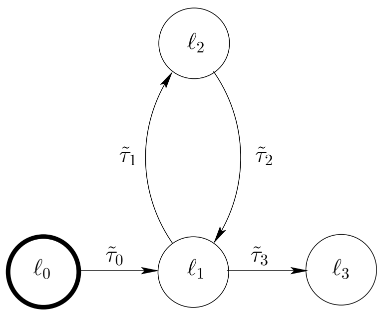
One of the most important practical consequences of Theorem 2 is that any algorithm that is able to detect negative cycles in weighted graphs (such as, for instance, Bellman-Ford, or Floyd-Warshall [23]) can be used to determine the existence of solutions to a difference system.

Theorem 2 can be extended to allow also *bound* inequalities, i.e., inequalities of the form $v \leq k$ or $v \geq k$, where v is a variable and k is a numeric constant: Given a system S that includes difference inequalities as well as bound inequalities, a fresh variable v_0 is introduced. Then a new system S^* is defined, which is like S but where each inequality of the form $v_i \leq k$ in S is replaced by $v_i - v_0 \leq k$, and each $v_i \geq k$, or equivalently $-v_i \leq -k$, is replaced by $v_0 - v_i \leq -k$. It is not difficult to prove that S has a solution iff S^* has one.

3 Proving Safety of Difference Programs

In this paper we will focus on *difference programs*, that is, programs whose transition relations are conjunctions of difference inequalities.

Although this may seem rather restrictive, in fact more general programs can be cast into this form: for any program with difference as well as bound inequalities in the transition relations, there exists an equivalent difference program, as it is well-known in the literature [15]. The trick consists in introducing an artificial variable x_0, which intuitively is always zero, and then transform bound inequalities into difference inequalities by adding x_0 with the appropriate sign. Thus, e.g., $n \geq 1$ is transformed into $n - x_0 \geq 1$. Moreover, the equation $x_0' = x_0$ has to be added to all transitions. For example, after this transformation the program in Figure 2 leads to that in Figure 1.



$\tilde{\tau}_0 : n \geq 1, \quad n' = n, i' = 0$
$\tilde{\tau}_1 : i \leq n - 2, n' = n, i' = i$
$\tilde{\tau}_2 : \quad n' = n, i' = i + 1$
$\tilde{\tau}_3 : i \geq n - 1, n' = n, i' = i$

Fig. 2. Program with difference and bound inequalities in the transition relations.

The problem we consider in this section is, given a location ℓ and a difference inequality φ, to prove that the program under consideration is safe with respect to the assertion (ℓ, φ). As the following theorem states, proving safety of a difference program is in general undecidable, and therefore we cannot hope for a sound and complete terminating algorithm that solves the problem:

Theorem 3. *Given a difference program $\mathcal{P}$, a location $\ell \in \mathcal{L}$ and a difference inequality φ, the problem of deciding whether $\mathcal{P}$ is safe with respect to the assertion (ℓ, φ) is undecidable.*

Proof. See Appendix A.

3.1 Specialization of the Constraint-Based Method

Here we attempt to prove difference programs safe by finding invariants consisting of difference inequalities with a specialization of the constraint-based method. [6] Let us first illustrate the gist of our technique with an example.

[6] Here a simplified procedure for proving an assertion is described in order to highlight the key contribution of this work, that is, how to circumvent non-linearities.

Example 2. Again let us consider the program in Figure 1 and assign a template to each non-initial location: $T_j := c_{0j}\, x_0 + c_{1j}\, n + c_{2j}\, i \leq d_j$ for all $j = 1 \ldots 3$. This program is a difference program. Let us also consider the assignment $c_{0,j} = 0, c_{1,j} = -1, c_{2,j} = 1$ for all $j = 1 \ldots 3$, $d_1 = d_3 = -1$, $d_2 = -2$, which instantiates the templates as follows:

$$T_1 \equiv i - n \leq -1 \qquad T_2 \equiv i - n \leq -2 \qquad T_3 \equiv i - n \leq -1,$$

and check that they are invariant. Since the above inequalities belong to difference logic, we can use Theorem 2 to check that indeed the formulas $\tau_0 \wedge \neg T_1'$, $T_1 \wedge \tau_1 \wedge \neg T_2'$, $T_2 \wedge \tau_2 \wedge \neg T_1'$ and $T_1 \wedge \tau_3 \wedge \neg T_3'$ are unsatisfiable, as required by the **Initiation** and **Consecution** conditions. By the theorem, the unsatisfiability of each of these formulas is equivalent to the existence of a negative cycle in the corresponding graph. In Figure 3 some of these graphs are shown for the particular solution considered here, and the respective negative cycles are highlighted. Solving the **Initiation** and **Consecution** constraints over the template coefficients can thus be seen as adding new weighted edges to the graphs of the transition relations so that, in the end, all graphs have a negative cycle. Notice that this must be done in a consistent way, so that, for instance, the edge of $\neg T_1'$ is the same in $\tau_0 \wedge \neg T_1'$ and in $T_2 \wedge \tau_2 \wedge \neg T_1'$.

