HAL
open science

Extending Abstract Acceleration Methods to Data-Flow Programs with Numerical Inputs

Peter Schrammel, Bertrand Jeannet

To cite this version:

Peter Schrammel, Bertrand Jeannet. Extending Abstract Acceleration Methods to Data-Flow Programs with Numerical Inputs. Numerical and Symbolic Abstract Domains, Sep 2010, Perpignan, France. pp.101-114, 10.1016/j.entcs.2010.09.009 . hal-00749914

HAL Id: hal-00749914

https://inria.hal.science/hal-00749914

Submitted on 8 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.

Replace this file with prentcsmacro.sty for your meeting, or with entcsmacro.sty for your meeting. Both can be found at the ENTCS Macro Home Page.

Extending Abstract Acceleration Methods to Data-Flow Programs with Numerical Inputs[1]

Peter Schrammel[2] Bertrand Jeannet[3]

INRIA Rhône-Alpes, Grenoble, 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 with Boolean and numerical variables, e.g. LUSTRE programs, firstly requires the enumeration of the Boolean states in order to obtain a control graph with numerical variables only. Secondly, acceleration methods have to deal with the non-determinism introduced by numerical input variables. In this article we address the latter problem by extending the concept of abstract acceleration of Gonnord et al. to numerical input variables.

Keywords: Static Analysis, Acceleration, Abstract Interpretation, Linear Relation Analysis

1 Introduction

This paper considers the reachability analysis of synchronous programs manipulating Boolean and numerical variables, and more generally the reachability analysis of *logico-numerical programs*, that are symbolic automata combining Boolean and numerical variables. The applications of such a reachability analysis are for instance the verification of safety properties [16] or model-based testing [19].

Abstract interpretation and acceleration. Since the reachability problem is not decidable for logico-numerical programs, two main approaches have been studied:

[1] This work was supported by the INRIA large-scale initiative SYNCHRONICS.
[2] Email: peter.schrammel@inrialpes.fr
[3] Email: bertrand.jeannet@inrialpes.fr

©2012 Published by Elsevier Science B. V.

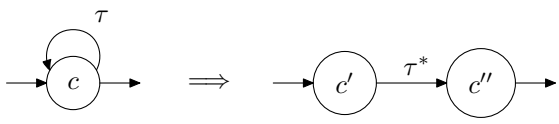

Fig. 1. Simple loop transition (left) and accelerated transition (right).

(i) Abstract interpretation techniques [6,7] compute only an over-approximation of the reachability set, but terminate always.
(ii) Acceleration techniques [21,2,3] compute the exact reachability set in favorable cases, but without 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 space. The key idea is to approximate sets of states X by an element Y of an *abstract domain*. A classical abstract domain for numerical invariants $X \in \wp(\mathbb{R}^n)$ is the domain of *convex polyhedra* $Pol(\mathbb{R}^n)$ [9], that can be represented as a conjunction of linear inequalities. 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 sequences, an extrapolation operator called *widening* is applied, which induces additional approximations.

The idea of *acceleration* is to accelerate cycles τ in the control structure of a program, by computing the effect of their transitive closure τ^* on a set of states, see Fig. 1. 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 the method is semi-complete: One starts enumerating and accelerating non-elementary cycles (which form an infinite set) in the hope of converging after a finite number of steps to the smallest fixed point. 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 [12,11,2], or FIFO queues using subclasses of regular expressions [4,1].

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 et al. [15,14] have proposed the concept of *abstract acceleration* which combines these approaches: Wherever possible, elementary loops are accelerated *in the abstract domain*, and in any other cases (nested loops, too expressive transitions) one resorts to the use of widening to guarantee the convergence of the approximated fixed point computation.

Abstract acceleration and logico-numerical programs. Acceleration

techniques such as [21,2] consider purely numerical automata. There are two shortcomings of the abstract acceleration approach when applied to logico-numerical programs such as Lustre [5] data-flow programs:

(1) In order to reduce such a program to a purely numerical automaton, all possible valuations of Boolean state variables need to be enumerated and encoded in a control graph. This *partitioning* and *partial evaluation* process may lead to a combinatorial explosion of control locations.

(2) The concept of input variables as encountered in Lustre programs requires an extension of the results of [15,14]. As opposed to Boolean input variables that can be encoded in an automaton by finite non-deterministic choices, numerical input variables demand a more specific treatment.

This article especially addresses point 2, although point 1 is our ultimate goal.

Contributions and outline. Our contribution is to extend the abstract acceleration concept as introduced in [15] to systems with numerical inputs, which raises some subtle points. In particular we show how to accelerate loops composed of a translation with resets *and inputs*, provided that the guard of the loop constrains separately state and input variables. Without this restriction indeed, one can emulate any affine transformation without inputs. After some preliminary notions in Section 2 about the considered program model, the operations on convex polyhedra and the general verification framework that we use for analysis, we recall the main results of abstract acceleration in Section 3. Section 4 details our contribution. We conclude in Section 6.

2 Analysis of Logico-Numerical Programs

Program model. We consider in this article programs modeled as a symbolic transition system $\begin{cases} \mathit{init}(\boldsymbol{s}) \\ \mathit{assert}(\boldsymbol{s},\boldsymbol{i}) \rightarrow \boldsymbol{s}' = \boldsymbol{f}(\boldsymbol{s},\boldsymbol{i}) \end{cases}$ where (1) $\boldsymbol{s}$ and $\boldsymbol{i}$ are vectors of state and input variables, that are either Boolean or real; (2) $\mathit{init}(\boldsymbol{s})$ is an initial condition on state variables; (3) $\mathit{assert}(\boldsymbol{s},\boldsymbol{i})$ is an *assertion* constraining input variables depending on state variables, and typically modeling the environment of the program; (4) $\boldsymbol{f}$ is the vector of transition functions.

