HAL
open science

Applying abstract acceleration to (co-)reachability analysis of reactive programs

Peter Schrammel, Bertrand Jeannet

To cite this version:

Peter Schrammel, Bertrand Jeannet. Applying abstract acceleration to (co-)reachability analysis of reactive programs. Journal of Symbolic Computation, 2012, 47 (12), pp.1512-1532. 10.1016/j.jsc.2011.12.051 . hal-00753412

HAL Id: hal-00753412

https://inria.hal.science/hal-00753412v1

Submitted on 19 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Applying Abstract Acceleration to (Co-)Reachability Analysis of Reactive Programs

Peter Schrammel Bertrand Jeannet

INRIA Grenoble - Rhône-Alpes
655 Avenue de l'Europe
38330 Montbonnot St-Martin
France

Abstract

Acceleration methods are commonly used for computing precisely the effects of loops in the reachability analysis of counter machine models. Applying these methods on synchronous data-flow programs, *e.g.* LUSTRE programs, requires to deal with the non-deterministic transformations due to numerical input variables. In this article we address this problem by extending the concept of abstract acceleration of Gonnord et al. to numerical input variables. Moreover, we describe the dual analysis for co-reachability. We compare our method with some alternative techniques based on abstract interpretation pointing out its advantages and limitations. At last, we give some experimental results.

Key words: Static Analysis, Acceleration, Abstract Interpretation, Linear Relation Analysis.

1. Introduction

This article considers the reachability analysis of non-recursive, numerical programs represented by symbolic automata manipulating numerical variables, as illustrated in Fig. 1(a). More specifically, we focus on techniques enabling a precise analysis of self-loops that can be smoothly combined with methods for general numerical programs. For instance, considering the program of Fig. 1(a) with the set X_0 of initial values for the

⋆ This work was supported by the INRIA large-scale initiative SYNCHRONICS and the ANR project ASOPT.

Email addresses: `peter.schrammel@inria.fr` (Peter Schrammel), `bertrand.jeannet@inria.fr` (Bertrand Jeannet).

URLs: `http://pop-art.inrialpes.fr/people/schramme` (Peter Schrammel), `http://pop-art.inrialpes.fr/people/bjeannet` (Bertrand Jeannet).

Corrected version of article published in JSC 47(12) *19 November 2012*

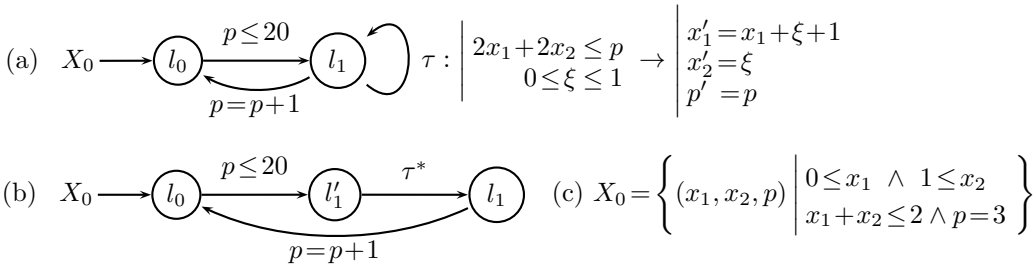


Fig. 1. Example program (a), transformed program (b) where τ^* denotes the transitive closure of the transition τ, and set of initial states (c).

state variables x_1, x_2, p, we want to compute the possible values of these state variables at locations l_0 and l_1. This article proposes *abstract acceleration methods* (as introduced by Gonnord and Halbwachs (2006)) to capture precisely the effect of the inner loop labelled by τ on a set of states $X \in \mathbb{R}^n$. It shows how these techniques integrate nicely with less precise but more general methods applying to programs with control structure involving nested loops and unstructured cycles.

Our ultimate motivation is the reachability analysis of data-flow synchronous programs manipulating Boolean and numerical variables, which are for instance specified with the LUSTRE language (Caspi et al. (1987)). Applications of such reachability analyses are for instance the verification of safety properties (Halbwachs et al. (1993)) or model-based testing (Jeannet et al. (2005)).

We first give an overview of existing methods for the reachability analysis of the systems we consider, before detailing the original contributions of this article.

Abstract interpretation and acceleration. Since the reachability problem is not decidable for numerical programs that encode two-counter automata (Minsky (1961)), two main approaches have been studied to overcome this fundamental limitation:

(1) Abstract interpretation techniques (Cousot and Cousot (1977, 1992)) always terminate with a sound over-approximation of the reachability set.
(2) Acceleration techniques (*e.g.* Leroux (2003); Bardin et al. (2003, 2005)) compute the exact reachability set for a restricted class of programs, *e.g.*, for programs with certain affine tests and assignments. However, there is no guarantee for termination.

In both approaches, the set of reachable states is obtained by solving iteratively an equation of the form $X = X_0 \cup post(X)$ where X is a set of states, X_0 the initial set, and *post* the postcondition operator associated with the program.

Abstract interpretation is a classical method for analyzing programs with infinite state spaces. The key idea is to approximate sets of states X by an element Y of an *abstract domain.* Two classical abstract domains for *numerical invariants* $X \in \wp(\mathbb{R}^n)$ are the domain of *convex polyhedra* $Pol(\mathbb{R}^n)$ (Cousot and Halbwachs (1978)), that are conjunctions of linear inequalities $\bigwedge_i(\boldsymbol{a_i x} \leq b_i)$ and the *linear congruences* domain (Granger (1991); Bagnara et al. (2006)) that represents conjunctions of linear congruences $\bigwedge_i(\boldsymbol{a_i x} = b_i \bmod c_i)$. An approximation of the reachable set is computed by solving iteratively the equation $Y = Y_0 \sqcup post(Y)$ in the abstract domain. In order to ensure termination when the abstract domain contains infinitely increasing chains, an extrapolation operator called *widening* is applied, which induces additional over-approximations.

The idea of *acceleration* is to accelerate cycles labelled by a function τ in the control structure of a program by computing the effect of its reflexive and transitive closure $\tau^* = \bigcup_{k \geq 0} \tau^k$ on a set of states X (the term "closure" refers to τ viewed as a relation). Applied to the program of Fig. 1(a), we obtain the program of Fig. 1(b). If the program is flat (*i.e.* it does not contain nested loops) and all loops can be accelerated, then the method is complete. If the program contains nested loops as in Fig. 1(a), the method is not complete any more; the standard heuristics is to enumerate and accelerate cycles (which form an infinite set) in the hope of terminating with the smallest fixed point after a finite number of steps. The same remark applies if transition functions in some cycles are too expressive to be accelerated. Acceleration has been mostly applied to automata manipulating integer variables using Presburger arithmetic (Fribourg and Olsén (1997); Finkel and Leroux (2002); Bardin et al. (2003)), or FIFO queues using subclasses of regular expressions (Boigelot and Godefroid (1997); Abdulla et al. (2004)).

Widening basically extrapolates the limit of a sequence of abstract invariants without referring to the program that generates them, whereas acceleration uses the structure of the program to perform an exact extrapolation. Gonnord and Halbwachs (2006) have proposed the concept of *abstract acceleration* which combines these approaches: wherever possible, simple loops are accelerated *in the abstract domain*, and in any other cases (multiple self-loops, nested loops, too expressive transitions) one resorts to the use of widening to guarantee the convergence of the fixed point computation at the cost of over-approximations.

Applying acceleration to reactive programs. Many acceleration techniques such as those introduced by Leroux (2003); Bardin et al. (2003); Gonnord and Halbwachs (2006) consider automata with transition functions in the form of guarded actions

$$\tau : \underbrace{g(\boldsymbol{x})}_{guard} \rightarrow \underbrace{\boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x})}_{action} \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^n \tag{1}$$

However, reactive programs such as LUSTRE data-flow programs interact with their environment: at each computation step they have to take into account the value of *input* variables, which typically correspond to values acquired by sensors.

Boolean input variables can be encoded in an automaton by finite non-deterministic choices but numerical input variables demand a more specific treatment. Indeed, they induce transition functions of the form

$$\tau : g(\boldsymbol{x}, \boldsymbol{\xi}) \rightarrow \boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\xi}) \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^n \quad \boldsymbol{\xi} \in \mathbb{R}^p \tag{2}$$

that depends on both state variables $\boldsymbol{x}$ and numerical input variables $\boldsymbol{\xi}$. This article addresses specifically this point, by extending the abstract acceleration concept introduced by Gonnord and Halbwachs (2006) to systems with numerical inputs. The methods developped in this article can also be seen as an alternative to acceleration methods-based on the computation of transitive closures of affine *relations* (rather than functions) (Beletska et al. (2009); Bozga et al. (2010)). Indeed, transition functions with inputs can be viewed as relations between states defined as

$$R(\boldsymbol{x}, \boldsymbol{x'}) = \big(\exists \boldsymbol{\xi} : g(\boldsymbol{x}, \boldsymbol{\xi}) \wedge \boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\xi})\big) \tag{3}$$

Contributions and outline. Our first two contributions are extensions of the abstract acceleration concept as introduced by Gonnord and Halbwachs (2006) in two directions:

(i) We consider open systems, *i.e.*, with numerical inputs, instead of closed systems; in particular we show how to accelerate loops where translations and resets depend on *inputs*, provided that the guard of the loop constrains separately state and input variables.

(ii) We also extend abstract acceleration techniques from forward (reachability) analysis to backward (co-reachability) analysis. In consequence, it possible to apply the abstract acceleration concept to related, but slightly different problems in verification, such as parameter synthesis for example. Moreover, experience shows that combining forward and backward analyses is very useful in the context of abstract interpretation (*e.g.* Jeannet (2003)).

A third contribution is (iii) a detailed comparison of the abstract acceleration approach to the derivative closure approach of Ancourt et al. (2010), which is related to methods based on transitive closures of relations. This article extends the results presented by Schrammel and Jeannet (2010) with the contributions (ii) and (iii).

After some preliminaries in Section 2 about the considered program model, operations on convex polyhedra and the general analysis framework that we use for verification, we recall the main results of abstract acceleration in Section 3. Section 4 extends abstract acceleration techniques to systems with numerical input variables, in the context of forward analysis. Section 5 extends these results to backward analysis. Section 6 is dedicated to a comparison with other methods. Section 7 gives an overview and some experimental results on how to apply abstract accleration to logico-numerical programs. After a discussion of further related work in Section 8, we conclude in Section 9.

2. Analysis of Logico-Numerical Programs

Program model. We assume a set V of variable names. We consider in this article programs modeled as numerical automata $(L, \boldsymbol{x}, \boldsymbol{\xi}, l_0, X_0, \mathcal{T})$ where

- L is a finite set of locations, $\boldsymbol{x} \in V^n$ a vector of real-valued *state* variables, and $\boldsymbol{\xi} \in V^p$ a vector of real-valued *input* variables;
- l_0 is the initial location and $X_0 \subseteq \mathbb{R}^n$ is the set of initial values for state variables $\boldsymbol{x}$ at location l_0;
- $\mathcal{T}$ is a finite set of transitions of the form $t : (l, l', \tau)$ where l and l' are respectively the origin and destination locations, and τ is a (partial) transition function of the form $g(\boldsymbol{x}, \boldsymbol{\xi}) \to \boldsymbol{x'} = f(\boldsymbol{x}, \boldsymbol{\xi})$.

An execution of such a system is a sequence $(l_0, \boldsymbol{x}_0) \xrightarrow{t_0, \boldsymbol{\xi}_0} \ldots (l_k, \boldsymbol{x}_k) \xrightarrow{t_k, \boldsymbol{\xi}_k} \ldots$ such that $\boldsymbol{x}_0 \in X_0$ and for any $k \geq 0$, $t_k = (l_k, l_{k+1}, \tau_k)$ and $\boldsymbol{x}_{k+1} = \tau_k(\boldsymbol{x}_k, \boldsymbol{\xi}_k)$.