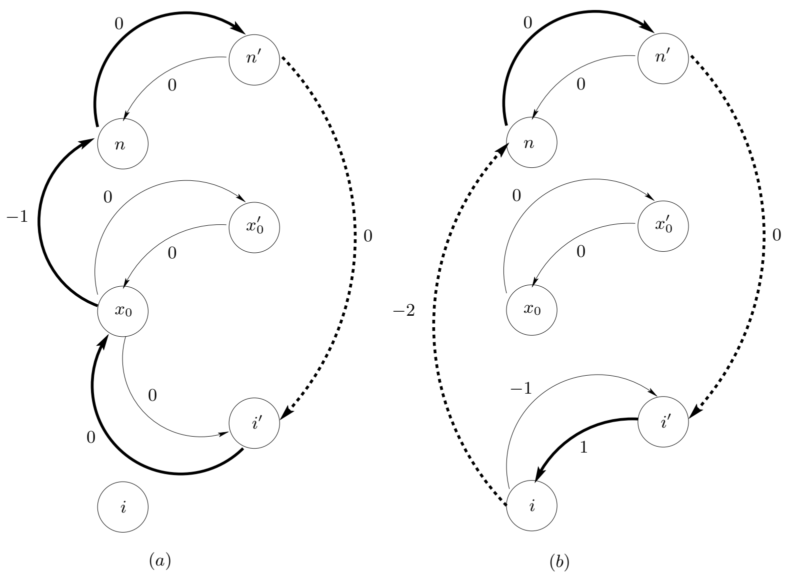


Fig. 3. Graphs for the formulas $\tau_0 \wedge \neg T_1'$ (a) and $T_2 \wedge \tau_2 \wedge \neg T_1'$ (b). The edges corresponding to the templates (or their negation) are dashed. The edges forming the negative cycles are highlighted with thicker lines.

□

In what follows, we assume we have associated to each non-initial location ℓ a template invariant T_ℓ of the form

$$c_{0,\ell}\, x_0 \;+\; c_{1,\ell}\, x_1 \;+\; \ldots \;+\; c_{n,\ell}\, x_n \le d_\ell$$

where the $c_{i,\ell}$ and the d_ℓ are template unknowns. For obvious reasons we will refer to the $c_{i,\ell}$ as *left-hand side variables*, whereas the d_ℓ are called *right-hand side variables* (*LHS* and *RHS* variables, respectively). Here we focus on difference inequalities, and therefore the domain of LHS variables is $\{+1, 0, -1\}$, while the domain of RHS variables is $\mathbb{Z}$.

We propose to find appropriate values for the RHS and LHS variables following an *eager* approach: we encode all required constraints into an SMT formula, and then use an off-the-shelf SMT solver to solve the resulting problem. As will be seen next, in our particular case the atoms in the SMT formula will be either Boolean variables or bound inequalities or difference inequalities. By virtue of the results reviewed in Section 2.3, the generated formula can be handled with an SMT solver of difference logic, for which efficient implementations are available.

The formula that expresses the constraints over template variables (LHS and RHS variables) is a conjunction of the following ingredients.

Membership to Difference Logic. First of all, we have to express that all templates are difference inequalities. To that end, for each LHS variable c_i we introduce two auxiliary Boolean variables: c_i^+ and c_i^-. Intuitively, c_i^+ will be true iff c_i is assigned to $+1$, and c_i^- will be true iff c_i is assigned to -1. If both c_i^+ and c_i^- are false, then c_i is 0. We need to enforce: (i) that the c_i^+ and c_i^- cannot be true at the same time, (ii) that exactly one of the c_i in each template is $+1$ (i.e., exactly one of the c_i^+ is true), and (iii) exactly one is -1 (i.e., exactly one of the c_i^- is true). This can be done by using the encoding of ALO (*At Least One*) constraints with clauses and one of the encodings of AMO (*At Most One*) constraints that are available in the literature (e.g., quadratic, logarithmic [24] or ladder [25]).

Unsatisfiability of Difference Systems. When encoding the **Initiation**, **Consecution** and **Safety** conditions, essentially one has to impose the unsatisfiability of a set of difference inequalities, some of which may be templates. Namely, in **Initiation** and **Safety** one has a single template, but while in the former the template appears negatively, in the latter it appears positively. On the other hand, in **Consecution** two templates appear, one negatively and the other positively. Here we will elaborate on this latter case, being the others simpler and easy to derive from it.