An execution of such a system is a sequence $\boldsymbol{s^0} \xrightarrow{\boldsymbol{i^0}} \boldsymbol{s^1} \xrightarrow{\boldsymbol{i^1}} \ldots \boldsymbol{s^k} \xrightarrow{\boldsymbol{i^k}} \ldots$ such that $\mathit{init}(\boldsymbol{s^0})$ and for any $k \geq 0$, $\mathit{assert}(\boldsymbol{s^k},\boldsymbol{i^k}) \wedge \boldsymbol{s^{k+1}} = \boldsymbol{f}(\boldsymbol{s^k},\boldsymbol{i^k})$.

This program model corresponds for example to the output of the front-end compilation of synchronous data-flow programs like Lustre and includes various models of counter automata (by emulating locations using Boolean variables) [3]. A control graph manipulating only numerical variables can be generated from this program model by performing a partial evaluation [20] of all Boolean state variables (which are then encoded in control locations)

and eliminating Boolean input variables by non-deterministic choices. The partition refinement mechanics implemented in the NBac tool [18] are capable of achieving this task and have been employed for connecting the Aspic tool [14,13] to Lustre, for example.

Convex Polyhedra. We use in this paper the abstract domain of convex polyhedra for representing invariants on numerical variables. Besides classical operations (intersection, convex hull, assignments of variables by linear expressions, ...) described *e.g.* in [17], we will use the following operations:

The *time elapse* operation [17] $X \nearrow D = \{\mathbf{x} + t\mathbf{d} \,|\, \mathbf{x} \in X, \mathbf{d} \in D, t \in \mathbb{R}^{\geq 0}\}$ is practically implemented as follows: Let (V_X, R_X) respectively (V_D, R_D) be the systems of generators of the polyhedra X and D then $(V_X, R_X \cup V_D \cup R_D)$ is a system of generators of the polyhedron $X \nearrow D$.

The *Minkowski sum* [10] of two polyhedra $X = X_1 + X_2$ is defined as $X(\boldsymbol{x}) = \exists \boldsymbol{x_1}, \boldsymbol{x_2} \ : (\boldsymbol{x} = \boldsymbol{x_1} + \boldsymbol{x_2}) \wedge X_1(\boldsymbol{x_1}) \wedge X_2(\boldsymbol{x_2})$.

3 Overview of Acceleration and Abstract Acceleration

As mentioned in the introduction, the idea of *acceleration* (Fig. 1) is to replace a loop transition τ by a single transition τ^* that computes the transitive closure of τ. *Abstract acceleration* [15,14] relaxes exact acceleration in the sense that it aims at approximating the exact set $\tau^*(X)$ by its convex hull $\tau^{\otimes}(X) \supseteq \tau^*(X)$. This method is also inspired by the time elapse operator used in timed or in hybrid automata [17].

Following the notations of Section 2, a loop transition τ will have 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

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \rightarrow \mathbf{x}' = \mathbf{C}\mathbf{x} + \boldsymbol{d} \tag{1}$$

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

- a *reset* if C is the zero matrix,
- a *translation* if C is the identity matrix,
- a *translation with resets* (or translation/reset) if C is a diagonal matrix with zeros and ones only,
- a *periodic affine transformation* if $\exists p > 0 : \mathbf{C}^p = \mathbf{C}^{2p}$,
- a *general affine transformation* otherwise.

Existing acceleration methods cannot deal with general affine transformations. We will not discuss the case of periodic affine transformations, as it seems to be of limited practical interest.

In the context of abstract acceleration, [15,14] shows that translations

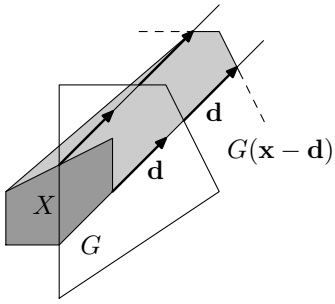

Fig. 2. Acceleration of a translation loop starting from X (dark shadowed) resulting in $\tau^{\otimes}(X)$ (whole shadowed area).

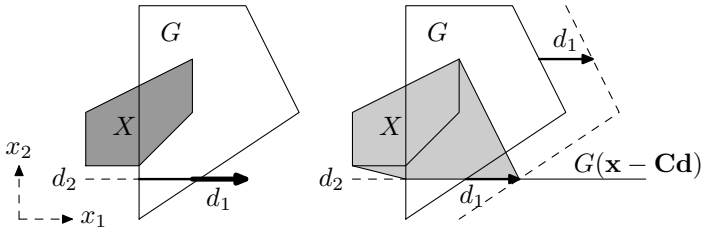

Fig. 3. Acceleration of a loop with translations/resets: On the left hand side, the application of $\tau(X)$ – here, with $x_1' = x_1 + d_1$ and $x_2' = d_2$, yields a polyhedron (bold line including arrow) containing the reset values. The accelerated transition gives $\tau^{\otimes}(S)$ (shadowed) on the right hand side.

(Fig. 2) and translations with resets (Fig. 3) can be accelerated as follows, with X denoting a convex polyhedron and $G(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ an affine guard (which is also a convex polyhedron):

Theorem 3.1 *Let τ be a translation $G \to \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{d}$. The convex polyhedron*

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

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

Theorem 3.2 *Let τ be a translation with resets $G \to \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$. The convex polyhedron*

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

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

Remark 3.3 Theorem 3.2 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 obtained formula.

Remark 3.4 Ideally, $\tau^{\otimes}(X)$ as defined in Theorems 3.1 and 3.2 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 $\tau : x_1 \leq 4 \to x_1' = x_1 + 2$. $\tau^{\otimes}(X) = [1, 6]$, whereas the best over-approximation of $\tau^*(X) = \{1, 3, 5\}$ is the interval $[1, 5]$. This is because the operations involved in $\tau^{\otimes}(X)$ manipulate dense sets and do not take into account arithmetic congruences. We will not improve on this in this work, but we will point out in our proofs where this dense approximation takes place.