This program model includes various models of counter automata (Comon and Jurski (1998); Bardin et al. (2005)). It can also be obtained from LUSTRE synchronous data-flow programs by (1) taking the output of their front-end compilation process, (2) performing on this output a partial evaluation (Jones et al. (1993)) of all Boolean state variables (which are then encoded in control locations), and (3) eliminating Boolean input variables using non-deterministic choices. The partition refinement mechanics implemented in the NBAC tool (Jeannet (2003)) are capable of achieving this task and have been employed for connecting the ASPIC tool (Gonnord (2007, 2009)) to LUSTRE, for example.

Convex Polyhedra. We use in this article the abstract domain of convex polyhedra for representing invariants on numerical variables. Convex polyhedra can be represented either as a conjunction of constraints denoted by

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ with } \boldsymbol{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$$

or as the set of vectors generated by a vector of vertices $\mathbf{V} = (\boldsymbol{v}_1 \dots \boldsymbol{v}_p) \in \mathbb{R}^{n\times p}$ and rays $\mathbf{R} = (\boldsymbol{r}_1 \dots \boldsymbol{r}_q) \in \mathbb{R}^{n\times q}$, denoted by (V, R):

$$\{\boldsymbol{x} \mid \exists \boldsymbol{\lambda}, \boldsymbol{\mu} \geq 0 \ : \ \textstyle\sum_i \lambda_i = 1 \wedge \boldsymbol{x} = \mathbf{V}\boldsymbol{\lambda} + \mathbf{R}\boldsymbol{\mu}\}$$

We can convert from one representation to the other one (see (Fukuda and Prodon (1996)). We will use the same notation for polyhedra X interexchangeably for both the predicate $X(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ and the set $X = \{\boldsymbol{x} \mid \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. We briefly remind some classical operations described *e.g.* by Halbwachs et al. (1997):

The constraint representation is used for computing the intersection $\sqcap$ of two polyhedra (*i.e.* the conjunction of their constraints) and the inverse image $\tau^{-1}(X)$ of a polyhedron $X = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ by an affine assignment $\tau : \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$, *i.e.* $\tau^{-1}(X) = (\mathbf{AC}\boldsymbol{x} \leq \boldsymbol{b} - \mathbf{A}\boldsymbol{d})$.

The generator representation is used for computing the convex hull (union) $\sqcup$ of two polyhedra (*i.e.* the union of their generators), the projection of dimensions, and the image $\tau(X)$ of a polyhedron $X = (V, R)$ by an affine assignment $\tau : \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$: $\tau(X) = \big((\mathbf{CV} + (\underbrace{\boldsymbol{d} \dots \boldsymbol{d}}_{p \text{ times}})), \mathbf{CR}\big)$.

Mind that the intersection of two convex polyhedra is again a convex polyhedron: $X \cap Y = X \sqcap Y$. In contrast, the union of two convex polyhedra is not convex in general. For that reason the convex hull, *i.e.* an over-approximation, is used instead of the ordinary union for sets: $X \cup Y \subseteq X \sqcup Y$

The *time elapse* operation, defined as $X \nearrow D = \{\mathbf{x} + t\mathbf{d} \mid \mathbf{x} \in X, \mathbf{d} \in D, t \in \mathbb{R}^{\geq 0}\}$ can be implemented using the systems of generators (V_X, R_X) and (V_D, R_D) of the polyhedra X and D: $(V_X, R_X \cup V_D \cup R_D)$ is a system of generators for $X \nearrow D$.

The *Minkowski sum* (see de Berg et al. (2008)) of two polyhedra $X = X_1 + X_2$ is defined by $X(\boldsymbol{x}) = \exists \boldsymbol{x}_1, \boldsymbol{x}_2 \ : \ (\boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2) \wedge X_1(\boldsymbol{x}_1) \wedge X_2(\boldsymbol{x}_2)$.

$\top$ and $\bot$ denote the polyhedra $\mathbb{R}^n$ and $\emptyset$ respectively.

Concerning the complexity, the translation between the two representations may be exponential in the number of dimensions n (see Fukuda and Prodon (1996)), hence any composition of basic operations requiring both representations has a worst-case exponential complexity.

Note that the operations defined on generators can also be computed using only systems of constraints and projection as shown by Benoy et al. (2005). However, this does not allow to improve the worst-case complexity.

3. Overview of Acceleration and Abstract Acceleration

As mentioned in the introduction, the idea of *acceleration* (Fig. 1) is to replace a self-loop τ by an ordinary transition τ^* that is the reflexive and transitive closure of τ. *Abstract acceleration* introduced by Gonnord and Halbwachs (2006) and Gonnord (2007) relaxes exact acceleration in the sense that it aims at approximating the exact set $\tau^*(X)$ by a convex polyhedron $\tau^{\otimes}(X) \supseteq \tau^*(X)$ that is close to the convex hull of the exact

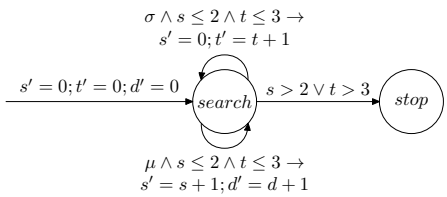


Fig. 2. Example: A robot car safety controller. The lower loop means: When a "meter" signal μ is received the speed estimation s and the distance d are incremented: this transition is a translation. On the contrary, the upper loop is a translation with resets: when a "second" signal σ is received the speed estimation is reset and the time t is incremented. If the speed is above $2m/s$ or no input signal has been received for $3s$ then the car is stopped.

set. This method is also inspired by the time elapse operator used in timed or in hybrid automata (Halbwachs et al. (1997)).

Following the notations of Section 2, a self-loop τ has the structure: $G \rightarrow A$ meaning "while guard G do action A". Generally, acceleration methods for numerical variables $\boldsymbol{x}$ deal with transitions of the form

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d} \tag{4}$$

where $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}$ represents a conjunction of linear constraints defining a convex polyhedron, and $\boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ is an affine transformation; $\mathbf{C}$ is a square matrix. A transition is called

- a *reset* iff $\mathbf{C}$ is the zero matrix,
- a *translation* iff $\mathbf{C}$ is the identity matrix,
- a *translation with resets* (or translation/reset) iff $\mathbf{C}$ is a diagonal matrix with zeros and ones only,
- a *periodic affine transformation* iff $\exists p > 0, \exists l > 0 : \mathbf{C}^{p+l} = \mathbf{C}^p$,
- a *general affine transformation* otherwise.

Existing acceleration methods do not address general affine transformations. We will not consider in this paper the case of periodic affine transformations, but we discuss the generalization of our results to this case in the conclusion.

Fig. 2 shows an example of a simplistic safety controller for a robot car (cf. Halbwachs et al. (1997); Ancourt et al. (2010)). The car should follow a given track, but in case it has lost the track it should keep on searching the track for a while before being stopped. This example illustrates two of the above types of transitions.

In the context of abstract acceleration, Gonnord and Halbwachs (2006) show that translations and translations with resets can be accelerated as follows:

Theorem 1 (Translations, Gonnord (2007))**.** *Let τ be a translation $G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}$, then for every convex polyhedron X, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \Big(\big((X \sqcap G) \nearrow \{\boldsymbol{d}\} \big) \sqcap (G + \{\boldsymbol{d}\}) \Big)$$

is a convex over-approximation of $\tau^*(X)$.

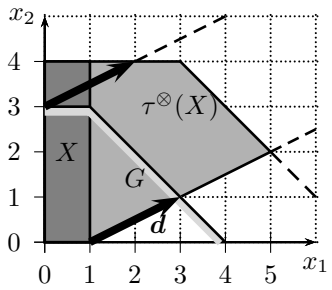


Fig. 3. Abstract acceleration of a *translation* (Ex. 1) by vector $\boldsymbol{d}$ starting from X (dark gray) resulting in $\tau^{\otimes}(X)$ (whole shadowed area).

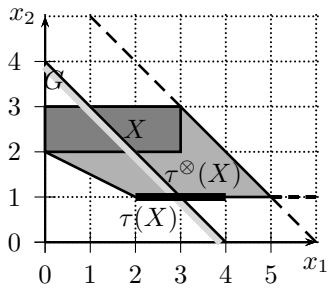


Fig. 4. Abstract acceleration of a *translation with resets* (Ex. 2) starting from X (dark gray): $\tau(X)$ (bold line) and result $\tau^{\otimes}(X)$ (whole shadowed area).

Example 1 (Translation)**.** (see Fig. 3)

$$\tau : \underbrace{x_1+x_2 \leq 4 \ \wedge \ x_2 \leq 3}_{G} \rightarrow \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} 2 \\ 1 \end{pmatrix}}_{\boldsymbol{d}}$$

Starting from $X = (0 \leq x_1 \leq 1 \ \wedge \ 0 \leq x_2 \leq 4)$ we compute $\tau^{\otimes}(X)$:

$$\begin{aligned}
X \sqcap G &= (0 \leq x_1 \leq 1 \ \wedge \ 0 \leq x_2 \leq 3) \\
(X \sqcap G) \nearrow \{\boldsymbol{d}\} &= \begin{cases} x_1 \geq 0 \ \wedge \ x_2 \geq 0 \ \wedge \ x_1 - 2x_2 \geq -6 \ \wedge \\ -x_1 + 2x_2 \geq -1) \end{cases} \\
((X \sqcap G) \nearrow \{\boldsymbol{d}\}) \sqcap (G + \{\boldsymbol{d}\}) &= \begin{cases} x_1 \geq 0 \ \wedge \ 0 \leq x_2 \leq 4 \ \wedge \ x_1 - 2x_2 \geq -6 \ \wedge \\ -x_1 + 2x_2 \geq -1 \ \wedge \ x_1 + x_2 \leq 7 \end{cases} \\
\tau^{\otimes}(X) &= \begin{cases} x_1 \geq 0 \ \wedge \ 0 \leq x_2 \leq 4 \ \wedge \ -x_1 + 2x_2 \geq -1 \ \wedge \\ x_1 + x_2 \leq 7 \end{cases}
\end{aligned}$$

Theorem 2 (Translations with resets, Gonnord (2007))**.** *Let τ be a translation with resets $G \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$, then for every convex polyhedron X, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \Big(\big((\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\} \big) \sqcap (G + \{\mathbf{C}\boldsymbol{d}\}) \Big)$$

is a convex over-approximation of $\tau^*(X)$.

Example 2 (Translation with resets)**.** (see Fig. 4)

$$\tau : x_1 + x_2 \leq 4 \ \wedge \ \begin{cases} x'_1 = x_1 + 2 \\ x'_2 = 1 \end{cases}$$

Starting from $X = (0 \leq x_1 \leq 3 \ \wedge \ 2 \leq x_2 \leq 3)$ we compute $\tau^{\otimes}(X)$:

$$\begin{aligned}
\tau(X) &= (2 \leq x_1 \leq 4 \ \wedge \ x_2 = 1) \\
\tau(X) \sqcap G &= (2 \leq x_1 \leq 3 \ \wedge \ x_2 = 1) \\
(\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\} &= (x_1 \geq 2 \ \wedge \ x_2 = 1) \\
((\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\}) \sqcap (G + \{\mathbf{C}\boldsymbol{d}\}) &= (2 \leq x_1 \leq 5 \ \wedge \ x_2 = 1) \\
\tau^{\otimes}(X) &= \begin{cases} x_1 \geq 0 \ \wedge \ 1 \leq x_2 \leq 3 \ \wedge \ x_1 + 2x_2 \geq 4 \ \wedge \\ x_1 + x_2 \leq 6 \end{cases}
\end{aligned}$$

Remark 1. Theorem 2 exploits the property that a translation with resets to *constants* iterated N times is equivalent to the same translation with resets, followed by a pure translation iterated $N-1$ times. Hence the structure of the obtained formula

Remark 2. Ideally, $\tau^{\otimes}(X)$ as defined in Theorems 1 and 2 should be the best over-approximation of $\tau^*(X)$ by a convex polyhedron. This is not the case as shown by the following example in one dimension. Let $X = [1, 1]$ and $\tau : x_1 \leq 4 \rightarrow x_1' = x_1 + 2$. $\tau^{\otimes}(X) = [1, 6]$, whereas the best convex over-approximation of $\tau^*(X) = \{1, 3, 5\}$ is the interval $[1, 5]$. This is because the operations involved in the definition of $\tau^{\otimes}(X)$ manipulate dense sets and do not take into account arithmetic congruences. In this article we will not improve in this respect, but we will point out in our proofs where this *dense* approximation takes place, and we discuss in Section 9 how the linear congruences abstract domain mentioned in the introduction can be exploited to improve on this point.