Thus, let $\mathcal{S}$ be a difference system over program variables $\mathcal{X}$, $\mathcal{X}'$ such that

$$c_0\, x_0 + \ldots + c_n\, x_n \le d \;\wedge\; \mathcal{S} \;\;\wedge\;\; \neg(\tilde{c_0}\, x_0' + \ldots + \tilde{c_n}\, x_n' \le \tilde{d})$$

must be unsatisfiable. Our goal is to instantiate the templates so that this is the case. Note $\neg(\tilde{c_0}\, x_0' + \ldots + \tilde{c_n}\, x_n' \le \tilde{d})$ is equivalent to $-\tilde{c_0}\, x_0' - \ldots - \tilde{c_n}\, x_n' \le -\tilde{d} - 1$.

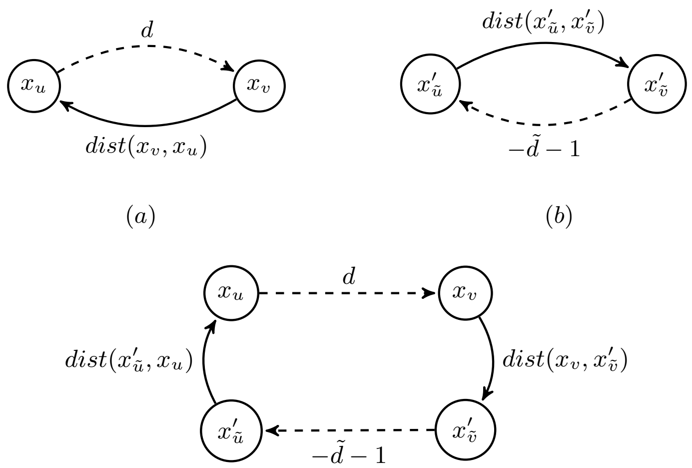
To ensure unsatisfiability, i.e., that a negative cycle exists, we first construct $\mathcal{G}$, the constraint graph induced by $\mathcal{S}$. We then apply Floyd-Warshall algorithm in order to compute the distances $dist(y, z)$ for each pair of vertices y and z in $\mathcal{G}$.

If for some vertex y we have $dist(y, y) < 0$, then $\mathcal{S}$ has a negative cycle and hence the unsatisfiability requirement is fulfilled independently from the templates. In this case, no clause needs to be added.

Otherwise $\mathcal{S}$ has no negative cycles, and the only possibility to construct one is to go through the edges induced by the templates. Let us consider an assignment such that $c_u = +1$, $c_v = -1$, $\tilde{c}_{\tilde{u}} = +1$ and $\tilde{c}_{\tilde{v}} = -1$ (i.e. c_u^+, c_v^-, $c_{\tilde{u}}^+$ and $c_{\tilde{v}}^-$ are true). In this case the instantiation of the positive template is $x_u - x_v \le d$, and the instantiation of the negation of the other template is $x'_{\tilde{v}} - x'_{\tilde{u}} \le -\tilde{d} - 1$. Hence, the former induces an edge from x_u to x_v with weight d, while the latter induces an edge from $x'_{\tilde{v}}$ to $x'_{\tilde{u}}$ with weight $-\tilde{d} - 1$.

To form a negative cycle, either (i) the cycle contains only the positive template, or (ii) contains only the negative template, or (iii) contains both. The first situation can be seen in Figure 4 (a), where it is needed that $dist(x_v, x_u) + d < 0$. The second situation is depicted in Figure 4 (b), where we need that $dist(x'_{\tilde{u}}, x'_{\tilde{v}}) - \tilde{d} - 1 < 0$. Finally the third situation can be seen in Figure 4 (c), where the needed condition is $d + dist(x_v, x'_{\tilde{v}}) - \tilde{d} - 1 + dist(x'_{\tilde{u}}, x_u) < 0$. Hence, we add the following clause:

$$c_u^+ \wedge c_v^- \wedge \tilde{c}_{\tilde{u}}^+ \wedge \tilde{c}_{\tilde{v}}^- \implies \begin{aligned} d \quad & \le -dist(x_v, x_u) - 1 \quad \vee \\ -\tilde{d} & \le -dist(x'_{\tilde{u}}, x'_{\tilde{v}}) \qquad \vee \\ d - \tilde{d} & \le -dist(x_v, x'_{\tilde{v}}) - dist(x'_{\tilde{u}}, x_u) \end{aligned} \tag{2}$$



(c)

Fig. 4. The only three ways of creating a negative cycle.

Note that it might be the case that some of the paths represented in Figure 4 do not actually exist. For example, if x_u is unreachable from x_v, i.e., $dist(x_v, x_u)$ is infinite, then there cannot be a negative cycle that only uses the positive template, independently of the value we give to its RHS variable. Hence the first inequality in the clause of Equation 2 can be dropped.