These theorems can be applied on a control graph by dividing locations with a single self-loop (as shown in Fig. 1). If there are several self-loops in a location then the cases where guards overlap and where they are disjoint must be distinguished, which results in a more elaborate division of the location. [14] gives a range of methods for dealing with more complex cases.

4 Abstract Acceleration with Numerical Inputs

We now extend numerical abstract acceleration w.r.t. numerical input variables $\boldsymbol{y}$. This means that we consider transitions of the form

$$\underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{L} \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix}}_{\mathbf{A}\boldsymbol{x}+\mathbf{L}\boldsymbol{y}\leq\boldsymbol{b} \,\wedge\, \mathbf{J}\boldsymbol{y}\leq\boldsymbol{k}} \rightarrow \boldsymbol{x'} = \underbrace{\begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x}+\mathbf{T}\boldsymbol{y}+\boldsymbol{u}} \tag{2}$$

Note that the 0 in the matrix of the guard does not imply a loss of generality.

A fundamental observation is that 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" $(\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{y} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}) \rightarrow \boldsymbol{x'} = \boldsymbol{y}$. This means that there is no hope to get precise acceleration for such resets 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.

Nevertheless, we can accelerate transitions with inputs when the constraints on the state variables do not depend on the inputs, *i.e.* when $\mathbf{L} = 0$ in Eqn. (2) and the guard is of the form $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{y} \leq \boldsymbol{k}$. We call the resulting guards *simple guards*. Otherwise, we provide in Section 4.3 a weaker over-approximation of the exact result for general guards.

4.1 Translations with inputs and simple guards

These are defined by $\begin{pmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix} \rightarrow \boldsymbol{x'} = \begin{pmatrix} \mathbf{I} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} + \boldsymbol{u}$ where I is the identity matrix.

First of all, assume that in a translation without inputs $\boldsymbol{d}$ is not constant, but constrained to be inside a convex polyhedron D. Then Theorem 3.1 can be generalized to such *polyhedral translations*.

Proposition 4.1 *Let τ be a transition $G \rightarrow \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{d}$ with $G(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$, $\boldsymbol{d} \in D$ and D a convex polyhedron. The set*

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

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

Note that $\tau(X)$ can be implemented by standard polyhedra operations: $\tau(X) = (X \sqcap G) + D$.

Proof. In the proof, the guard G is seen alternatively as a predicate or a set.

$$
\begin{aligned}
& \boldsymbol{x'} \in \bigsqcup_{k\geq 1} \tau^k(X) \\
\Leftrightarrow\ & \exists k \geq 1, \exists \boldsymbol{x_0} \in X, \exists \boldsymbol{d_1}, \ldots, \boldsymbol{d_k} \in D \ : \begin{cases} \boldsymbol{x'} = \boldsymbol{x_0} + \sum_{j=1}^{k} \boldsymbol{d_j} \ \wedge\ G(\boldsymbol{x_0}) \ \wedge \\ \forall k' \in [1, k-1] \ :\ G(\boldsymbol{x_0} + \sum_{j=1}^{k'} \boldsymbol{d_j}) \end{cases} \\
\Leftrightarrow\ & \exists k \geq 1, \exists \boldsymbol{x_0} \in X, \exists \boldsymbol{d}, \boldsymbol{d_k} \in D, \exists \boldsymbol{x_{k-1}} \ : \\
& \quad \boldsymbol{x_{k-1}} = \boldsymbol{x_0} + (k-1)\boldsymbol{d} \ \wedge\ G(\boldsymbol{x_0}) \ \wedge\ G(\boldsymbol{x_{k-1}}) \ \wedge\ \boldsymbol{x'} = \boldsymbol{x_{k-1}} + \boldsymbol{d_k} \\
& \quad \text{(because } D \text{ and } G \text{ are convex)} \\
\Rightarrow\ & \exists \alpha \geq 0, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d'} \in D, \exists \boldsymbol{x''} \ : \\
& \quad \boldsymbol{x''} = \boldsymbol{x_0} + \alpha \boldsymbol{d} \ \wedge\ \boldsymbol{x'} = \boldsymbol{x''} + \boldsymbol{d'} \ \wedge\ G(\boldsymbol{x''}) \qquad \text{(dense approximation)} \\
\Leftrightarrow\ & \exists \boldsymbol{x''} \in \big((X \sqcap G) \nearrow D\big) \sqcap G, \exists \boldsymbol{d'} \in D \ :\ \boldsymbol{x'} = \boldsymbol{x''} + \boldsymbol{d'} \\
\Leftrightarrow\ & \boldsymbol{x'} \in \Big(\big((X \sqcap G) \nearrow D\big) \sqcap G\Big) + D
\end{aligned}
$$

We conclude by observing that $\Big(\big((X \sqcap G) \nearrow D\big) \sqcap G\Big) + D = \tau\big((X \sqcap G) \nearrow D\big)$. $\square$

Mind that the only approximation takes place in the line ($\Rightarrow$) where the integer coefficient $k-1 \geq 0$ is replaced by a dense coefficient $\alpha \geq 0$. This is the technical explanation of Remark 3.4.

Remark 4.2 One might think that Theorem 3.1 can be applied directly by accelerating the transition for each $\boldsymbol{d} \in D$ and taking the union, i.e. computing $\tau^*(X)$ by $X \sqcup \bigsqcup_{\boldsymbol{d} \in D} X_{\boldsymbol{d}}$ with $X_{\boldsymbol{d}} = \big((X \sqcap G) \nearrow \boldsymbol{d}\big) \sqcap G(\boldsymbol{x} - \boldsymbol{d})$, but there is a subtle difference: This formula computes the correct set for all states reachable within G, but for the last step crossing the border of G it allows only those vectors d having been used for the previous iterations, whereas actually there is a choice among all $d \in D$.