4. Abstract Acceleration with Numerical Inputs

We now extend numerical abstract acceleration by numerical input variables $\boldsymbol{\xi}$. We consider transitions τ of the form [1]

$$\underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{L} \\ \mathbf{0} & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix}}_{\mathbf{A}\boldsymbol{x}+\mathbf{L}\boldsymbol{\xi} \leq \boldsymbol{b} \ \wedge \ \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}} \rightarrow \boldsymbol{x}' = \underbrace{\begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x}+\mathbf{T}\boldsymbol{\xi}+\boldsymbol{u}} \tag{5}$$

General and simple guards. The following proposition shows the challenge raised by adding inputs:

Proposition 1 (General guards and general affine transformations)**.** Any general affine transformation without inputs $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ can be expressed

– as a "reset with inputs" $(\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{\xi} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}) \rightarrow \boldsymbol{x}' = \boldsymbol{\xi}$,

– as well as a "translation with inputs" $(\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{\xi} = (\mathbf{C} - \mathbf{I})\boldsymbol{x} + \boldsymbol{d}) \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\xi}$.

This means that there is no hope to get precise acceleration for such resets or translations with inputs, unless we know how to accelerate precisely general affine transformations without inputs, which is out of the scope of the current state of the art.

[1] Note that the $\mathbf{0}$ in the matrix of the guard does not imply a loss of generality.

Nevertheless, we can accelerate transitions with inputs if the constraints on the state variables do not depend on the inputs, *i.e.*, the guard is of the form $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}$, *i.e.*, when $\mathbf{L} = 0$ in Eqn. (5). We call the resulting guards *simple guards*.

In the following, we show how to accelerate translations and translations with resets with simple guards. We provide in §4.3 an over-approximation of our results for *general guards*.

4.1. Translations with Inputs and Simple Guards

Definition 1. Translations with inputs and simple guards are defined as

$$\underbrace{\underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix}}_{\underbrace{\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}}_{G}} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}} \rightarrow \boldsymbol{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} + \boldsymbol{u}}_{\boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}}$$

The first step we perform is to reduce such a translation with inputs to a *polyhedral translation* $\tau : G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + D$ and defined by $\tau(X) = (X \sqcap G) + D$.

Proposition 2 (Translation with inputs = polyhedral translation)**.** A translation τ with inputs and a simple guard (Def. 1) is equivalent to a polyhedral translation defined by

$$G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + D \quad \text{with} \quad D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{J}\boldsymbol{\xi} \leq \boldsymbol{k}\}$$

(D can be computed by standard polyhedra operations.)

Proof.

$$\begin{aligned}
& \boldsymbol{x}' \in \tau(X) \\
\iff & \exists \boldsymbol{x} \in X, \exists \boldsymbol{\xi} \ : \ \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{x}' = \boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u} \\
\iff & \exists \boldsymbol{x} \in X \sqcap G, \exists \boldsymbol{\xi}, \exists \boldsymbol{d} \ : \ \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d} \\
\iff & \exists \boldsymbol{x} \in X \sqcap G, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d} \\
& \qquad\qquad \text{with } D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} \ : \ \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}\}
\end{aligned}$$

□

We now generalize Thm. 1 from ordinary translations to polyhedral translations.

Proposition 3 (Polyhedral translation)**.** Let τ be a polyhedral translation $G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + D$. Then, the set

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$$

is a convex over-approximation of $\tau^*(X)$.

Proof. $\boldsymbol{x}' \in \bigcup_{k \geq 1} \tau^k(X) \iff \boldsymbol{x}' \in \tau(\bigcup_{k \geq 0} \tau^k(X))$

$$\iff \exists k \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \ldots \boldsymbol{d}_k \in D \ : \ \begin{cases} \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \\ \boldsymbol{x}_k = \boldsymbol{x}_0 + \sum_{j=1}^{k} \boldsymbol{d}_j \\ G(\boldsymbol{x}_0) \ \wedge \ \forall k' \in [1, k] \ : \ G(\boldsymbol{x}_0 + \sum_{j=1}^{k'} \boldsymbol{d}_j) \end{cases}$$

$\iff \exists k \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x_0} + k\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0) \ \wedge \ G(\boldsymbol{x}_k)$

(because D and G are convex, see Rem. 3)

$\implies \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0)$

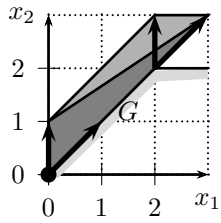
(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)

$\iff \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{x}_k \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k \in (\{\boldsymbol{x}_0\} \nearrow D)$

$\iff \boldsymbol{x}' \in \tau((X \sqcap G) \nearrow D) \quad \square$

Mind that the only approximation takes place in the line ($\Rightarrow$) where the integer coefficient $k \geq 0$ is replaced by a real coefficient $\alpha \geq 0$. This is the technical explanation of Rem. 2.

Remark 3 (Convexity argument)**.** For any k loop iterations with $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in D$ s.t. $\forall k' \in [0, k] : G(\boldsymbol{x} + \sum_{j=1}^{k'} \boldsymbol{d}_j)$ we have $\exists \boldsymbol{d} \in D, \exists \alpha \geq 0 : G(\boldsymbol{x}) \wedge G(\boldsymbol{x} + \alpha\boldsymbol{d})$ s.t. $\sum_{j=1}^{k} \boldsymbol{d}_j = \alpha\boldsymbol{d}$: any intermediate point $\boldsymbol{x} + \alpha\boldsymbol{d}$ must be in G, because G is convex; moreover $\alpha \geq 0$ and a vector $\boldsymbol{d} \in D$ actually exist: for example, take $\alpha = k$ and $\boldsymbol{d} = \frac{1}{k} \sum_{j=1}^{k} \boldsymbol{d}_j$, which is in D because D is convex.



Remark 4 (Inputs vs. constants)**.** One might think that Thm. 1 can be applied directly by accelerating the transition for each $\boldsymbol{d} \in D$ and taking the union, *i.e.* computing $\tau^{\otimes}(X)$ by $X \sqcup \bigsqcup_{\boldsymbol{d} \in D} X_{\boldsymbol{d}}$ with $X_{\boldsymbol{d}} = \big((X \sqcap G) \nearrow \{\boldsymbol{d}\}\big) \sqcap (G + \{\boldsymbol{d}\})$. However, this formula is not correct for the last step beyond the guard, which is illustrated in the figure on the right-hand side for a polyhedral translation with $X = \{(0,0)\}$, $D = (0 \leq d_1 \leq 1 \wedge d_2 = 1)$ and the guard G given in the figure. For the step crossing the guard, *e.g.* at $(2, 2)$, there is actually a choice among all values in D (correct abstract acceleration: whole shadowed area), whereas the wrong acceleration (dark gray) considers only the vector $\boldsymbol{d} = (1, 1)$ which led to $(2, 2)$.

We combine Propositions 2 and 3 to formulate the following theorem:

Theorem 3 (Translation with inputs and simple guards)**.** *Let τ be a translation with inputs and a simple guard*

$$\tau : (\underbrace{\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}}_{G}) \wedge (\mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}) \quad \rightarrow \quad \boldsymbol{x}' = \boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}$$

Then, the set

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$$

with $D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{J}\boldsymbol{\xi} \leq \boldsymbol{k}\}$ is a convex over-approximation of $\tau^*(X)$.

Proof. Follows directly from Prop. 2 and 3. $\square$

Example 3 (Translation with inputs and simple guards)**.** Consider the polyhedron $X = \{(x_1, x_2) \mid 0 \leq x_1 \leq x_2 \leq 1\}$ and the transition

$$\tau : \left| \begin{array}{r} x_1 + x_2 \ \leq \ 4 \\ 1 \leq \xi \ \leq \ 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$$

Eliminating the inputs as in Proposition 2 yields $D = \{(d_1, d_2) \,|\, 1 \leq d_1 \leq 3 \wedge -d_1 + 2d_2 = 1\}$, see Fig. 5 left-hand side. After translation of X by D (Fig. 5 right-hand side) we obtain the polyhedron $\{(x_1, x_2) \,|\, x_1 \geq 0 \wedge -x_1 + x_2 \leq 1 \wedge x_1 + x_2 \leq 9 \wedge -2x_1 + 4x_2 \leq 9 \wedge 2x_1 - 3x_2 \leq 0\}$.

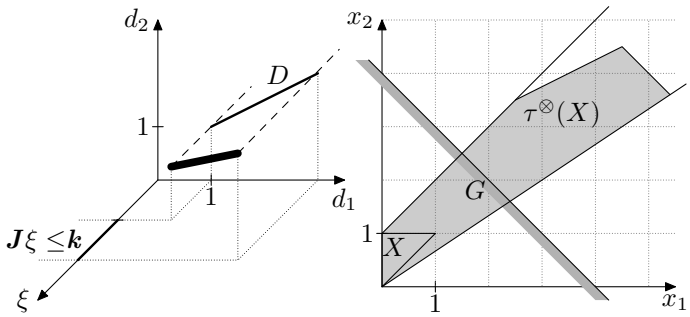


Fig. 5. Translation with inputs (Ex. 3): The left-hand side shows the computation of D: $\mathbf{J}\xi \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\xi + \boldsymbol{u}$ (bold line) is projected on variables $\boldsymbol{d}$ to obtain D. The shadowed area in the right-hand side figure is $\tau^{\otimes}(X)$.

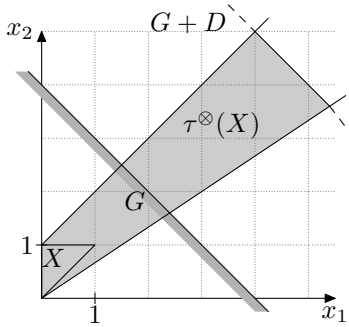


Fig. 6. Precision loss in example 3 when using the approximate formula according to Rem. 5.

Remark 5 (Alternative, less precise formula)**.** In analogy to Thm. 1, we could alternatively consider the formula

$$X \sqcup ((((X \sqcap G) \nearrow D) \sqcap (G + D)).$$

In order to justify this, we extend the proof of Proposition 3 by continuing at the label (dense approximation):

$\iff \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{x}_k, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_k + \boldsymbol{d}' \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha \boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_k)$

$\iff \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha \boldsymbol{d} + \boldsymbol{d}' \ \wedge \ G(\boldsymbol{x}' - \boldsymbol{d}')$

$\implies \big(\exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha \boldsymbol{d} + \boldsymbol{d}'\big) \ \wedge \ \big(\exists \boldsymbol{d}' \in D : G(\boldsymbol{x}' - \boldsymbol{d}')\big)$

$\iff \big(\exists \alpha' \geq 1, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}'' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha' \boldsymbol{d}''\big) \ \wedge \ \big(\exists \boldsymbol{d}' \in D : G(\boldsymbol{x}' - \boldsymbol{d}')\big)$

$\implies \boldsymbol{x}' \in (X \sqcap G) \nearrow D \ \wedge \ \boldsymbol{x}' \in (G + D)$

using $\{\boldsymbol{x} \mid \exists \boldsymbol{d} \in D \ \wedge \ G(\boldsymbol{x} - \boldsymbol{d})\} = \{\boldsymbol{z} + \boldsymbol{d} \mid \boldsymbol{d} \in D \wedge G(\boldsymbol{z})\} = (G + D)$. It can be observed that for the translation of example 3 the latter formula results in an over-approximation (see Fig. 6) as compared to the result in Fig. 5. This reflects the additional approximation steps in the proof indicated by ($\implies$).

4.2. Translations/Resets with Inputs and Simple Guards

Definition 2. Translations/resets with inputs and simple guards are defined as

$$\underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix}}_{\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \ \wedge \ \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}} \rightarrow \boldsymbol{x}' = \underbrace{\begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}}$$

where $\mathbf{C}$ is a diagonal matrix with $\mathbf{C}_{i,i} \in \{0, 1\}$ for all i.