This reasoning is applied to all vertices, namely, to all $u, v, \tilde{u}, \tilde{v}$ with $u \neq v$, $\tilde{u} \neq \tilde{v}$, and $u, v, \tilde{u}, \tilde{v} \in \{0, 1, \ldots, n\}$, adding in each case the respective clause.

Satisfiability of Difference Systems. The opposite problem to the previous one, that is, to enforce that a difference system is *satisfiable*, also arises in the constraint-based method. This is the case when, for example, one performs several rounds of invariant generation as described above, and requires that the newly generated invariants are not redundant with respect to the already computed ones: then there must exist a witness that certifies the non-redundancy.

Hence, let $\mathcal{S}$ be a difference system over program variables $\mathcal{X}$ such that

$$\mathcal{S} \wedge \neg(c_0\, x_0 + \ldots + c_n\, x_n \leq d) \;\;\equiv\;\; \mathcal{S} \wedge -c_0\, x_0 - \ldots - c_n\, x_n \leq -d - 1$$

must be satisfiable [7]. By Theorem 2, this amounts to proving that no negative cycle exists in the corresponding constraint graph. Again, we will start by constructing $\mathcal{G}$, the constraint graph induced by $\mathcal{S}$, and applying Floyd-Warshall.

If a negative cycle is already detected, the satisfiability requirement cannot be met. Otherwise $\mathcal{S}$ has no negative cycles, and the only possibility to achieve one is to go through the edge induced by the template. If $c_u = 1$ and $c_v = -1$, then the negation of the template is $x_v - x_u \leq -d - 1$, which induces an edge from x_v to x_u with weight $-d - 1$. This edge is part of a negative cycle iff $dist(x_u, x_v) - d - 1 < 0$. Since we want to avoid negative cycles, we should enforce that $dist(x_u, x_v) - d - 1 \geq 0$, or equivalently $d \leq dist(x_u, x_v) - 1$. Hence, we should add the clause:

$$c_u^+ \wedge c_v^- \implies d \leq dist(x_u, x_v) - 1\,.$$

Note that if $dist(x_u, x_v)$ is infinite then the clause is trivially satisfied, and hence can be dropped.

3.2 Experiments

In order to experimentally evaluate the encoding presented in the previous section, we first ran our verification system VeryMax [21] on C++ implementations of numerical algorithms from [26], checking whether all array accesses are within bounds. For each such check, several queries need to be processed, all of which consist of a small program with an assertion to be proved. Among them, we chose the ones where the program and the assertion can be expressed in difference logic. For these queries, VeryMax requires one of five possible outputs:

[7] Note that, if the template and the inequalities in $\mathcal{S}$ are general linear inequalities, this yields a non-linear problem.

I. An invariant at each location proving the assertion
II. An invariant at each location disabling a transition
III. A conditional invariant at each location proving the assertion
IV. An invariant at each location
V. None of the previous ones

Solving one such query using the constraint-based method generates an SMT formula with multiple **Initiation**, **Consecution**, **Safety** and other conditions (e.g. no redundant invariants are generated, conditional invariants are compatible with initial transitions) that can be encoded via Farkas' Lemma or via our novel difference logic encoding presented in the previous section. By making some of these conditions soft with the use of appropriate weights as in [27], we can order the five possible outputs from most desirable (I) to least desirable (V). For example, the optimal solution gives output (III) only if no solution exists that gives results (II) or (I).

The resulting Max-SMT formula can be processed with an off-the-shelf Max-SMT solver, such as Opti-Mathsat [28], Z3Opt [29] or Barcelogic [30]. Unfortunately, we had to discard Opti-Mathsat because it cannot deal with non-linearities. Between the remaining two, it was Barcelogic the one that showed a better performance, probably due to its novel method to deal with non-linearities [11]. Regarding the optimization part, Barcelogic implements a very simple branch-and-bound approach as explained in [31]. Due to its better performance, in what follows only experiments with Barcelogic will be reported.

Experiments were performed on an Intel i5 2.8 GHz CPU with 8 Gb of memory. For each of the 3270 generated queries and each encoding, we consider the best solution obtained within a time limit of 5 seconds[8]. In Table 1 we can see the output and the running time of four different encodings: **Farkas** (the standard encoding based on Farkas' Lemma), **FarkasDL** (the previous one additionally restricting the templates to be difference logic), **FarkasDL-$\boldsymbol{\lambda}$** (the previous one additionally imposing that the λ multipliers are 0 or 1), and **Diff Logic** (our novel encoding introduced in the previous section).