The following proposition reduces translations with inputs to generalized translations:

Proposition 4.3 *A translation with inputs and a simple guard τ is equivalent to a polyhedral translation without inputs defined by*

$$
\begin{cases} \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{d} \in D \rightarrow \mathbf{x}' = \boldsymbol{x} + \boldsymbol{d} \\ D = \{\boldsymbol{d} \mid \exists \boldsymbol{y} : \boldsymbol{d} = \mathbf{T}\mathbf{y} + \mathbf{u} \wedge \mathbf{J}\mathbf{y} \leq \mathbf{k}\} \end{cases}
$$

Note that D can be computed by standard polyhedra operations.

Proof.

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

$\square$

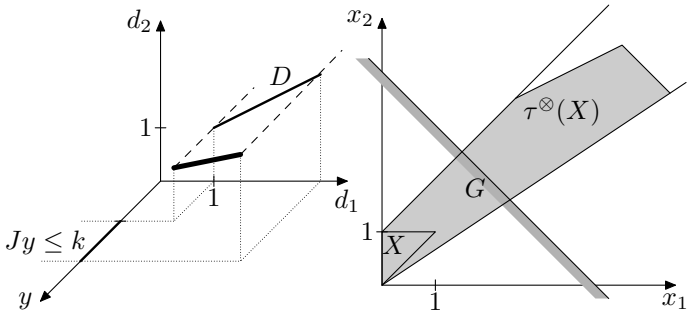

Fig. 4. Translation with inputs: Example 4.5: The left hand side shows the transformation of the inputs: $\boldsymbol{Jy} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{y} + \boldsymbol{u}$ (bold line) is projected on variables $\boldsymbol{d}$. The shadowed area in the right figure is the result of the accelerated transition $\tau^{\otimes}(X)$.

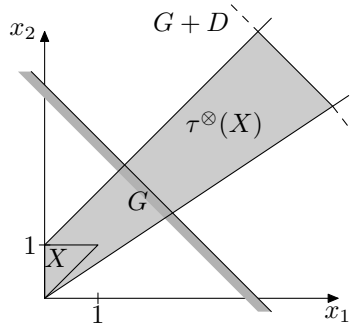

Fig. 5. Precision loss in example 4.5 when using the approximate formula according to remark 4.6.

Theorem 4.4 *The accelerated transition $\tau^{\otimes}$ for a translation with inputs and simple guard τ can be computed by applying Propositions 4.1 and 4.3.*

Example 4.5 Consider the polyhedron $X = \{(x_1, x_2) \mid 0 \leq x_1 \leq x_2 \leq 1\}$ and the transition $\tau : \left| \begin{array}{r} x_1 + x_2 \leq 4 \\ 1 \leq y \leq 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2y - 1 \\ x_2' = x_2 + y \end{array} \right.$. Eliminating the inputs as in Proposition 4.3 yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 3 \ \wedge \ -d_1 + 2d_2 = 1\}$, see Fig. 4 left. After translation of X by D (Fig. 4 right) we obtain the polyhedron $\{(x_1, x_2) \mid x_1 \geq 0 \ \wedge \ -x_1 + x_2 \leq 1 \ \wedge \ x_1 + x_2 \leq 9 \ \wedge \ -2x_1 + 4x_2 \leq 9 \ \wedge \ 2x_1 - 3x_2 \leq 0\}$.

Remark 4.6 In analogy to Theorem 3.1, we could consider the formula $X \sqcup ((((X \sqcap G) \nearrow D) \sqcap (G + D))$. In order to justify this, we extend the proof of Proposition 4.1 continuing at the label (dense approximation):

$$\begin{aligned}
&\Leftrightarrow \exists \alpha \geq 0, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d'} \in D : \boldsymbol{x'} = \boldsymbol{x_0} + \alpha \boldsymbol{d} + \boldsymbol{d'} \ \wedge \ G(\boldsymbol{x'} - \boldsymbol{d'}) \\
&\Rightarrow (\exists \alpha \geq 0, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d'} \in D : \boldsymbol{x'} = \boldsymbol{x_0} + \alpha \boldsymbol{d} + \boldsymbol{d'}) \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\exists \boldsymbol{d'} \in D : G(\boldsymbol{x'} - \boldsymbol{d'})) \\
&\Rightarrow \boldsymbol{x'} \in (X \sqcap G) \nearrow D \ \wedge \ \boldsymbol{x'} \in (G + D)
\end{aligned}$$

using $\{\boldsymbol{x} \mid \exists \boldsymbol{d} \in D \ \wedge \ G(\boldsymbol{x} - \boldsymbol{d})\} = \{\boldsymbol{z} + \boldsymbol{d} \mid \boldsymbol{d} \in D \wedge G(\boldsymbol{z})\} = (G + D)$. But it can be observed that for the translation of example 4.5 the latter formula results in an over-approximation (see Fig. 5) as compared to the result in Fig. 4. This reflects the additional approximation steps in the proof.

4.2 Translations/Resets with inputs and simple guards

These are defined by $\begin{pmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix} \rightarrow \boldsymbol{x'} = \begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} + \boldsymbol{u}$ where C is a diagonal matrix with $C_{i,i} \in \{0, 1\}$.