Notations. Let $\mathbf{C}' = \mathbf{I} - \mathbf{C}$, then we can decompose any vector $\boldsymbol{x}$ in $\mathbf{C}\boldsymbol{x} + \mathbf{C}'\boldsymbol{x}$. We denote $\mathbf{C}\boldsymbol{x} = \boldsymbol{x}^{t,0}$ a vector where the reset dimensions are set to zero, and $\mathbf{C}'\boldsymbol{x} = \boldsymbol{x}^{0,r}$ a vector where the translated dimensions are set to zero. We extend such notations to sets: $X^{t,0} = \{\boldsymbol{x}^{t,0} \mid \boldsymbol{x} \in X\}$ and $X^{0,r} = \{\boldsymbol{x}^{0,r} \mid \boldsymbol{x} \in X\}$. We use a similar notation for projection: $X^{t,\bullet} = \{\boldsymbol{x} \mid \boldsymbol{x}^{t,0} \in X^{t,0}\}$ and $X^{\bullet,r} = \{\boldsymbol{x} \mid \boldsymbol{x}^{0,r} \in X^{0,r}\}$ denote the sets obtained by existential quantification of the reset (resp. translated) dimensions.

Observe that the over-approximation $X^{t,\bullet} \sqcap X^{\bullet,r}$ of a set X by the cartesian product w.r.t. to translated and reset dimensions is equal to the Minkowski sum $X^{t,0} + X^{0,r}$.

The case of translations/resets with inputs can be handled similarly to translations: we combine Proposition 2 and Thm. 2 to reduce translations/resets with inputs to *polyhedral translations with resets* $\tau : G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$ defined by $\tau(X) = (X \sqcap G)^{t,0} + D$.

Mind, however, that Rem. 1 does not apply any more and cannot be exploited in the presence of inputs, because the variables being reset may be assigned a different value in each iteration.

Proposition 4 (Polyhedral translation with resets)**.** Let τ be a polyhedral translation with resets $G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$. Then, the set

$$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big)$$

is a convex over-approximation of $\tau^*(X)$.

In the formula above and in the proof below, we unfold τ twice, that is, we accelerate only the central part of the sequence $\boldsymbol{x} \xrightarrow{\tau} \boldsymbol{x}_0 \dots \boldsymbol{x}_n \xrightarrow{\tau} \boldsymbol{x'}$ with $\boldsymbol{x} \in X$ because we have $\forall k \in [0, n] : \boldsymbol{x_k} \in G \sqcap D^{\bullet,r}$, whereas we only have $\boldsymbol{x} \in G$ at the start-point, and $\boldsymbol{x'} \in D^{\bullet,r}$ at the end-point.

Proof. The formula is trivially correct for 0 or 1 iterations of the self-loop τ. It remains to show that, for the case of $k \geq 2$ iterations, our formula yields an over-approximation of $\bigcup_{k\geq 2} \tau^k(X)$.

$$\boldsymbol{x}' \in \bigcup_{k\geq 2} \tau^k(X) \iff \boldsymbol{x}' \in \tau\left(\bigcup_{k\geq 0} \tau^k(\tau(X))\right)$$

$$\iff \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$$

$$\begin{cases} \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \\ \wedge\ \forall k' \in [1, k] \ : \ \begin{cases} \boldsymbol{x}^i_{k'} = \boldsymbol{x}^i_0 + \sum_{j=1}^{k'} \boldsymbol{d}^i_j & \text{for } i \in I^t \\ \boldsymbol{x}^i_{k'} = \boldsymbol{d}^i_{k'} & \text{for } i \in I^r \end{cases} \\ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \\ \wedge\ \forall k' \in [0, k] \ : \ G(\boldsymbol{x}_{k'}) \end{cases}$$

$$\iff \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$$

$$\begin{cases} \forall k' \in [1, k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}^{t,0}_0 + (\sum_{j=1}^{k'} \boldsymbol{d}^{t,0}_j) + \boldsymbol{d}^{0,r}_{k'} \\ \wedge\ \forall k' \in [0, k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

$$\implies \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0}_1 \dots \boldsymbol{d}^{t,0}_k \in D^{t,0}, \exists \boldsymbol{d}^{0,r}_1 \dots \boldsymbol{d}^{0,r}_k \in D^{0,r} \ :$$

$$\begin{cases} \forall k' \in [1, k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}^{t,0}_0 + (\sum_{j=1}^{k'} \boldsymbol{d}^{t,0}_j) + \boldsymbol{d}^{0,r}_{k'} \\ \wedge\ \forall k' \in [0, k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

(D approximated by the sum $(D^{t,0} + D^{0,r})$)

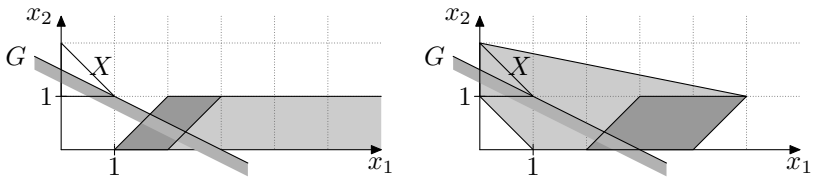


Fig. 7. Translation/reset with inputs: Ex. 4. Left-hand side: $\tau(X)$ (dark shadowed) and $((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}) + D^{0,r}$ (whole shadowed area). Right-hand side: $\tau(((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}) + D^{0,r})$ (dark shadowed) and $\tau^{\otimes}(X)$ (whole shadowed area).

$$\iff \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_k^{0,r} \in D^{0,r} \ :$$

$$\begin{cases} \quad \boldsymbol{x}_k = \boldsymbol{x}_0^{t,0} + k\boldsymbol{d}^{t,0} + \boldsymbol{d}_k^{0,r} \\ \wedge\ G(\boldsymbol{x}_0)\ \wedge\ G(\boldsymbol{x}_k) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x})\ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

(because $D^{t,0}, D^{0,r}$ and G are convex and $\boldsymbol{x}_0^{0,r} \in D^{0,r}$)

$$\implies \exists \alpha \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_k^{0,r} \in D^{0,r} \ :$$

$$\begin{cases} \wedge\ \boldsymbol{x}_k = \boldsymbol{x}_0^{t,0} + \alpha\boldsymbol{d}^{t,0} + \boldsymbol{d}_k^{0,r} \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x})\ \wedge\ G(\boldsymbol{x}_0)\ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

(dense over-approximation; $G(\boldsymbol{x}_k)$ already implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)

$$\iff \boldsymbol{x}' \in \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big) \quad \square$$

Theorem 4 (Translation with resets, inputs and simple guards)**.** *The accelerated transition $\tau^{\otimes}$ for a translation/reset with inputs and a simple guard τ can be computed by applying Proposition 4 with D defined as in Proposition 2.*

Example 4 (Translation with resets, inputs and simple guards)**.** Consider the polyhedron $X = \{(x_1, x_2) \mid 0 \leq x_1 \ \wedge\ 1 \leq x_2 \ \wedge\ x_1 + x_2 \leq 2\}$ and the transition

$$\tau : \left| \begin{array}{c} x_1 + 2x_2 \ \leq\ 3 \\ 0 \leq \xi \ \leq\ 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \end{array} \right.$$

Eliminating the inputs yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \ \wedge\ d_1 - d_2 = 1\}$ and $D^{t,0} = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \ \wedge\ d_2 = 0\}$. We obtain $\tau^{\otimes}(X) = \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \ \wedge\ x_2 \geq 0 \ \wedge\ x_1 - x_2 \leq 4 \ \wedge\ x_1 + 5x_2 \leq 10 \ \wedge\ x_1 \geq 0\}$, see Fig. 7.

4.3. Relaxing General Guards to Simple Guards

As discussed at the beginning of §4, allowing constraints that relate state variables with input variables in guards, *i.e.* $G = \mathbf{A}\boldsymbol{x} + \mathbf{L}\boldsymbol{\xi} \leq \boldsymbol{b} \ \wedge\ \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}$ with $\mathbf{L} \neq 0$ (see Eqn. (5)), makes acceleration very difficult (Prop. 1). Our solution is to relax the guard G to a simple guard (or cartesian product) $\overline{G} = \underbrace{(\exists \boldsymbol{\xi} : G)}_{\mathbf{A}'\boldsymbol{x} \leq \boldsymbol{b}'} \wedge \underbrace{(\exists \boldsymbol{x} : G)}_{\mathbf{J}'\boldsymbol{\xi} \leq \boldsymbol{k}'}$.

We can now apply the accelerated transition from Theorems 3 and 4 with $G' =$

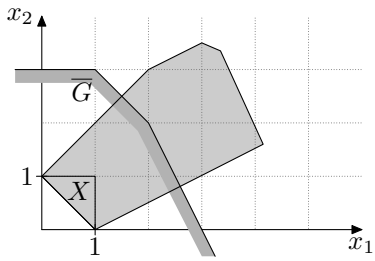


Fig. 8. Ex. 5: accelerated transition $\tau^{\otimes}(X)$ using the relaxed guard $\overline{G}$ (result shadowed).

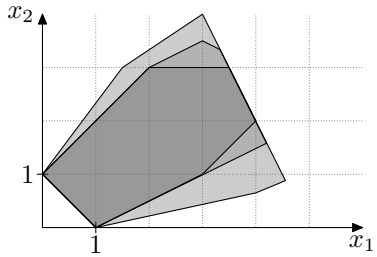


Fig. 9. Ex. 5: comparison between convex hull of the exact result (dark gray), our method (gray), and widening with no delay and 3 descending iterations (light gray).

$(\mathbf{A}'\boldsymbol{x} \leq \boldsymbol{b'})$ and $D' = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T\xi} + \boldsymbol{u} \wedge \boldsymbol{J'\xi} \leq \boldsymbol{k'}\}$. This trivially results in a sound over-approximation because a weaker guard is used for abstract acceleration.

Note however that in the corresponding acceleration formulas, we can still compute exactly the function τ using the original guard G. Indeed, the proofs of those theorems are not based on the assumption $\mathbf{L} \neq 0$ when they introduce the function τ.

Example 5 (Relaxed guard)**.** Consider the polyhedron $X = \{(x_1, x_2) \mid x_1 \leq 1 \wedge x_2 \leq 1 \wedge x_1 + x_2 \geq 1\}$ and the transition $\tau : \left| \begin{array}{r} 2x_1 + x_2 + \xi \leq 6 \\ x_2 - \xi \leq 2 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$

The relaxed guard is $\overline{G} = (2x_1 + x_2 \leq 6 \wedge x_1 + x_2 \leq 4 \wedge x_2 \leq 3) \wedge (0 \leq \xi \leq 1)$. Eliminating the inputs yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \wedge d_2 = 1\}$. We obtain $\tau^{\otimes}(X) = \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge x_2 - x_1 \leq 1 \wedge -4 \leq x_1 - 2x_2 \leq 1 \wedge x_1 + 2x_2 \leq 10 \wedge 2x_1 + x_2 \leq 10\}$, see Fig. 8. The convex hull of the exact result is $\{(x_1, x_2) | x_1 + x_2 \geq 1 \wedge -2 \leq x_2 - x_1 \leq 1 \wedge x_1 - 2x_2 \leq 1 \wedge x_2 \leq 3 \wedge 2x_1 + x_2 \leq 10\}$, see Fig. 9.

5. Backward Abstract Acceleration

Abstract acceleration has been applied to forward reachability analysis in order to compute the reachable states starting from a set of initial states. Backward analysis computes the states co-reachable from the *error* states. For example, combining forward and backward analysis allows to obtain an approximation of the sets of states belonging to a path from initial to error states (see for instance Jeannet (2003)). Moreover, a backward analysis allows us to *synthesize* constraints on *parameter* variables that ensure that a property is satisfied (see *e.g.* Alur et al. (1995)).

In this section, we present how to compute the accelerated backward transitions in the case of translations and translations/resets. Although the inverse of a translation is a translation, the difference is that the intersection with the guard occurs after the (inverted) translation. The case of backward translations with resets is more complicated than for the forward case, because resets are not invertible. Finally, the relaxation of general guards to simple guards applies in the same way to backward acceleration.