Method	(I) Inv. prove	(II) Disable tr.	(III) Cond. inv. prove	(IV) Invariant	(V) Nothing	Time
Farkas	215	427	330	1024	1274	4h 11m 47s
FarkasDL	215	526	322	1042	1165	3h 8m 22s
FarkasDL-$\boldsymbol{\lambda}$	217	594	324	1042	1039	3h 1m 52s
Diff Logic	786	1044	328	1112	0	56m 20s

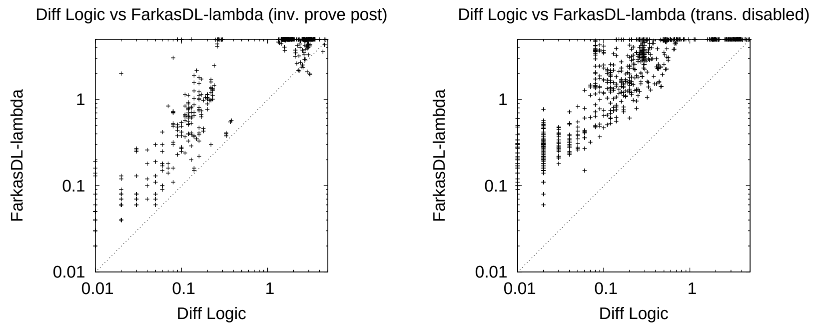
Table 1. Results on the 3270 generated queries with a time limit of 5 seconds.

The experiments confirm our intuition that our specialized difference logic encoding outperforms Farkas both in runtime and in quality of solutions. Even if

[8] This is the time limit used for this type of queries inside VeryMax.

we try to improve Farkas with additional constraints that limit the search space, as in **FarkasDL** and **FarkasDL-$\boldsymbol{\lambda}$**, the differences are still dramatic. We want to remark that in no query Farkas gave a better-quality result than **Diff Logic**.

More detailed results can be seen in Figure 5, where in the scatter plots we display the timings (in seconds, logarithmic scale) over queries whose optimal solution finds invariants proving the assertion (a) or disabling a transition (b). One can see that even the best Farkas-based encoding is systematically slower than **Diff Logic**. We can also observe that in lots of queries Farkas times out, which means that the Max-SMT solver could not prove the solution to be optimal. One could think this is because proving optimality is equivalent to proving unsatisfiability, something at which Barcelogic non-linear techniques are particularly bad. However, a careful inspection of the results reveals the situation is worse, as in more than 80% of the queries the found solution was not optimal.



(a) (b)

Fig. 5. Comparison of **Diff Logic** and **FarkasDL-$\boldsymbol{\lambda}$** runtimes over queries whose optimal solution gives invariants proving the assertion (a) or disabling a transition (b).

4 Finding Invariant Subsets

Another important problem that we need to solve inside VeryMax is the *Invariant Subset Selection Problem.* Formally, we are given a program, an assertion (ℓ_{ass}, φ) and, for each location $\ell \in \mathcal{L}$, a set $Cand(\ell)$ of m_ℓ candidate invariants

$$\begin{array}{rl} c_1^{\ell,1}x_1 + \cdots + & c_n^{\ell,1}x_n \leq d^{\ell,1} \\ c_1^{\ell,2}x_1 + \cdots + & c_n^{\ell,2}x_n \leq d^{\ell,2} \\ \vdots & \\ c_1^{\ell,m_\ell}x_1 + \cdots + & c_n^{\ell,m_\ell}x_n \leq d^{\ell,m_\ell} \end{array}$$

where the $c_i^{\ell,j}$ are fixed integer numbers and the $d^{\ell,j}$ are integer variables. The goal is to select, if it exists, a subset of $Cand(\ell)$ for each $\ell \in \mathcal{L}$, and find an assignment to the $d^{\ell,j}$'s such that (i) the chosen subsets are invariant and (ii) the invariants chosen at ℓ_{ass} imply φ.

As in Sections 2.2 and 3 we will show that, in the general case, if we use Farkas' Lemma we obtain a non-linear formula, whereas non-linearities can be avoided when the program, the assertion and the candidate invariants are difference logic. In this case, the resulting formula belongs to linear arithmetic.

4.1 General Case

We can imagine the process of finding a solution as working in two stages. First of all, we have to select a subset of the candidate invariants at each location, together with their corresponding right-hand sides $d^{\ell,j}$. After that, we need to ensure that **Initiation**, **Consecution** and **Assertion** conditions are satisfied. To prove these conditions, we only need to find the right Farkas' multipliers.