Notations. Let $\mathbf{C'} = \mathbf{I} - \mathbf{C}$ with $\mathbf{I}$ the identity matrix. Any vector $\boldsymbol{x}$ can be decomposed in $\boldsymbol{x} = \boldsymbol{x^t} + \boldsymbol{x^r}$ with $\boldsymbol{x^t} = \mathbf{C}\boldsymbol{x}$ and $\boldsymbol{x^r} = \mathbf{C'}\boldsymbol{x}$. We extend

such notations to sets: $X^t = \{\boldsymbol{x^t} \mid \boldsymbol{x} \in X\}$ and $X^r = \{\boldsymbol{x^r} \mid \boldsymbol{x} \in X\}$. If I denotes the set of dimensions, $I_t = \{i \in I \mid C_{i,i} = 1\}$ and $I_r = I \setminus I_t$ are the set of translated and reset dimensions. Any set X can be approximated by the Minkowski sum $X^t + X^r$, which can also be seen as the Cartesian product of X projected on the subspace of translated dimensions and X projected on the subspace of reset dimensions.

This case can be handled in a way similar to Section 4.1: We combine Theorem 3.2 and Proposition 4.3 reducing translations/resets with inputs to generalized translations/resets without inputs. Mind, however, that remark 3.3 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 4.7 . *Let τ be a translation with resets $G \to \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ with $G(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$, $\boldsymbol{d} \in D$ and D a convex polyhedron. The set*

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

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

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

$$\begin{aligned}
& \boldsymbol{x'} \in \textstyle\bigsqcup_{k\geq 2} \tau^k(X) \\
\Leftrightarrow\ & \exists k \geq 2, \exists \boldsymbol{x_0} \in X, \exists \boldsymbol{d_1} \ldots \boldsymbol{d_k} \in D, \exists \boldsymbol{x_1} \ldots \boldsymbol{x_k} \ : \\
& \quad \begin{cases} \boldsymbol{x'} = \boldsymbol{x_k} \\ \wedge\ \forall k' \in [1,k] \ : \ \begin{cases} \boldsymbol{x_{k'}}^i = \boldsymbol{x_0}^i + \sum_{j=1}^{k'} \boldsymbol{d_j}^i & \text{for } i \in I^t \\ \boldsymbol{x_{k'}}^i = \boldsymbol{d_{k'}}^i & \text{for } i \in I^r \end{cases} \\ \wedge\ G(\boldsymbol{x}) \wedge \forall k' \in [1,k-1] \ : \ G(\boldsymbol{x_{k'}}) \end{cases} \\
\Leftrightarrow\ & \exists k \geq 2, \exists \boldsymbol{x_0} \in X, \exists \boldsymbol{d_1} \ldots \boldsymbol{d_k} \in D, \exists \boldsymbol{x_1} \ldots \boldsymbol{x_k} \ : \\
& \quad \begin{cases} \boldsymbol{x'} = \boldsymbol{x^t} + (\sum_{j=1}^{k} \boldsymbol{d_j^t}) + \boldsymbol{d_k^r} \\ \wedge\ G(\boldsymbol{x}) \wedge \forall k' \in [1,k-1] \ : \ G(\boldsymbol{x^t} + (\sum_{j=1}^{k'} \boldsymbol{d_j^t}) + \boldsymbol{d_{k'}^r}) \end{cases} \\
\Rightarrow\ & \exists k \geq 2, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d_1}, \boldsymbol{d_k} \in D, \exists \boldsymbol{d_2^t} \ldots \boldsymbol{d_{k-1}^t} \in D^t, \exists \boldsymbol{d_2^r} \ldots \boldsymbol{d_{k-1}^r} \in D^r, \\
& \exists \boldsymbol{x_1} \ldots \boldsymbol{x_k} \ : \begin{cases} \boldsymbol{x_1} = \boldsymbol{x_0^t} + \boldsymbol{d_1} \\ \wedge\ \forall k' \in [2,k-1] \ : \ \boldsymbol{x_{k'}} = \boldsymbol{x_1^t} + (\sum_{j=2}^{k'} \boldsymbol{d_j^t}) + \boldsymbol{d_{k'}^r} \\ \wedge\ \boldsymbol{x'} = \boldsymbol{x_{k-1}^t} + \boldsymbol{d_k} \\ \wedge\ \forall k' \in [1,k-1] \ : \ G(\boldsymbol{x_{k'}}) \end{cases} \\
& (D \text{ approximated by the sum } (D^t + D^r) \text{ for } k' \in [2,k-1])
\end{aligned}$$

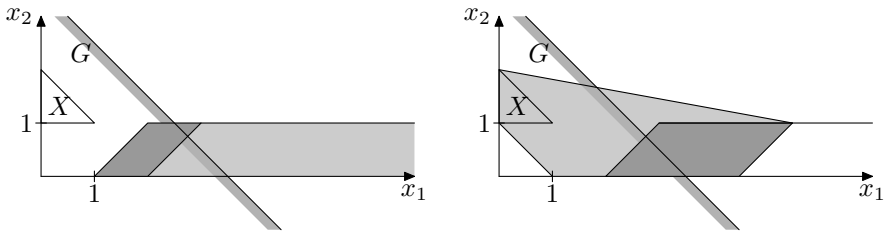

Fig. 6. Translation/reset with inputs: Example 4.9. Left hand side: $\tau(X)$ (dark shadowed) and $((\tau(X) \sqcap G)^t \nearrow D^t) + D^r$ (whole shadowed area). Right hand side: $\tau(((\tau(X) \sqcap G)^t \nearrow D^t) + D^r)$ (dark shadowed) and $\tau^{\otimes}(X)$ (whole shadowed area).