Proposition 5 (Polyhedral backward translation)**.** Let τ be a polyhedral translation $G \to \boldsymbol{x'} = \boldsymbol{x} + D$. Then the set

$$\tau^{-\otimes}(X') = X' \sqcup \big((\tau^{-1}(X') \nearrow (-D)\big) \sqcap G$$

is a convex over-approximation of $\tau^{-*}(X')$, where $\tau^{-*} = (\tau^{-1})^* = (\tau^*)^{-1}$ is the reflexive and transitive backward closure of τ.

The negation[2] of a polyhedron has the following meaning: $\boldsymbol{d} \in (-D) \iff (-\boldsymbol{d}) \in D$. It is hence the reflexion of D w.r.t. the origin.

Proof.

$\boldsymbol{x}_0 \in \bigcup_{k\geq 1} \tau^{-k}(X') \iff \exists \boldsymbol{x'} \in X' : \boldsymbol{x'} \in \tau\left(\bigcup_{k\geq 0} \tau^k(\{\boldsymbol{x}_0\})\right)$

$$\iff \exists k \geq 0, \exists \boldsymbol{x'} \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in D \ : \begin{cases} \boldsymbol{x}_k = \boldsymbol{x}_0 + \sum_{j=1}^{k} \boldsymbol{d}_j \\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_0 + \sum_{j=1}^{k'} \boldsymbol{d}_j) \\ \boldsymbol{x'} \in \tau(\{\boldsymbol{x}_k\}) \end{cases}$$

(forward reachability)

$$\iff \exists k \geq 0, \exists \boldsymbol{x'} \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in D \ : \begin{cases} \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x'}\}) \\ \boldsymbol{x}_0 = \boldsymbol{x}_k - \sum_{j=1}^{k} \boldsymbol{d}_j \\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_k - \sum_{j=k'+1}^{k} \boldsymbol{d}_j) \end{cases}$$

(rewritten as backward reachability)

$$\iff \exists k \geq 0, \exists \boldsymbol{x'} \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x'}\}) \ \wedge \ \boldsymbol{x}_0 = \boldsymbol{x}_k - k\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0) \ \wedge \ G(\boldsymbol{x}_k)$$

(because D and G are convex)

$$\implies \exists \alpha \geq 0, \exists \boldsymbol{x'} \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x'}\}) \ \wedge \ \boldsymbol{x}_0 = \boldsymbol{x}_k - \alpha\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0)$$

(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x'}\})$)

$\iff \boldsymbol{x}_0 \in \big((\tau^{-1}(X') \nearrow (-D)\big) \sqcap G.$ □

Example 6 (Polyhedral backward translation)**.** Consider the polyhedron $X' = \{(x_1, x_2) \mid 3 \leq x_1 \leq 6 \ \wedge \ 4 \leq x_2 \leq 5\}$ and the transition

$$\tau : \left| \begin{array}{c} x_1 + 2x_2 \leq 10 \ \wedge 0 \leq x_1 \leq 4 \ \wedge \\ 0 \leq x_2 \ \wedge 1 \leq \xi \leq 2 \end{array} \right. \to \left| \begin{array}{l} x'_1 = x_1 + 1 \\ x'_2 = x_2 + \xi \end{array} \right.$$

The polyhedron D is $\{(d_1, d_2) \,|\, d_1 = 1 \wedge 1 \leq d_2 \leq 2\}$. As result of the backward acceleration (Fig. 10) we obtain the polyhedron $\{(x_1, x_2) \mid 0 \leq x_1 \leq 6 \ \wedge \ 0 \leq x_2 \leq 5 \ \wedge \ -x_1 + x_2 \leq 2 \ \wedge \ 4x_1 - 3x_2 \leq 12\}$.

[2] Not to be confounded with the *complement*.

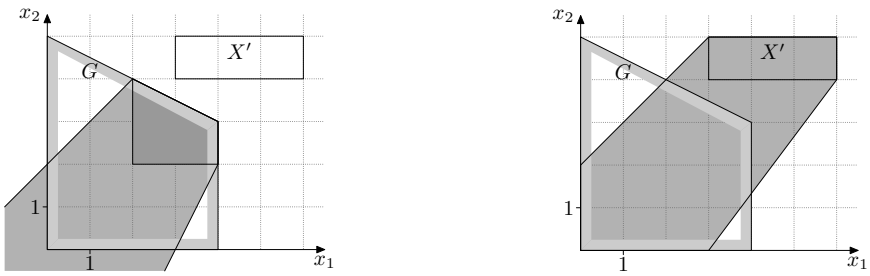


Fig. 10. Backward acceleration of a translation loop (Ex. 6) starting from X' with $\tau^{-1}(X')$ (dark gray) and $\tau^{-1}(X') \nearrow (-D)$ (whole shadowed area) on the left-hand side and the final result (right-hand side).

5.2. Translations with Resets

Proposition 6 (Polyhedral backward translation with resets)**.** Let τ be a polyhedral translation with resets $G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$. Then, the set

$$\tau^{-\otimes}(X') = X' \sqcup \tau^{-1}(X') \sqcup \tau^{-1}\Big(\big((\tau^{-1}(X')^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap D^{\bullet,r} \sqcap G\Big)$$

is a convex over-approximation of $\tau^{-*}(X')$.

Proof. The formula is trivially correct for 0 or 1 backward iterations of the self-loop τ, thus, it remains to show that, for the case of $k \geq 2$ iterations, our formula yields an over-approximation of $\bigcup_{k\geq 2} \tau^{-k}(X)$.

$\boldsymbol{x} \in \bigcup_{k\geq 2} \tau^{-k}(X') \iff \exists \boldsymbol{x'} \in X' \ : \ \boldsymbol{x}' \in \tau\left(\bigcup_{k\geq 0} \tau^{k}(\tau(\boldsymbol{x}))\right)$

$\iff \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$

$$\begin{cases} \quad \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \wedge \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \\ \wedge\ \forall k' \in [1,k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_0^{t,0} + \sum_{j=1}^{k'} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \end{cases}$$

(forward reachability)

$\iff \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$

$$\begin{cases} \quad \forall k' \in [0,k-1] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_k^{t,0} - \sum_{j=k'+1}^{k} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge\ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

(rewritten as backward reachability)

$$\Longrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \ldots \boldsymbol{x}_k, \exists \boldsymbol{d}_1^{t,0} \ldots \boldsymbol{d}_k^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_1^{0,r} \ldots \boldsymbol{d}_k^{0,r} \in D^{0,r} \ :$$

$$\begin{cases} \quad \forall k' \in [0, k-1] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_k^{t,0} - \sum_{j=k'+1}^{k} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge \ \forall k' \in [0, k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

(D approximated by the sum $(D^{t,0} + D^{0,r})$)

$$\Longleftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}^{0,r} \in D^{0,r} \ :$$

$$\begin{cases} \quad \boldsymbol{x}_0 = \boldsymbol{x}_k^{t,0} - k\boldsymbol{d}^{t,0} + \boldsymbol{d}^{0,r} \\ \wedge \ G(\boldsymbol{x}_0) \wedge G(\boldsymbol{x}_k) \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

(because $D^{t,0}, D^{0,r}$ and G are convex and $\boldsymbol{x}_k^{0,r} \in D^{0,r}$)

$$\Longrightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}^{0,r} \in D^{0,r} \ :$$

$$\begin{cases} \quad \boldsymbol{x}_0 = \boldsymbol{x}_k^{t,0} - \alpha\boldsymbol{d}^{t,0} + \boldsymbol{d}^{0,r} \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \ \wedge \ G(\boldsymbol{x}_0) \end{cases}$$

(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\})$)

$$\Longleftrightarrow \boldsymbol{x} \in \tau^{-1}\Big(\big((\tau^{-1}(X'))^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap D^{\bullet,r} \sqcap G\Big) \quad \square$$

(because $\boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}'\}) \Rightarrow \boldsymbol{x}' \in D^{\bullet,r}$)

Example 7 (Polyhedral backward translation with resets)**.** Consider the polyhedron $X' = \{(x_1, x_2) \mid 4 \leq x_1 \leq 5 \ \wedge \ 1 \leq x_2 \leq 4\}$ and the transition

$$\tau : \left| \begin{array}{l} 3 \leq x_1 + x_2 \leq 5 \ \wedge \\ \qquad 1 \leq \xi \leq 3 \ \wedge 0 \leq x_2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 1 \\ x_2' = \xi \end{array} \right.$$

The polyhedron D is $\{(d_1, d_2) | d_1 = 1 \wedge 1 \leq d_2 \leq 3\}$. As result of the backward acceleration (Fig. 11), we obtain the polyhedron $\{(x_1, x_2) | -1 \leq x_1 \leq 5 \wedge 0 \leq x_2 \wedge x_1 + x_2 \geq 3 \wedge x_1 - x_2 \leq 4 \ \wedge \ x_1 + 3x_2 \leq 17\}$.

6. Evaluation and Comparison

This section discusses the advantages and shortcomings of abstract acceleration in comparison with more general abstract-interpretation-based methods like standard widening (Cousot and Cousot (1977)) and the affine derivative closure method of Ancourt et al. (2010).

6.1. Comparing Abstract Acceleration with Kleene Iteration

Abstract acceleration aims at computing a tight over-approximation of $\alpha(\bigcup_{k \geq 0} \tau^k(X_0))$ where X_0 is a convex polyhedron and τ is an affine transformation with an affine guard. Since convex polyhedra are closed under affine transformations ($\alpha(\tau(X_0)) = \tau(X_0)$), we

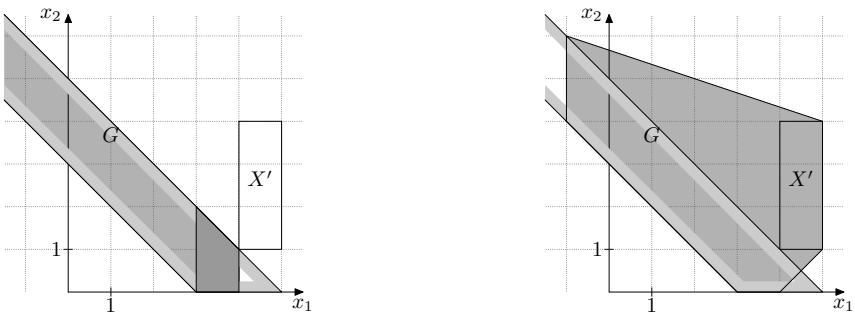


Fig. 11. Backward acceleration of a loop with translations and resets (Ex. 7) starting from the initial set X'. Right-hand side: $\tau^{-1}(X')$ (dark gray) and $\big((\tau^{-1}(X')|_t \nearrow (-D^t)\big) \sqcap G$ (whole shadowed area). Left-hand side: final result (whole shadowed area).

have $\alpha(\bigcup_{k\geq 0} \tau^k(X_0)) = \bigsqcup_{k\geq 0} \tau^k(X_0)$. The latter formula is known as *Merge-Over-All-Paths* (MOP) solution of the reachability problem (Kam and Ullman (1977)), which computes the limit of the sequence:

$$X_0 \qquad X_1 = X_0 \sqcup \tau(X_0) \qquad X_2 = X_0 \sqcup \tau(X_0) \sqcup \tau^2(X_0) \qquad \ldots$$

In contrast, the standard approach in abstract interpretation computes the fixed point X'_∞ of $X = X_0 \sqcup \tau(X)$, known as the *Minimal-Fixed-Point* (MFP) solution. It proceeds as follows:

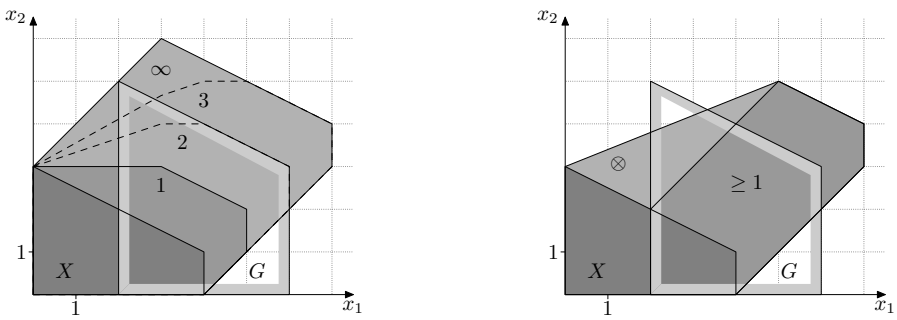
$$X'_0 = X_0 \qquad X'_1 = X'_0 \sqcup \tau(X'_0) \qquad X'_2 = X'_0 \sqcup \tau(X'_0 \sqcup \tau(X'_0)) \qquad \ldots$$

The MOP solution is more precise than the MFP solution (Kam and Ullman (1977)). The reason is that, in general, τ does not distribute over $\sqcup$ and we have $\tau(X_1) \sqcup \tau(X_2) \sqsubseteq \tau(X_1 \sqcup X_2)$. For instance, if $X_0 = [0,0]$ and $\tau : x \leq 1 \rightarrow x' = x+2$, we have $X_2 = [0,0] \sqcup [2,2] \sqcup \bot = [0,2]$ and $X'_2 = [0,0] \sqcup \tau([0,2]) = [0,3]$. Since abstract acceleration should deliver a tight over-approximation of the MOP solution, we should generally have the relationship $\bigsqcup_{k\geq 0} \tau^k(X_0) \sqsubseteq \tau^\otimes(X_0) \sqsubseteq X'_\infty$, *i.e.* abstract acceleration should be more precise than the standard abstract interpretation approach (MFP) even without widening.