More precisely, for each location $\ell \in \mathcal{L}$ and $1 \leq j \leq m_\ell$, let us consider a Boolean variable $chosen_{\ell,j}$ that indicates whether the j-th invariant in $Cand(\ell)$ is chosen. Additionally, to each coefficient $c_i^{\ell,j}$ we will associate a fresh integer variable $\widehat{c}_i^{\ell,j}$, and to each $d^{\ell,j}$ a fresh integer variable $\widehat{d}^{\ell,j}$. The following formulas

$$chosen_{\ell,j} \implies \bigwedge_{i=1}^{n} \widehat{c}_i^{\ell,j} = c_i^{\ell,j} \quad \wedge \quad \widehat{d}^{\ell,j} = d^{\ell,j} \tag{1}$$

$$\neg chosen_{\ell,j} \implies \bigwedge_{i=1}^{n} \widehat{c}_i^{\ell,j} = 0 \qquad \wedge \quad \widehat{d}^{\ell,j} = 0 \tag{2}$$

constraint the shape of the invariants depending on whether they are chosen or not. In the following, we will consider that $\widehat{Cand}(\ell)$ consists of all elements of $Cand(\ell)$ where all c's and d's have been replaced by their respective $\widehat{c}$'s and $\widehat{d}$'s.

Let us explain how a **Consecution** condition will be enforced (for **Initiation** and **Assertion** an analogous idea applies). Let (ℓ_S, τ, ℓ_T) be the transition to which consecution refers. We want to enforce that $\widehat{Cand}(\ell_S) \wedge \tau \models \widehat{Cand}(\ell_T)'$, which amounts to checking, for each $\widehat{inv}' \in \widehat{Cand}(\ell_T)'$, that $\widehat{Cand}(\ell_S) \wedge \tau \models \widehat{inv}'$, or equivalently, that $\widehat{Cand}(\ell_S) \wedge \tau \wedge \neg \widehat{inv}'$ is unsatisfiable. The latter can be easily encoded into a non-linear formula by using Farkas' Lemma.

4.2 Difference Logic Case

Let us now assume that all candidate invariants, the formula in the assertion and the input program are expressed in difference logic. The idea of the encoding is similar. However, in Section 4.1 new inequalities were globally introduced standing for the original inequalities or the trivial inequality $0 \leq 0$, depending on whether they had been chosen or not. Instead, here we exploit the fact that in Farkas' proofs of unsatisfiability of difference sets, multipliers are 0 or 1: for each unsatisfiability proof that must hold, new inequalities are locally introduced, standing for the *product* of the Farkas' multiplier with the original inequality.

As an example, let us explain how to encode a **Consecution** condition referring to a transition (ℓ_S, τ, ℓ_T). The $chosen_{\ell,j}$ variables will be as before, common to the overall encoding. However, for each $inv \in Cand(\ell_T)$, in order to enforce that $Cand(\ell_S) \wedge \tau \models inv'$, we will now introduce fresh $\widehat{c}$'s and $\widehat{d}$'s and add, for $1 \leq j \leq m_{\ell_S}$, the previous formula (2) and:

$$(\bigwedge_{i=1}^{n} \widehat{c}_i^{\ell_S,j} = 0 \ \wedge \ \widehat{d}^{\ell_S,j} = 0) \quad \vee \quad (\bigwedge_{i=1}^{n} \widehat{c}_i^{\ell_S,j} = c_i^{\ell_S,j} \ \wedge \ \widehat{d}^{\ell_S,j} = d^{\ell_S,j})$$

The intuition is that $\widehat{c_1}^{\ell_S,j} x_1 + \cdots + \widehat{c_n}^{\ell_S,j} x_n \leq \widehat{d}^{\ell_S,j}$ is the inequality resulting from multiplying $c_1^{\ell_S,j} x_1 + \cdots + c_n^{\ell_S,j} x_n \leq d^{\ell_S,j}$ by the corresponding multiplier in Farkas' proof of unsatisfiability of $Cand(\ell_S) \wedge \tau \wedge \neg inv'$. Similarly, let us assume that *inv* is $c_1 x_1 + \cdots + c_n x_n \leq d$, with *chosen* being the variable that indicates whether we pick it or not. Then we will add the formula

$$(\bigwedge_{i=1}^{n} c_i^\star = 0 \ \wedge \ d^\star = 0) \quad \vee \quad (\bigwedge_{i=1}^{n} c_i^\star = -c_i \ \wedge \ d^\star = -1 - d),$$

which intuitively means that $c_1^\star x_1 + \cdots + c_n^\star x_n \leq d$ is the inequality resulting from multiplying $\neg(c_1 x_1 + \cdots + c_n x_n \leq d) \equiv -c_1 x_1 - \cdots - c_n x_n \leq -1 - d$ by the corresponding multiplier in the proof of unsatisfiability of $Cand(\ell_S) \wedge \tau \wedge \neg inv'$.

The encoding finishes by: (i) applying Farkas' Lemma to enforce unsatisfiability of $\widehat{Cand(\ell_S)} \wedge \tau \wedge c_1^\star x_1' + \cdots + c_n^\star x_n' \leq d'$ as in the general case, but now assuming that multipliers are 1, which gives a linear formula F; and (ii) adding the implication $chosen \Rightarrow F$ to the encoding. Detailed experiments comparing the general and the particular encoding for difference logic give similar results to Section 3.2, and we omit them here due to lack of space.

