

## Combining Widening and Acceleration in Linear Relation Analysis <sup>\*</sup>

Laure Gonnord and Nicolas Halbwachs<sup>\*\*</sup>

Vérimag<sup>\*\*\*</sup>, Grenoble – France

**Abstract.** Linear Relation Analysis [CH78, Hal79] is one of the first, but still one of the most powerful, abstract interpretations working in an infinite lattice. As such, it makes use of a widening operator to enforce the convergence of fixpoint computations. While the approximation due to widening can be arbitrarily refined by delaying the application of widening, the analysis quickly becomes too expensive with the increase of delay. Previous attempts at improving the precision of widening are not completely satisfactory, since none of them is guaranteed to improve the precision of the result, and they can nevertheless increase the cost of the analysis. In this paper, we investigate an improvement of Linear Relation Analysis consisting in computing, when possible, the exact (abstract) effect of a loop. This technique is fully compatible with the use of widening, and whenever it applies, it improves both the precision and the performance of the analysis.

Linear Relation Analysis [CH78, Hal79] (LRA) is one of the very first applications of abstract interpretation [CC77], and aims at computing an upper approximation of the reachable states of a numerical program, as a convex polyhedron (or a set of such polyhedra). It was applied in various domains like compile-time error detection [DRS01], program parallelization [IJT91], automatic verification [HPR97, HHWT97] and formal proof [BBC<sup>+</sup>00, BBM97].

Like any approximate verification method, LRA is faced with the compromise between precision and cost. Since its relatively high cost restricts its applicability, any situation where the precision can be improved at low cost must be exploited. One source of approximation in LRA is widening, the operator that ensures the termination of iterative computations, by extrapolating an upper approximation of their limit. When the approximation due to widening is the cause of the lack of precision of the result of an analysis, a possible way to improve the precision is to delay widening: instead of applying it at each iteration, one can start with a number of steps without widening, thus providing a more precise basis for subsequent extrapolations. Now, delaying widening is generally very expensive:

---

<sup>\*</sup> This work has been partially supported by the APRON project of the “ACI Sécurité Informatique” of the French Ministry of Research

<sup>\*\*</sup> email: {Laure.Gonnord, Nicolas.Halbwachs}@imag.fr

<sup>\*\*\*</sup> Verimag is a joint laboratory of Université Joseph Fourier, CNRS and INPG associated with IMAG.

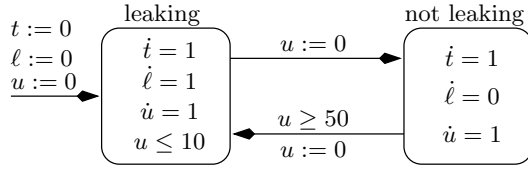
not only does it increase the number of iterations, but, more importantly, it leads to the construction of much more complex polyhedra (that would be simplified otherwise thanks to widening). So, if we can find some cheap ways to improve the precision of widening, we may not only improve the overall precision, but also avoid the cost of delaying widening.

The next question then is “what is a better widening?”. The fact that one single application of a widening operator gives smaller results [BHRZ03] does not necessarily mean that its repeated application will involve a convergence towards a more precise limit (an example can be seen in [SSM04]). Moreover, the use of such a widening is likely to slow down the convergence, by increasing the number of necessary iterations.

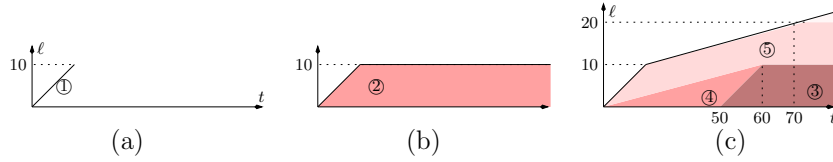
These remarks led us to look at situations where the widening can obviously be improved — in the sense that a faster convergence towards a better limit can be archived — at low cost with respect to the cost of usual polyhedra operators. For that, a source of inspiration are the so-called “acceleration techniques” proposed by several authors [BW94,WB98,CJ98,FS00,BFLP03]. These works consist in identifying categories of loops whose effect can be computed exactly. Roughly speaking, the effect of a simple loop, guarded by a linear condition on integer variables, and consisting of incrementations/decrementations of these variables can be computed exactly as a Presburger formula. These methods have the advantage of giving exact results. Now, because they are exact, they are restricted to some classes of programs (e.g., “flat counter automata”, i.e., without nested loops). Moreover, the exact computation with integer variables has a very high complexity (generally double-exp). So the applicability of these methods is somewhat limited.

In this paper, we investigate the use of acceleration methods in LRA, *in complement to widening*. Of course, when the effect of a loop can be computed exactly (and at low cost) there is no need to approximate it. Now, since we want to integrate these results in LRA, only the exact *abstract* effect of the loop is necessary, that is the *convex hull* of the reachable states during or after the loop. This means that we won’t use expensive computations in Presburger arithmetic. Moreover, we only look for an improvement of standard LRA: wherever an acceleration is possible, its application will improve the results, but the resulting method will not be restricted to those programs where acceleration applies everywhere.