Fig. 12 shows an example illustrating this issue: in the standard abstract interpretation approach (MFP), each iteration translates the approximation added by the convex union with the result of the previous iteration in each step and converges slowly. Abstract acceleration translates only the intersection with the guard and takes the union as a last step. Observe, however, that in the case where X_0 is contained in the guard G, MOP and MFP solutions give identical results for this example (Leroux and Sutre (2007)).

Non-flat systems. Generally, the invariants computed by abstract acceleration of a single self-loop τ are not inductive, which implies that $\tau^\otimes$ is not idempotent, *i.e.* $\tau^\otimes(\tau^\otimes(X_0)) \neq \tau^\otimes(X_0)$. While this is not a problem for flat systems, it has negative effects in the presence of nested loops.

For example, in the system $(id \circ \tau^*)^*$ we can apply abstract acceleration to the innermost loop: $(id \circ \tau^\otimes)^* = (\tau^\otimes)^*$. If $\tau^\otimes$ is not idempotent (like in Fig. 12b), then the outer loop might not converge and thus widening is needed. Thus, the considerations w.r.t. MOP and MFP solutions for τ^* above apply to $(\tau^\otimes)^*$ in the same manner.



(a) MFP: X dark gray, iteration 1 medium gray, iterations 2 and 3 dashed, final result whole shadowed area.

(b) Abstract acceleration: X dark gray, iterations ≥ 1 medium gray, final result whole shadowed area.

Fig. 12. Comparison between standard abstract interpretation (MFP) (a) and abstract acceleration (b): $\tau : G \to (x_1', x_2') = (x_1+1, x_2+1)$ with G and X as given in the figures.

Since this problem arises in particular when the initial set is not contained in the guard G (like in Fig. 12b), Leroux and Sutre (2007) propose to accelerate translations by the formula $\tau^{\otimes}(X) = \tau(X \nearrow D)$, *i.e.* without initially intersecting with G, which is idempotent and hence convergence without widening can be expected more often. However, this formula is clearly less precise and should not be used for accelerating non-nested loops.

6.2. Comparing Abstract Acceleration to Widening

In this section, we illustrate by some examples where abstract acceleration helps increasing the predictability and the robustness of the analysis.

Widening in self-loops. We consider Ex. 5, which has the structure depicted in Fig. 13 when analyzed with abstract acceleration or Kleene iteration with widening and descending iterations respectively.

The standard widening operator for convex polyhedra and refinements of it, like *widening with thresholds* (Halbwachs et al. (1997)), may sometimes lead to good results. Analyzing such a program using widening after a number N of initial steps resorts to computing the limit of the sequences

$$X_0 = X \qquad\qquad X_0' = X_N$$
$$X_{n+1} = X_0 \sqcup \tau(X_n) \quad \text{for } n < N \qquad\qquad X_{n+1}' = X_n' \nabla (X_n' \sqcup \tau(X_n'))$$

in which X_n, X_n' are associated with location l_1 on Fig. 13 (right-hand side). The properties of the widening operator ∇ guarantee that the sequence $(X_n')_{n\geq 0}$ converges in a finite number of steps to X_∞' (Cousot and Cousot (1992)), which is an over-approximation of the reachable valuations at location l_1. This result may be improved by computing the first elements of the *narrowing* sequence $X_0'' = X_\infty', X_{n+1}'' = X_0 \sqcup \tau(X_n'')$, which does not necessarily converge.

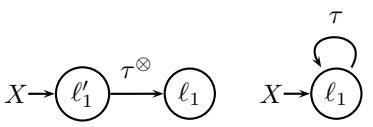


Fig. 13. Analysis with acceleration (left-hand side) and with widening (right-hand side) for Ex. 5.

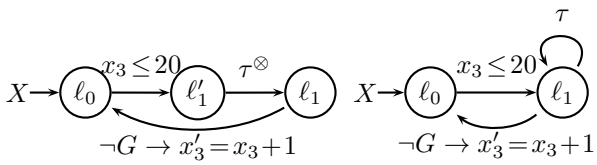


Fig. 14. Analysis with acceleration (left-hand side) and with widening (right-hand side) for Ex. 8. $G = (2x_1 + 2x_2 \leq x_3)$.

We try first an analysis with undelayed widening, *i.e.* $N = 0$:

$$
\begin{aligned}
X'_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1\} && \text{(result of converged widening sequence)}\\
X''_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge x_2 \leq 4 \wedge 0 \leq x_1 \leq 6 \wedge 3x1 + 5x_2 \geq 3\}\\
&\ldots\\
X''_3 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge 3x_2 - 2x_1 \leq 6 \wedge 3x_2 - 4x_1 \leq 3 \wedge\\
&\qquad 5x_1 - 22x_2 \leq 8 \wedge 29x_1 - 157x_2 \leq 29\} \sqsupset \tau^{\otimes}(X)\\
&\ldots && \text{(descending sequence)}
\end{aligned}
$$

Here, the descending iterations do not converge and improve the result slowly. See Fig. 9 for X''_3.

By increasing the delay to $N = 1$, we can improve the result, and we get the same result as with abstract acceleration: $X''_1 = \tau^{\otimes}(X)$. However, this is not guaranteed in general, because widening is not monotonic. Moreover, abstract acceleration is more efficient computationally: delaying widening and a long descending sequence increase the number of iterations. In flat programs, like in Ex. 5, abstract acceleration does not even require convergence tests.

Widening in nested loops. In the previous example, delayed widening and descending iterations allowed to get the same result as with abstract acceleration. However, this is less likely if the loop is embedded in an outer loop as in Fig. 14: descending iterations cannot be applied during the ascending iterations, but only after convergence of the widening sequence of the whole program, otherwise convergence would not be guaranteed.

Example 8 (Widening in nested loops)**.** We consider the program depicted in Fig. 14 in which the inner loop τ is adapted from Ex. 4:

$$
\tau : \left| \begin{array}{c} 2x_1 + 2x_2 \leq x_3 \\ 0 \leq \xi \leq 1 \end{array} \right. \to \left| \begin{array}{l} x'_1 = x_1 + \xi + 1 \\ x'_2 = \xi \\ x'_3 = x_3 \end{array} \right. \qquad X = \left\{ (x_1, x_2, x_3) \left| \begin{array}{l} 0 \leq x_1 \ \wedge \ 1 \leq x_2 \\ x_1 + x_2 \leq 2 \wedge x_3 = 3 \end{array} \right. \right\}
$$

The analysis without abstract acceleration yields for any widening delay $N \geq 1$ (widening point ℓ_1) and any number of descending iterations $N'' \geq 1$ the following very weak invariant:

$$
X'_1 = \{(x_1, x_2, x_3) \mid 0 \leq x_1 \wedge 3 \leq x_3\} \quad X''_1 = \{(x_1, x_2, x_3) \mid 0 \leq x_1 \wedge 1 \leq x_1 + x_2 \wedge 3 \leq x_3\}
$$

Abstract acceleration with widening delay $N \geq 1$ (widening point ℓ_1) and one descending iteration gives much better results: we give here a simplified (over-approximated)

invariant, because the actual result consists of more constraints:

$$X_1'' = \{(x_1, x_2, x_3) \mid 0 \leq x_1 \leq 12 \wedge 0 \leq x_2 \leq 3 \wedge 3 \leq x_3 \leq 20 \wedge 1 \leq x_1 + x_2\}$$

One can also consider widening with thresholds. A natural threshold set for our example is the postcondition of the guard of τ by the body of τ: $\tau(\top) = \{(x_1, x_2) \mid 0 \leq x_2 \leq 1\}$. Yet, this does improve the result.

Extending the threshold set with the postcondition of the guard of the outer loop $x_3 \leq 21$ improves the result (all variables are bounded), but it is still less precise than the result obtained by combining abstract acceleration and widening (in particular the descending iteration does not converge).

6.3. Comparing Abstract Acceleration to the Affine Derivative Closure Algorithm

The affine derivative closure algorithm of Ancourt et al. (2010) is another abstract interpretation-based analysis method. The idea is to compute an abstract transformer, *i.e.* a relation between variables $\boldsymbol{x}$ and $\boldsymbol{x}'$, independently of the initial state of the system. The abstract transformer *abstracts* the effect of the loop by a polyhedral translation

$$\mathit{true} \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D_R \quad \text{with} \quad D_R = \{\boldsymbol{d} \mid \exists \boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{x}' : R(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{x}') \wedge \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}\}$$

where R is the concrete transition relation. The polyhedron D_R is called the "derivative" of the relation R. The effect of several self-loops with relations $R_1, \ldots, R_k$ is abstracted by considering the convex union $\bigsqcup_i D_{R_i}$.

Then, the reflexive and transitive closure

$$R^* = \{(\boldsymbol{x}, \boldsymbol{x}') \mid \exists k \geq 0 : \boldsymbol{x}' = \boldsymbol{x} + k\boldsymbol{d} \wedge D_R(\boldsymbol{d})\}$$

is applied to a polyhedron X of initial states:

$$R^*(X) = \{\boldsymbol{x}' \mid \exists \boldsymbol{x} : R^*(\boldsymbol{x}, \boldsymbol{x}') \wedge X(\boldsymbol{x})\}$$

The final result is obtained by computing one "descending" iteration, in the same way as it is done in standard abstract interpretation after widening.

The affine derivative closure algorithm is implemented in the code optimization tool PIPS [3].

In single self-loops with translations or translations/resets, the method works similarly to abstract acceleration, as illustrated by the following example involving resets and inputs:

Example 9 (Single loop)**.**

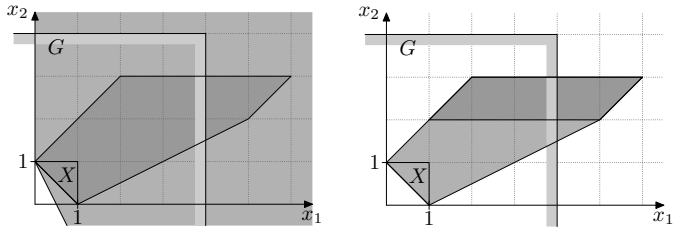
$$\tau : \left| \begin{array}{l} x_1 \leq 4 \\ x_2 \leq 4 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi + 2 \end{array} \right. \qquad X = \left\{ (x_1, x_2) \left| \begin{array}{l} x_1 \leq 1 \ \wedge \ x_2 \leq 1 \\ x_1 + x_2 \geq 1 \end{array} \right. \right\}$$

The transition relation

$$R = \{(x_1, x_2, \xi, x_1', x_2') \mid x_1' = x_1 + \xi + 1 \wedge x_2' = \xi + 2 \wedge x_1 \leq 4 \wedge x_2 \leq 4 \wedge 0 \leq \xi \leq 1\}$$

expressed in terms of derivatives is

[3] `http://pips4u.org`



(a) Derivative closure method: R^* applied to X (whole shadowed area), final result (dark gray).