One final remark is that the previous encoding can be extended to solve the problem in Section 3 by exhaustively considering in $Cand(\ell)$ all possible pairs of differences of variables. Note that this allows one to discover simultaneously more than one linear invariant at each location, in particular coinductive invariants. Hence, the price to pay if we want a complete method is moving from a difference logic formula to a linear arithmetic one.

5 Experiments

The goal of this section is to assess to which extent a constraint-based verifier like VeryMax can be improved by incorporating the novel encodings introduced in this paper. Note that it is not uncommon that huge improvements on the runtime of a theorem prover (SAT solver, SMT solver or first-order theorem prover) are diluted into non-significant improvements on the verifier using it.

We compared the original VeryMax safety prover, which uses Farkas as the encoding methodology to find linear invariants, with a new version VeryMaxDL where the problems described in Sections 3 and 4 are solved using the novel encodings. A time limit of 900 seconds was given to each problem. A summary of the experiments can be seen in Table 2, where for each system we report the number of problems found to be safe (Yes), not found to be safe[9] (No), proved safe only by this version of the system (Only-yes) and the total runtime.

[9] Note that this does not mean that they are unsafe.

The results are extremely positive since the runtime is reduced to one third, and the loss of verification power due to generating only difference logic invariants, as opposed to linear invariants, is very limited. We analyzed all problems that VeryMax could prove safe whereas VeryMaxDL could not and they all need linear invariants outside difference logic, which means that they cannot be proved using the techniques on which VeryMaxDL is based.

System	Yes	No	Only-yes	Time
VeryMax	524	312	27	11h 58m 59s
VeryMaxDL	516	320	19	4h 12m 38s

Table 2. Results comparing VeryMax and VeryMaxDL.

Figure 6 contains scatter plots comparing VeryMax and VeryMaxDL on all problems, problems proved safe by VeryMaxDL, and problems not found to be safe by VeryMaxDL. At first glance, although the difference logic version is faster, we observe that the plots are not as clean as the ones of Section 3.2. This is not a surprise: if the subproblems solved via Farkas or difference logic give different results (e.g. they disable different transitions), the overall behavior of the verification system changes and this has an impact on the overall runtime. The second observation is that VeryMaxDL is faster, independently of whether the problem can be found to be safe or not. This opens the way to run both versions in parallel, or even first run VeryMaxDL and if the program cannot be proved safe, run VeryMax in a second attempt.

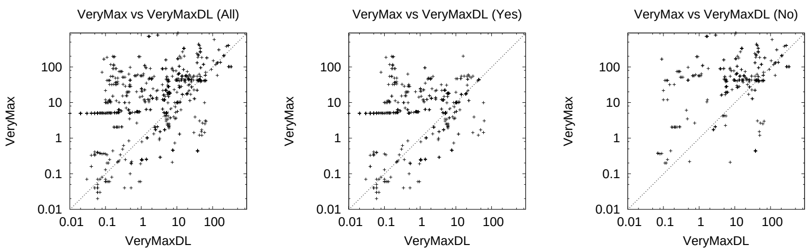


Fig. 6. Scatter plots comparing VeryMax and VeryMax.

6 Conclusions and Future Work

It is well acknowledged that the current bottleneck in the effectiveness of the constraint-based method compared to other approaches for verification is the technology for solving non-linear constraints. In this paper we have introduced novel encodings that, if we restrict ourselves to programs and invariants in difference logic, allow one to replace non-linear solvers by cheaper ones. Experiments show that this yields a huge gain in terms of runtime at the expense of a certain but acceptable loss of verification power.

As future work, we plan to extend the use of similar encodings in our verification system VeryMax. E.g., finding simple ranking functions in (non)-termination problems is a particularly interesting area where related ideas could be applied.