$$\Leftrightarrow \exists k \geq 2, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d_1}, \boldsymbol{d_k} \in D, \exists \boldsymbol{d^t} \in D^t, \exists \boldsymbol{d^r_2} \ldots \boldsymbol{d^r_{k-1}} \in D^r, \exists \boldsymbol{x_1} \ldots \boldsymbol{x_k} :$$
$$\begin{cases} \boldsymbol{x_1} = \boldsymbol{x^t_0} + \boldsymbol{d_1} \ \wedge \ \boldsymbol{x_{k-1}} = \boldsymbol{x^t_1} + (k-2)\boldsymbol{d^t} + \boldsymbol{d^r_{k-1}} \\ \wedge\ G(\boldsymbol{x_1}) \wedge G(\boldsymbol{x_{k-1}}) \\ \wedge\ \boldsymbol{x'} = \boldsymbol{x^t_{k-1}} + \boldsymbol{d_k} \end{cases}$$
(because D^t and G are convex)

$$\Rightarrow \exists \alpha \geq 0, \exists \boldsymbol{x_0} \in X \sqcap G, \exists \boldsymbol{d_1}, \boldsymbol{d'} \in D, \exists \boldsymbol{d^t} \in D^t, \exists \boldsymbol{d^r} \in D^r, \exists \boldsymbol{x''} :$$
$$\begin{cases} \boldsymbol{x_1} = \boldsymbol{x^t_0} + \boldsymbol{d_1} \ \wedge \ \boldsymbol{x''} = \boldsymbol{x^t_1} + \alpha \boldsymbol{d^t} + \boldsymbol{d^r} \ \wedge \ G(\boldsymbol{x_1}) \ \wedge \ G(\boldsymbol{x''}) \\ \wedge\ \boldsymbol{x'} = \boldsymbol{x''^t} + \boldsymbol{d'} \end{cases}$$
(dense over-approximation)

$$\Leftrightarrow \exists \alpha \geq 0, \exists \boldsymbol{x_1} \in \tau(X) \sqcap G, \exists \boldsymbol{d'} \in D, \exists \boldsymbol{d^t} \in D^t, \exists \boldsymbol{d^r} \in D^r, \exists \boldsymbol{x''} :$$
$$\boldsymbol{x''} = \boldsymbol{x^t_1} + \alpha \boldsymbol{d^t} + \boldsymbol{d^r} \ \wedge \ G(\boldsymbol{x''}) \ \wedge \ \boldsymbol{x'} = \boldsymbol{x''^t} + \boldsymbol{d'}$$
$$\Leftrightarrow \exists \boldsymbol{x''} \in \Big(((\tau(X) \sqcap G)^t \nearrow D^t) + D^r \Big) \sqcap G, \exists \boldsymbol{d'} \in D, \boldsymbol{x'} = \boldsymbol{x''^t} + \boldsymbol{d'}$$
$$\Leftrightarrow \boldsymbol{x'} \in \Big(\Big(((\tau(X) \sqcap G)^t \nearrow D^t) + D^r \Big) \sqcap G \Big)^t + D$$

The last expression is equal to $\tau\Big(((\tau(X) \sqcap G)^t \nearrow D^t) + D^r \Big)$. □

Theorem 4.8 *The accelerated transition $\tau^{\otimes}$ for a translation/reset with inputs and a simple guard τ can be computed by applying Proposition 4.7 with D defined as in Proposition 4.3.*

Example 4.9 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 $\tau : \left| \begin{array}{c} 2x_1 + 2x_2 \leq 7 \\ 0 \leq y \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x'_1 = x_1 + y + 1 \\ x'_2 = y \end{array} \right.$.
Eliminating the inputs yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \ \wedge \ d_1 - d_2 = 1\}$ and $D^t = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \ \wedge \ d_2 = 0\}$. We obtain $\tau^{\otimes}(X) = \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \ \wedge \ x_2 \geq 0 \ \wedge \ 2x_1 - 2x_2 \leq 9 \ \wedge \ 2x_1 + 11x_2 \leq 22 \ \wedge \ x_1 \geq 0\}$, see Fig. 6.

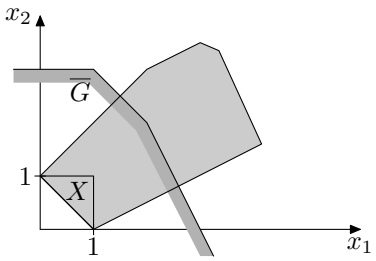

Fig. 7. Example 4.10: Accelerated transition $\tau^{\otimes}(X)$ using the weakened guard $\overline{G}$ (result shadowed).

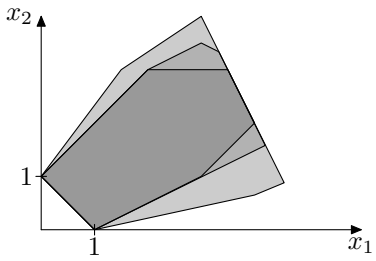

Fig. 8. Example 4.10: comparison between convex hull of the exact result (dark grey), our method (grey), and widening with no delay and 3 (!) descending iterations (light grey)

4.3 Weakening general guards to simple guards

As discussed at the beginning of Section 4, allowing constraints on both state and input variables in guards ($\mathbf{L} \neq 0$ in Eqn. (2)) makes acceleration very difficult. Our solution is to weaken the guard $G(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{A}\boldsymbol{x} + \mathbf{L}\boldsymbol{y} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{y} \leq \boldsymbol{k}$ by the simple guard (or cartesian product) $\bar{G} = \underbrace{(\exists \boldsymbol{y} : G)}_{\mathbf{A}'\boldsymbol{x} \leq \boldsymbol{b}'} \wedge \underbrace{(\exists \boldsymbol{x} : G)}_{\mathbf{J}'\boldsymbol{y} \leq \boldsymbol{k}'}$ and to apply Theorems 4.4 and 4.8. This trivially results in a sound over-approximation because a weaker guard is used for abstract acceleration.