(b) Abstract acceleration: iterations ≥ 1 (dark gray), final result (whole shadowed area).

Fig. 15. Comparison between the affine derivative closure algorithm (a) and abstract acceleration (b), Ex. 9.

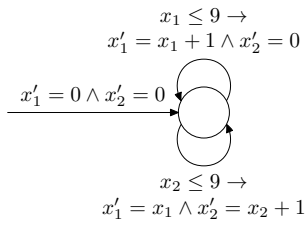


Fig. 16. Ex. 10.

$$D_R = \{(d_1, d_2) \mid \exists x_1, x_2, \xi, x_1', x_2' : R(x_1, x_2, \xi, x_1', x_2') \wedge x_1' = x_1 + d_1 \wedge x_2' = x_2 + d_2\}$$
$$= \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \wedge d_1 - d_2 \leq 3\}$$

The closure of the loop starting from X gives

$$R^*(X) = \{(x_1', x_2') \mid \exists k \geq 0, x_1, x_2 : x_1' \geq x_1 + k \wedge x_1' \leq x_1 + 2k \wedge$$
$$x_1' - x_2' \leq x_1 - x_2 + 3k \wedge X(x_1, x_2)\} =$$
$$= \{(x_1', x_2') \mid 0 \leq x_1' \wedge 2x_1' + x_2' \geq 1\}$$

Finally, a descending iteration is computed:

$$X \sqcup \tau(R^*(X)) = \{(x_1', x_2') \mid x_1' + x_2' \geq 1 \wedge x_1' - 2x_2' \leq 1 \wedge x_1' - x_2' \leq 3 \wedge$$
$$x_2' \leq 3 \wedge x_1' - x_2' \geq -1\}$$

This result equals the one obtained by abstract acceleration (see Fig. 15).

Resets cannot be expressed as polyhedral translations: for instance, if $R(x, x') = (x' = 0)$, then $D_R = \{d \mid \exists x, x' : x' = 0 \wedge x' = x + d\} = \top$. However, this information is recovered during the descending iteration. Hence, similarly to widening, these descending iterations may fail in the presence of multiple loops:

Example 10 (Multiple loops)**.** For the CFG in Fig. 16, the derivatives for the upper and the lower loop are $D_{R_1} = (d_1 = 1)$ and $D_{R_2} = (d_1 = 0 \wedge d_2 = 1)$ respectively. Their convex union is $D_{R_{1,2}} = (0 \leq d_1 \leq 1)$. The transitive closure applied to $X = (x_1 = x_2 = 0)$ gives $R^*_{1,2}(X) = (x_1 \geq 0)$, and the final result after the descending iteration is $X' = (x_1 \geq 0 \wedge x_2 \leq 10)$. Here, the descending iteration fails to recover the upper bound of x_1.

In contrast, abstract acceleration converges after two ascending iterations with the invariant $X_2 = (0 \leq x_1 \leq 10 \wedge 0 \leq x_2 \leq 10)$.

Hence, even though the derivative closure method elegantly deals with multiple loops by taking the convex union of the derivatives, it is also less precise than abstract acceleration for such programs. However, the main advantage of the derivative closure method is that it is more general than abstract acceleration, because it automatically approximates any kind of transformations. Moreover, since it computes abstract transformers, it is modular and can be used in the context of interprocedural analyses.

	size	nbACCel	Nbac
Gate 1	7	**0.73**	?
Escalator 1	12	**0.49**	?
Traffic 1	18	**0.19**	3.49
Traffic 2	18	**0.35**	?
LCM Quest 0a-1	7	**0.04**	0.05
LCM Quest 0a-2	6	**0.05**	0.19
LCM Quest 0b-1	19	**0.08**	?
LCM Quest 0b-2	17	**0.20**	?
LCM Quest 0c-1	28	**0.16**	0.86

	size	nbACCel	Nbac
LCM Quest 0c-2	25	**0.24**	14.8
LCM Quest 1-1	114	**0.92**	2.45
LCM Quest 2-1	247	**7.84**	12.8
LCM Quest 3-1	483	8.49	**3.76**
LCM Quest 3b-1	1724	43.8	**19.1**
LCM Quest 3c-1	1319	**34.2**	?
LCM Quest 3d-1	281	**5.43**	?
LCM Quest 3e-1	638	**20.6**	?
LCM Quest 4-1	4482	186	**50.1**

Table 1. Experimental comparison between nbACCel and Nbac (size. . . number of locations of the enumerated CFG; times in seconds; ?. . . property not proved)

7. Applying Abstract Acceleration

Control flow graphs. Program analysis using abstract interpretation in general and abstract acceleration in particular, is done on a control flow graph (CFG) of the program, which manipulates only numerical variables. This graph can be easily obtained from imperative programs by associating control points to programming constructs.

As mentioned in the introduction our goal is to analyze logico-numerical data-flow programs such as Lustre (Caspi et al. (1987)) programs: In order to reduce such a program to a purely numerical CFG, all possible valuations of Boolean state variables need to be *enumerated* and encoded in locations of the CFG. This *partitioning* and *partial evaluation* process may lead to a combinatorial explosion of control locations.

Jeannet (2003) presented abstract interpretation methods for analyzing such programs using *controlled partitioning*, *i.e.* some Boolean states are encoded explicitly as locations in the CFG, whereas the other Boolean variables are handled symbolically. Pursuing this approach, we have developed methods for applying abstract acceleration to such a *logico-numerical* CFG (see Schrammel and Jeannet (2011) for details).

Pre-processing. Some transformations are necessary in order to prepare a self-loop in a CFG for abstract acceleration. These comprise especially the elimination of Boolean input variables resulting in a non-deterministic CFG and splitting non-convex guards into convex ones. Methods for dealing with multiple self-loops in a single location are described by Gonnord (2007).

Experimental results. Table 1 shows some experimental results comparing our tool nbACCel, which implements the methods presented in this article, with the tool Nbac (Jeannet (2003)). The benchmarks have quite different sizes from ten up to a few hundred lines of Lustre code. The safety properties we want to prove about the small examples require a very good precision on the numerical variables. We observe that we are able to prove some benchmarks that are not provable using Nbac. This is due to the precision gained by abstract acceleration. But also on the other examples nbACCel is mostly faster than Nbac; except for some of the larger benchmarks where the more sophisticated dynamic partitioning techniques employed in Nbac start to pay off. Further results including a comparison with Aspic (Gonnord (2009)) can be found in Schrammel and Jeannet (2011).

8. Related Work

As already outlined in the introduction, there are essentially two approaches to the reachability problem: Those that aim at computing exact results using additive integer (Presburger) or real arithmetic without guarantee of termination (in the cases mentioned in the introduction); and those that are based on abstract interpretation which terminate but generally ignore divisibility properties and deliver only conservative approximations.

Presburger-based approaches. Bultan et al. (1997) use Presburger formulas to represent the state space of integer variables in symbolic model checking of concurrent programs, but they use an abstract-interpretation-like widening operator in order to accelerate loops and actually compute an over-approximation.

Boigelot and Wolper (1994); Boigelot (1999) introduce a representation called *periodic vector sets* of the form $\exists \boldsymbol{k} \in \mathbb{Z}^m : \boldsymbol{x} = \mathbf{C}\boldsymbol{k}+\boldsymbol{d} \wedge \mathbf{P}\boldsymbol{k} \leq \boldsymbol{q}$ for capturing the transitive closure of affine transformations, and apply it to communication protocols involving integer variables. Their method tries to accelerate selected cycles in so-called *meta-transitions*, but without guarantee for termination.

Fribourg and Olsén (1997) compute the reachable state space of Petri nets with unbounded markings using a decision procedure over Presburger arithmetic.

Finkel and Leroux (2002); Bardin et al. (2004) apply Presburger-based acceleration to communication protocols. They extend the results of Boigelot (1999) to periodic affine transformations as defined in Sec. 3. Their methods have also been applied to FIFO queues using subclasses of regular expressions (Boigelot and Godefroid (1997); Abdulla et al. (2004)). Bardin et al. (2003) describes the tool FAST for analyzing *flattable* programs, *i.e.* without nested loops. Bardin et al. (2005) present an overall framework for various classes of systems that can be treated by exact acceleration.

Bozga et al. (2010) unifies the computation of transitive closures of such as difference relations, octagonal relations and finite monoid affine transformations by defining the notion of *ultimately periodic relations*. Their method is also based on Presburger arithmetic. Like us, this work investigates the case of relations instead of deterministic functions. Difference relations (and their octogonal generalization) have an incomparable expressive power compared to translations/resets with inputs and simple guards seen as relations:

- Difference relations can express for instance variables permutation, unlike translations/resets;
- but they cannot express translations/resets involving guards with general linear constraints, such as $(x_1+2x_2 \leq 10) \rightarrow \left| \begin{array}{l} x_1' = x_1+1 \\ x_2' = x_2+2 \end{array} \right.$ or $(\xi_1+2\xi_2=3) \rightarrow \left| \begin{array}{l} x_1' = x_1+\xi_1 \\ x_2' = x_2+\xi_2 \end{array} \right.$

Last, finite monoid affine transformations, which are deterministic, correspond to the periodic case that we did not consider in Sec. 3.

Acceleration has also been used in the analysis of *timed and hybrid automata*. For example, the concepts of Boigelot (1999) were developed further by Boigelot et al. (2003) to the acceleration of general affine relations $R(\boldsymbol{x}, \boldsymbol{x}')$. The basic technique applies to relations of the form $G(\boldsymbol{x}) \wedge G'(\boldsymbol{x}') \wedge P(\boldsymbol{x}' - \boldsymbol{x})$ where G, G' and P are conjunctions of linear inequalities. Observe that, if G' is true, such relations can be expressed in our setting as pure translations with inputs and simple guards, defined by $G(\boldsymbol{x}) \wedge P(\boldsymbol{\xi}) \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\xi}$. Boigelot and Herbreteau (2006) develop transformation methods (based

mainly on variable changes) to reduce subclasses of affine relations to this basic case. It would be interesting to apply and adapt their methods to abstract acceleration. These two papers apply this approach to the verification of rectangular hybrid systems (as defined by Alur et al. (1995)) that include timed automata.

Recent research exhibited results on the transitive closure of *affine integer tuple relations* (Beletska et al. (2009)). Their goal is to compute multiple clause relations, *i.e.* several self-loops, with commutative transition relations: $\tau_1 \circ \tau_2 = \tau_2 \circ \tau_1$, *e.g.* $\tau_1 : x_1' = x_1 + 1; x_2' = 2x_2$ and $\tau_2 : x_1' = x_1 + 3; x_2' = 5x_2$. They use Presburger-based methods and apply their results in the context of program parallelization.

Abstract interpretation-based approaches to acceleration are comparatively recent. Also our method falls into this category and extends the work of Gonnord and Halbwachs (2006) and Gonnord (2007). Apart from self-loops of translations and translations with resets for which we developed our extensions, they also deal with periodic affine transformations and some special cases of multiple self-loops. Furthermore, they describe methods for unfolding multiple self-loops in order to compute the fixed point more efficiently.

The affine derivative closure algorithm (Ancourt et al. (2010)) that we explained in more detail in Section 6.3 is also based on abstract interpretation methods, but unlike abstract acceleration it neither inspects the types of transitions in order to apply an optimized acceleration formula nor specializes w.r.t. a given initial state.

9. Conclusion and Discussion

We have presented an extension of abstract acceleration to numerical inputs for forward and backward analysis. Table 2 shows a summary of the formulas. This extension is less straightforward than supposed – most notably due to the observation that inputs can be used to turn translations into arbitrary affine transformations; also, resetting variables to input values may cause some subtle behavior. Regarding approximations, Table 2 shows the cases where our method is precise in the sense that we perform only dense and convex approximations, and the more complex cases for which additional approximations are necessary to abstract away the number of iterations.