References

1. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: Proc. CAV '03. Volume 2725 of LNCS., Springer (2003) 420–432
2. Sankaranarayanan, S., Sipma, H., Manna, Z.: Petri net analysis using invariant generation. In: Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday. Volume 2772 of LNCS., Springer (2003) 682–701
3. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. Formal Methods in System Design **32**(1) (2008) 25–55
4. Beyer, D., Henzinger, T.A., Majumdar, R., Rybalchenko, A.: Invariant synthesis for combined theories. In: Proc. VMCAI '07. Volume 4349 of LNCS., Springer (2007) 378–394
5. Larraz, D., Rodríguez-Carbonell, E., Rubio, A.: SMT-based array invariant generation. In: Proc. VMCAI '13. Volume 7737 of LNCS., Springer (2013) 169–188
6. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Proc. VMCAI '04. Volume 2937 of LNCS., Springer (2004) 239–251
7. Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination of imperative programs using Max-SMT. In: Proc. FMCAD '13, IEEE (2013) 218–225
8. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using Max-SMT. In: Proc. CAV '14. Volume 8559 of LNCS., Springer (2014) 779–796
9. Borralleras, C., Lucas, S., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: SAT modulo linear arithmetic for solving polynomial constraints. Journal of Automated Reasoning **48**(1) (2012) 107–131
10. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Proc. IJCAR '12. Volume 7364 of LNCS., Springer (2012) 339–354
11. Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Minimal-model-guided approaches to solving polynomial constraints and extensions. In: Proc. SAT '14. Volume 8561 of LNCS., Springer (2014) 333–350
12. Cousot, P., Cousot, R.: Abstract interpretation : A unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In: Proc. POPL '77, ACM Press (1977) 238–252
13. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proc. POPL '78, ACM Press (1978) 84–96
14. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. 2nd International Symposium on Programming. (1976) 106–130
15. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Proc. PADO '01. Volume 2053. (2001) 155–172
16. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation **19**(1) (2006) 31–100
17. Clarisó, R., Cortadella, J.: The octahedron abstract domain. Science of Computer Programming **64**(1) (2007) 115–139
18. Menasche, M., Berthomieu, B.: Time petri nets for analyzing and verifying time dependent communication protocols. In: Protocol Specification, Testing, and Verification. (1983) 161–172

19. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Proc. International Workshop on Automatic Verification Methods for Finite State Systems. Volume 407 of LNCS., Springer (1989) 197–212
20. Yovine, S.: Model checking timed automata. In: Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems. Volume 1494 of LNCS., Springer (1996) 114–152
21. Brockschmidt, M., Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Compositional safety verification with Max-SMT. In: Proc. FMCAD '15, IEEE (2015) 33–40
22. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (June 1998)
23. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. 2nd edn. McGraw-Hill Higher Education (2001)
24. Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.W.: Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings. Journal of Automated Reasoning **35**(1-3) (oct 2005) 143–179
25. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables to problems with boolean variables. In: Proc. SAT'04. Volume 3542 of LNCS., Springer-Verlag (2005) 1–15
26. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing. Cambridge Univ. Press (1989)
27. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Annals of Mathematics and Artificial Intelligence **62**(3-4) (2011) 317–343
28. Sebastiani, R., Trentin, P.: OptiMathSAT: A tool for optimization modulo theories. In: Proc. CAV '15. Volume 9206 of LNCS., Springer (2015) 447–454
29. Bjørner, N., Phan, A., Fleckenstein, L.: νz - an optimizing SMT solver. In: Proc. TACAS '15. Volume 9035 of LNCS., Springer (2015) 194–199
30. Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The Barcelogic SMT Solver. In: Proc. CAV '08. Volume 5123 of LNCS. (2008) 294–298
31. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In: Proc. of SAT '06. Volume 4121 of LNCS., Springer (2006) 156–169
32. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)

A Proofs

Proof (of Theorem 3). We will reduce the problem of reachability of Petri nets with inhibitor arcs, which is known to be undecidable [32], to that in the statement of the theorem.

Let $\mathcal{N} = (P, T, W, I)$ be a Petri net with inhibitor arcs, where P is the set of places, T is the set of transitions, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is the flow relation, and $I \subseteq P \times T$ is the set of inhibitions. Let $m_0, m_f : P \rightarrow \mathbb{N}$ be the initial and final markings. The problem of reachability consists in determining whether m_f can be reached from m_0 in $\mathcal{N}$.

The dynamics of the Petri net can be captured by means of a difference program $\mathcal{P}$ as follows. There is a variable x_p for each place $p \in P$, meaning its number of tokens. In addition to a single initial location ℓ_0, there are two other locations ℓ, corresponding to the header of a single loop, and ℓ_{ERR}, corresponding to an error location. As regards transitions, there is a transition from ℓ_0 to ℓ that initializes the values of the variables according to the initial marking m_0. Then there is a transition from ℓ to ℓ for each of the transitions $t \in T$, which expresses the firing of t: availability of enough tokens is expressed with inequalities like $x_p \geq k$, inhibitor arcs with $x_p = 0$, and removing or adding tokens with $x'_p = x_p - k$ or $x'_p = x_p + k$, respectively. Finally there is a transition from ℓ to ℓ_{ERROR}, which is only executed when the values of the variables coincide with the final marking m_f. Since the resulting program may involve bound inequalities, the transformation from [15] described in Section 3 must be applied to obtain an equivalent difference program $\mathcal{P}$.

The proof concludes by noticing that m_f is reachable from m_0 in $\mathcal{N}$ if and only if ℓ_{ERROR} is reachable in the program $\mathcal{P}$, or in other words, if $\mathcal{P}$ is safe with respect to the assertion $(\ell_{\text{ERROR}}, \text{false})$.