Example 4.10 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 $\tau(X) : \left| \begin{array}{r} 2x_1 + x_2 + y \leq 6 \\ x_2 - y \leq 2 \\ 0 \leq y \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + y + 1 \\ x_2' = x_2 + 1 \end{array} \right.$.
The weakened guard is $\bar{G} = (2x_1 + x_2 \leq 6 \wedge x_1 + x_2 \leq 4 \wedge x_2 \leq 3) \wedge (0 \leq y \leq 1)$. 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. 7. The convex hull of the exact result is $\{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge -2 \leq x_2 - x_1 \leq 1 \wedge x_1 - 2x_2 \leq 1 \wedge x_2 \leq 3 \wedge 2x_1 + x_2 \leq 10\}$, see Fig. 8.

5 Comparison with widening

The standard widening operator for convex polyhedra and refinements of it like limited widening [17] [4] may sometimes lead to good results. In this section, we compare the acceleration and the widening approaches on Examples 4.9 and 4.10. Analyzing such a program using widening after a number N of

[4] Limited widening is also called widening with thresholds [8].

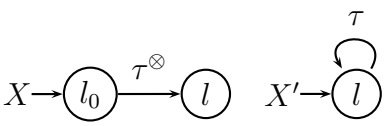

Fig. 9. Analysis with acceleration (left) and with widening (right) for Examples 4.9 and 4.10

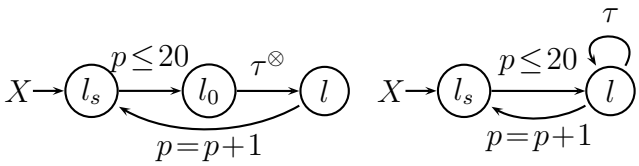

Fig. 10. Analysis with acceleration (left) and with widening (right) for Example 5.1.

initial steps resorts to computing the limit of the sequences

$$Y_0 = X \qquad\qquad Z_0 = Y_N$$
$$Y_{n+1} = X \sqcup \tau(Y_n) \quad \text{for } n < N \qquad\qquad Z_{n+1} = Z_n \nabla (Z_n \sqcup \tau(Z_n))$$

in which X_n, Y_n, Z_n are associated with location l on Fig. 9. The technical properties of the widening operator ∇ guarantee that the sequence $(Z_n)_{n\geq 0}$ converges in a finite number of steps to Z_∞ [7], which is an over-approximation of the reachable valuations at location l. This result may be improved by computing the first elements of the sequence $W_0 = Z_\infty, W_{n+1} = X \sqcup \tau(W_n)$, which does not necessarily converge.

Translation/reset with inputs and simple guard

If we compute the sequences defined above in the context of Example 4.9, we obtain with $N = 0$

$$Z_\infty = Z_1 = \{(x_1, x_2) \mid x_1 \geq 0\}$$
$$W_1 = \{x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 1 \wedge x_2 \leq 2\} \qquad W_\infty = W_2 = \tau^{\otimes}(X)$$

Delaying widening by one step ($N = 1$) improves the result for Z_∞ and makes the sequence $(W_n)_{n\geq 0}$ converge in only one step:

$$Z_\infty = Z_1 = \{x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 1\}$$
$$W_\infty = W_1 = \tau^{\otimes}(X)$$

In both cases Z_∞ is clearly much less precise than the result obtained by acceleration: neither x_1 nor x_2 get upper bound (to be compared with Fig. 6).

One or two descending iterations allow to get the same result as the one obtained by acceleration. However, it should be pointed out that if this loop is a program fragment, for instance embedded in an outer loop as in Fig. 10, *it is not possible any more* to apply a descending iteration in the middle of an ascending iteration (otherwise convergence is not guaranteed). Moreover, the acceleration technique is more efficient computationally (in particular it does not require convergence tests), and it has a monotonic behavior, which is not the case of widening.

Example 5.1 To illustrate these points, we consider the program depicted on Fig. 10 in which the inner loop τ is adapted from Example 4.9:

$$\tau : \left| \begin{array}{r} 2x_1 + 2x_2 \leq p \\ 0 \leq y \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + y + 1 \\ x_2' = y \\ p' = p \end{array} \right. , \; X = \left\{ (x_1, x_2, p) \left| \begin{array}{l} 0 \leq x_1 \;\wedge\; 1 \leq x_2 \\ x_1 + x_2 \leq 2 \wedge p = 1 \end{array} \right. \right\}$$

In both cases we apply widening on location l with a delay $N = 1$, and we perform one descending iteration after convergence of the ascending iteration. Without acceleration, we obtain a very weak invariant:

$$Z_\infty = \{(x_1, x_2, p) \mid x_1 \geq 0 \wedge p \geq 1\} \qquad\qquad W_\infty = W_1 = Z_\infty$$

With acceleration we obtain much better results:

$$\begin{aligned} Z'_\infty &= Z_\infty \cap \{(x_1, x_2, p) \mid x_1 + x_2 \geq 1 \wedge x_1 - x_2 \leq 4 \wedge x_1 + 5x_2 \leq 10\} \\ W'_\infty = W'_1 &= Z'_\infty \cap \{(x_1, x_2, p) \mid p \leq 20\} \end{aligned}$$

One can also consider widening with thresholds, that keeps in the result of the widening operation the subset of a fixed set of *threshold constraints* that are satisfied by both of its arguments. In the case of Example 4.9, a natural threshold constraint set is defined by the postcondition of the guard of τ by the body of τ, which is just $\tau(\top) = \{(x_1, x_2) \mid 0 \leq x_2 \leq 1\}$. Using it with $N = 0$ one obtains the same Z_∞ as with standard widening applied with $N = 1$. On Example 5.1 and with the same threshold set extended with $\{p \leq 21\}$, the results are improved but are still less precise than those obtained by combining acceleration and widening (in particular the descending iteration does not converge).