Abstract acceleration elegantly integrates into an abstract interpretation-based verification tool, where it is usually used in combination with widening: as pointed out in Sec. 6.2, due to its monotonicity property it is possible to accelerate the innermost loops precisely while using widening for the outer loops in nested loop situations. Thus, much better invariants can be computed for programs where a lot of information is lost when using widening only.

In comparison to other abstract interpretation-based transitive closure methods, for instance the affine derivative closure algorithm of Ancourt et al. (2010), abstract acceleration deals only with some frequently occurring types of self-loop transitions, for which it yields more precise results (especially for transitions with resets), but it needs to resort to widening in the general case.

We have reported some first experimental results that give evidence about the potential of abstract acceleration w.r.t. improving reachability analysis in terms of precision and performance.

Regarding future work, we could extend slightly the transformations we consider. Firstly, we conjecture that it is possible to generalize our results from the case where C

Forward acceleration:

Translations	$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$
Translations/resets	$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big)$

Backward acceleration:

Translations	$\tau^{-\otimes}(X') = X' \sqcup \big((\tau^{-1}(X') \nearrow (-D)\big) \sqcap G$
Translations/resets	$\tau^{-\otimes}(X') = X' \sqcup \tau^{-1}(X') \sqcup$ $\sqcup \tau^{-1}\Big(\big((\tau^{-1}(X')^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap D^{\bullet,r} \sqcap G\Big)$

with

$$\tau \;:\; (\overbrace{\mathbf{A}\boldsymbol{x} + \mathbf{L}\boldsymbol{\xi} \leq \boldsymbol{b}}^{G_0(\boldsymbol{x},\boldsymbol{\xi})} \wedge \overbrace{\mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}}^{G_1(\boldsymbol{\xi})}) \longrightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u} \qquad \mathbf{C} \text{ diagonal with 0 or 1 only}$$

$$G(\boldsymbol{x}) = \exists \boldsymbol{\xi} : G_0(\boldsymbol{x}, \boldsymbol{\xi}) \qquad D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge G_1(\boldsymbol{\xi}) \wedge \exists \boldsymbol{x} : G_0(\boldsymbol{x}, \boldsymbol{\xi})\}$$

Approximations:

In all cases	dense and convex approximation
$L \neq \mathbf{0}$	G and D are decoupled.
Translations/resets	D is approximated by the Cartesian product $D^t \times D^r$.

Table 2. Overview of abstract acceleration formulas

is a diagonal matrix with either 0 or 1 to the case where C is periodic with prefix p and period l as defined in Sec. 3. This could be done in the same way as Finkel and Leroux (2002) by rewriting $\bigcup_{k \geq 0} \tau^k$ as

$$\tau^0 \cup \ldots \cup \tau^{p-1} \cup \bigcup_{k \geq 0} (\tau^l)^k \circ \tau^p \cup \ldots \cup \bigcup_{k \geq 0} (\tau^l)^k \circ \tau^{p+l-1}$$

and accelerating τ^l on the image of τ^{p+i} with $0 \leq i \leq l-1$.

Secondly, it should be also possible to generalize and unify the two classes of relations induced resp. by forward and backward translations/resets with inputs and simple guards to a more general symmetrical class, and to consider the acceleration of such relations. A candidate could be

$$R(\boldsymbol{x}, \boldsymbol{x}') \Leftrightarrow \left\{ \begin{array}{c} \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq k \\ \mathbf{A}'\boldsymbol{x}' \leq \boldsymbol{b}' \end{array} \right. \wedge \left\{ \begin{array}{rcll} \boldsymbol{x}'_t - \boldsymbol{x}_t & = & \mathbf{T}_t\boldsymbol{\xi} + \boldsymbol{u}_t & \text{(translated dimensions)} \\ \boldsymbol{x}'_f & = & \mathbf{T}_f\boldsymbol{\xi} + \boldsymbol{u}_f & \text{(forward reset dimensions)} \\ \boldsymbol{x}_b & = & \mathbf{T}_b\boldsymbol{\xi} + \boldsymbol{u}_b & \text{(backward reset dimensions)} \end{array} \right.$$

in which the set of dimensions is partioned into three classes.

Regarding integer (*e.g.,* divisibility) properties, our techniques based on convex polyhedra cannot express them and Remark 2 discusses the effect of the induced dense approximation. To improve on this we could combine our techniques with the linear congruence abstract domain introduced in Granger (1991). This domain satisfies the finite ascending condition, hence it does not require widening nor acceleration. By this means we are able to tighten the results. For instance, if the abstract acceleration results in a convex polyhedron $2 \leq x_1 \leq 8 \wedge 2x_1 + x_2 \leq 9$, and we know from the linear congruences domain that

$x_1 = 1 \bmod 4 \wedge x_2 = 0 \bmod 2$ (typically because (x_1, x_2) are iteratively translated by $(4, 2)$ from a known value), then the convex polyhedron can be tightened to $x_1 = 5 \wedge 2x_1 + x_2 \leq 8$. Compared to Presburger arithmetic, we still limit ourselves to convex sets with such a technique. The general goal would be to add congruence constraints in the guard of the transformation τ and to exploit them as sketched above.

The question under which condition translations/resets with inputs viewed as relations are ultimately periodic relations in the sense of Bozga et al. (2010) could be also investigated. We conjecture that this is only true in the cases where our methods perform only dense and convex approximations (see Table 2).

References

Abdulla, P. A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B., 2004. Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design 25 (1), 39–65.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S., 1995. The algorithmic analysis of hybrid systems. Theoretical Computer Science 138 (1), 3–34.

Ancourt, C., Coelho, F., Irigoin, F., 2010. A modular static analysis approach to affine loop invariants detection. In: Numerical and Symbolic Abstract Domains. Vol. 267 of ENTCS. Elsevier, pp. 3–16.

Bagnara, R., Dobson, K., Hill, P. M., Mundell, M., Zaffanella, E., 2006. Grids: A domain for analyzing the distribution of numerical values. In: Logic-Based Program Synthesis and Transformation, LOPSTR'06. Vol. 4407 of LNCS. pp. 219–235.

Bardin, S., Finkel, A., Leroux, J., 2004. Faster acceleration of counter automata in practice. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 2988 of LNCS. pp. 576–590.

Bardin, S., Finkel, A., Leroux, J., Petrucci, L., 2003. FAST: Fast acceleration of symbolic transition systems. In: Computer Aided Verification. Vol. 2725 of LNCS. Springer, pp. 118–121.

Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P., 2005. Flat acceleration in symbolic model checking. In: Automated Technology for Verification and Analysis. Vol. 3707 of LNCS. Springer, pp. 474–488.

Beletska, A., Barthou, D., Bielecki, W., Cohen, A., 2009. Computing the transitive closure of a union of affine integer tuple relations. In: Combinatorial Optimization and Applications. Vol. 6508 of LNCS. Springer, pp. 98–109.

Benoy, F., King, A., Mesnard, F., 2005. Computing convex hulls with a linear solver. Theory and Practice of Logic Programming 5 (1-2), 259–271.

Boigelot, B., 1999. Symbolic methods for exploring infinite state spaces. Phd thesis, University of Liège.

Boigelot, B., Godefroid, P., 1997. Symbolic verification of communication protocols with infinite state spaces using QDDs. Formal Methods in System Design 14 (3), 237–255.

Boigelot, B., Herbreteau, F., 2006. The power of hybrid acceleration. In: Computer Aided Verification. Vol. 4144 of LNCS. Springer, pp. 438–451.

Boigelot, B., Herbreteau, F., Jodogne, S., 2003. Hybrid acceleration using real vector automata (extended abstract). In: Computer Aided Verification. Vol. 2725 of LNCS. pp. 193–205.

Boigelot, B., Wolper, P., 1994. Symbolic verification with periodic sets. In: Computer Aided Verification. Vol. 818 of LNCS. Springer, pp. 55–67.

Bozga, M., Iosif, R., Konecný, F., 2010. Fast acceleration of ultimately periodic relations. In: Computer Aided Verification. Vol. 6174 of LNCS. Springer, pp. 227–242.

Bultan, T., Gerber, R., Pugh, W., 1997. Symbolic model checking of infinite state systems using presburger arithmetic. In: Computer Aided Verification. Vol. 1254 of LNCS. Springer, pp. 400–411.

Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J. A., 1987. LUSTRE: a declarative language for real-time programming. In: Principles of Programming Languages. ACM, pp. 178–188.

Comon, H., Jurski, Y., 1998. Multiple counters automata, safety analysis and presburger arithmetic. In: Computer Aided Verification. Vol. 1427 of LNCS. Springer, pp. 268–279.

Cousot, P., Cousot, R., 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Principles of Programming Languages. pp. 238–252.

Cousot, P., Cousot, R., 1992. Abstract interpretation and application to logic programs. Journal of Logic Programming 13 (2–3), 103–179.

Cousot, P., Halbwachs, N., 1978. Automatic discovery of linear restraints among variables of a program. In: Principles of Programming Languages. ACM, pp. 84–97.

de Berg, M., Cheong, O., van Kreveld, M., Overmars, M., 2008. Computational Geometry: Algorithms and Applications, 3rd Edition. Springer.

Finkel, A., Leroux, J., 2002. How to compose Presburger-accelerations: Applications to broadcast protocols. In: Foundations of Software Technology and Theoretical Computer Science. Vol. 2556 of LNCS. Springer, pp. 145–156.

Fribourg, L., Olsén, H., 1997. Proving safety properties of infinite state systems by compilation into presburger arithmetic. In: Conference on Concurrency Theory. Vol. 1243 of LNCS. Springer, pp. 213–227.

Fukuda, K., Prodon, A., 1996. Double description method revisited. In: Combinatorics and Computer Science. Vol. 1120 of LNCS. Springer, pp. 91–111.

Gonnord, L., 2007. Accélération abstraite pour l'amélioration de la précision en analyse des relations linéaires. Phd thesis, Université Joseph Fourier, Grenoble.

Gonnord, L., 2009. The ASPIC tool: Accelerated symbolic polyhedral invariant computation, `http://laure.gonnord.org/pro/aspic/aspic.html`.

Gonnord, L., Halbwachs, N., 2006. Combining widening and acceleration in linear relation analysis. In: Static Analysis Symposium. Vol. 4134 of LNCS. Springer, pp. 144–160.

Granger, P., 1991. Static analysis of linear congruence equalities among variables of a program. In: TAPSOFT'91. Vol. 493 of LNCS. pp. 169–192.

Halbwachs, N., Lagnier, F., Raymond, P., 1993. Synchronous observers and the verification of reactive systems. In: Algebraic Methodology and Software Technology. Workshops in Computing. Springer, pp. 83–96.

Halbwachs, N., Proy, Y.-E., Roumanoff, P., 1997. Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11 (2), 157–185.

Jeannet, B., 2003. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. Formal Methods in System Design 23 (1), 5–37.

Jeannet, B., Jéron, T., Rusu, V., Zinovieva, E., 2005. Symbolic test selection based on approximate analysis. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 3440 of LNCS. Springer, pp. 349–364.

Jones, N. D., Gomard, C., Sestoft, P., 1993. Partial Evaluation and Automatic Program Generation. Prentice Hall International.

Kam, J. B., Ullman, J. D., 1977. Monotone data flow analysis frameworks. Acta Informatica 7, 305–317.

Leroux, J., 2003. Algorithmique de la vérification des systèmes à compteurs – approximation et accélération – implémentation dans l'outil FAST. Phd thesis, École Normale Supérieure de Cachan.

Leroux, J., Sutre, G., 2007. Acceleration in convex data-flow analysis. In: Foundations of Software Technology and Theoretical Computer Science. Vol. 4855 of LNCS. Springer, pp. 520–531.

Minsky, M. L., 1961. Recursive unsolvability of post's problem of 'tag' and other topics in theory of turing machines. Annals of Mathematics 74 (3), 437–455.

Schrammel, P., Jeannet, B., 2010. Extending abstract acceleration to data-flow programs with numerical inputs. In: Numerical and Symbolic Abstract Domains. Vol. 267 of ENTCS. Elsevier, pp. 101–114.

Schrammel, P., Jeannet, B., 2011. Logico-numerical abstract acceleration and application to the verification of data-flow programs. In: Static Analysis Symposium, SAS'11. Vol. 6887 of LNCS. Springer, pp. 233–248.