Translation with inputs and non-simple guard

In the context of Example 4.10, we obtain with $N = 0$:

$$\begin{aligned} Z_\infty = Z_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1\} \\ W_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge x_2 \leq 4 \wedge \\ &\qquad 0 \leq x_1 \leq 6 \wedge 3x1 + 5x_2 \geq 3\} \\ \dots \\ W_3 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge 3x_2 - 2x_1 \leq 6 \wedge 3x_2 - 4x_1 \leq 3 \wedge \\ &\qquad 5x_1 - 22x_2 \leq 8 \wedge 29x_1 - 157x_2 \leq 29\} \sqsupset \tau^{\otimes}(X) \end{aligned}$$

Again Z_∞ is very unprecise, but here the descending iteration does not converge (even if we use widening with thresholds), see Fig. 8 for W_3. If we use $N = 1$, then Z_∞ is more precise, and $W_\infty = W_1 = \tau^{\otimes}(X)$.

These results are just small experiments, but they illustrate the sensitiveness of widening (if we delay it, it might improve the result, but this is not

guaranteed either because it is not monotonic) and the fact that if the loop is part of a more complex program, the result might be much less precise.

6 Conclusion

We have presented an extension of abstract acceleration to numerical inputs. 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. Our methods are ready for use in purely numerical automata by adopting the partitioning methods from [14] for treating more complex cycles than the case of single self-loops that we have presented in this article. Moreover, limiting ourselves to convex guards and inputs that are contained in convex polyhedra is not a theoretical restriction of our methods, since non-convex polyhedra can always be decomposed in convex ones.

Acceleration vs. Widening. From a theoretical point of view acceleration has – in contrast to widening – some advantageous properties that underpin its utility as an auxiliary technique for treating loops: First, a better precision can be obtained since it directly exploits information from the program. Second, the number of iterations in fixed point computation decreases, because accelerating transitions effectively removes loops in the control graph. Furthermore, widening is not a monotonous operator, whereas acceleration is, which makes approximations more regular and predictable. Practical experience has to be gained in order to estimate the degree of improvement attained by these properties. Nonetheless, we have to resort to widening for non-accelerable transitions in order to ensure convergence.

Future work. As mentioned in the introduction, finding an appropriate control graph is the second issue (we have not dealt with in this article) in applying acceleration to logico-numerical programs, such as synchronous dataflow programs without explicit control flow. On the one hand the control graph should allow for a reasonably precise reachability analysis, on the other hand it should enable the use of abstract acceleration of numerical variables, while remaining sufficiently symbolic w.r.t. Boolean variables in order to prevent a combinatorial explosion. Our idea is to heuristically identify sets of states which behave like timed or linear hybrid automata, such that abstract acceleration can be applied.

Another direction is the extension to input variables of some results of [14] on the combined acceleration of several self-loops around the same location.

References

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

[2] Bardin, S., A. Finkel, J. Leroux and L. Petrucci, *FAST: Fast acceleration of symbolic transition systems*, in: *Computer-Aided Verification (CAV)* (2003), pp. 118–121.

[3] Bardin, S., A. Finkel, J. Leroux and P. Schnoebelen, *Flat acceleration in symbolic model checking*, in: *Automated Technology for Verification and Analysis (ATVA)*, LNCS **3707** (2005), pp. 474–488.

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

[5] Caspi, P., D. Pilaud, N. Halbwachs and J. A. Plaice, *LUSTRE: a declarative language for real-time programming*, in: *Principles of Programming Languages (POPL)* (1987), pp. 178–188.

[6] Cousot, P. and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *Principles of Programming Languages (POPL)*, 1977, pp. 238–252.

[7] Cousot, P. and R. Cousot, *Abstract interpretation and application to logic programs*, Journal of Logic Programming **13** (1992), pp. 103–179.

[8] Cousot, P. and R. Cousot, *Comparing the Galois connection and widening/narrowing approaches to abstract interpretation*, in: *PLILP'92*, LNCS **631**, 1992, pp. 269–295.

[9] Cousot, P. and N. Halbwachs, *Automatic discovery of linear restraints among variables of a program*, in: *Principles of Programming Languages (POPL)* (1978), pp. 84–97.

[10] de Berg, M., O. Cheong, M. van Kreveld and M. Overmars, "Computational Geometry: Algorithms and Applications," Springer, 2008, 3rd edition.

[11] Finkel, A. and J. Leroux, *How to compose Presburger-accelerations: Applications to broadcast protocols*, in: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Lecture Notes in Computer Science **2556** (2002), pp. 145–156.

[12] Fribourg, L. and H. Olsén, *Proving safety properties of infinite state systems by compilation into presburger arithmetic*, in: *Conference on Concurrency Theory (CONCUR)*, LNCS **1243** (1997), pp. 213–227.

[13] Gonnord, L., *The ASPIC tool: Accelerated Symbolic Polyhedral Invariant Computation*, http://laure.gonnord.org/pro/aspic/aspic.html.

[14] Gonnord, L., "Accélération abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires," Thèse de doctorat, Université Joseph Fourier, Grenoble (2007).

[15] Gonnord, L. and N. Halbwachs, *Combining widening and acceleration in linear relation analysis*, in: *Static Analysis Symposium (SAS)*, Seoul, Korea, 2006, pp. 144–160.

[16] Halbwachs, N., F. Lagnier and P. Raymond, *Synchronous observers and the verification of reactive systems*, in: *Methodology and Software Technology (AMAST)* (1993), pp. 83–96.

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

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

[19] Jeannet, B., T. Jéron, V. Rusu and E. Zinovieva, *Symbolic test selection based on approximate analysis*, in: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS **3440**, 2005, pp. 349–364.

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

[21] Leroux, J., "Algorithmique de la vérification des systèmes à compteurs – Approximation et accélération – Implémentation dans l'outil FAST," Thèse de doctorat, École Normale Supérieure de Cachan (2003).