To illustrate our goal better, let us consider a very simple example, the classical “leaking gas burner” [CHR91]: one wants to model and analyze the assumption that, whenever the “gas burner” leaks, the leakage is fixed within 10 seconds, and that the minimum interval between two leakages is 50 seconds. The standard modelling of this system is by a linear hybrid automaton [ACH<sup>+</sup>95,HHWT97] (see Fig. 1). The linear relation analysis of this hybrid automaton proceeds as follows (the successive results are projected onto the variables  $t$  and  $\ell$ , which represent, respectively, the global time elapsed and the global leaking time, the variable  $u$  being a local variable used to count the time elapsed in each location):



**Fig. 1.** Hybrid automaton of the gas burner



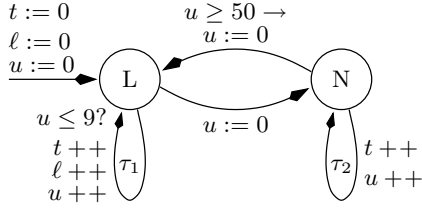
**Fig. 2.** Analysis of the hybrid gas burner

At first step, the location “leaking” is reached with the single point  $\{t = \ell = 0\}$ , and the “time elapse” operator<sup>1</sup> gives the polyhedron  $\{0 \leq t = \ell \leq 10\}$  (①, Fig. 2.a). So, the location “not leaking” is reached with  $\{0 \leq t = \ell \leq 10\}$ , and the “time elapse” operator provides  $\{0 \leq \ell \leq 10, t \geq \ell\}$  (②, Fig.2.b). At step 2, the location “leaking” is reached back with  $\{0 \leq \ell \leq 10, t \geq \ell + 50\}$ , (③, Fig. 2.c) the convex hull with  $\{t = \ell = 0\}$ , gives  $\{0 \leq \ell \leq 10, t \geq 6\ell\}$  (④, Fig. 2.c), the “time elapse” provides  $\{0 \leq \ell \leq 20, t \geq \ell, t \geq 6\ell - 50\}$ , and the (standard) widening provides  $\{0 \leq \ell \leq t, t \geq 6\ell - 50\}$  (⑤, Fig. 2.c), which is also the solution for “not leaking” and terminates the iteration with an optimal result: it is the convex hull of the reachable states in each location.

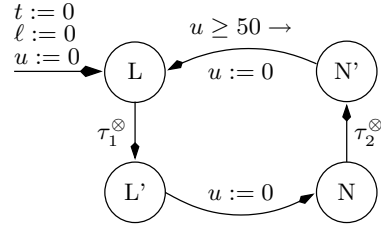
Now, let us consider a discrete version of the gas burner (Fig. 3). First, since there is a loop around each location, we must apply widening in each of them. Now, if we detail the computations, we get that for the L(eaking) location, initially  $t = \ell = 0$ , then  $t = \ell = 1$  (with no contribution back from N(otleaking)), so the convex hull is  $\{0 \leq t = \ell \leq 1\}$ , and widening provides  $\{0 \leq t = \ell\}$ . This is already a less precise result than in the continuous case. Further narrowing does not improve the result. To obtain better results, we should delay widening for at least 10 iterations (because of the constant 10 appearing in the problem). Of course, delaying widening in such a way is expensive; moreover it is rather ad hoc, and it would not work if the constant 10 was replaced by a symbolic parameter, say  $\delta$ .

This example shows that the analysis of hybrid automata can be much more precise and efficient than the analysis of the corresponding discrete counter automata. The obvious reason is the availability of the “time elapse” operator, which plays the role of a specialized exact widening operation. One goal of the paper is to detect that the effect of the single loops in the counter automaton of

<sup>1</sup> which computes the effect of letting the time pass in the location as long as the location invariant is true.



**Fig. 3.** Automaton of the gas burner



**Fig. 4.** “Accelerated” automaton

Fig. 3 can be computed exactly, so that these loops can be *subsumed* by single transitions, exactly as it is done by using the time elapse operator on hybrid automata. In other words, instead of analyzing the automaton of Fig. 3, we will apply the standard analysis to the automaton of Fig. 4, where  $\tau_1^\otimes, \tau_2^\otimes$  denote the operations subsuming the effect of the two single loops in the initial automaton. In this standard analysis, the two single loops will be accelerated, but widening is still applied, e.g., in  $L$ , because of the remaining global loop.

The paper is organized as follows: after making our notations precise (Section 1), we consider first, in Section 2, the trivial case of a single loop where variables are just incremented with constants. Such a loop is called a *translation loop*. We can then formally define, in Section 3, the *abstract acceleration* we want to compute, which is a convex and dense closure of the exact reachable set. Then, in Section 4, we consider the case of several translation loops, and in Section 5, we deal with combinations of translations and assignments of constants. We conclude the paper with comparisons with related work, and some perspectives.

## 1 Definitions and notations

Throughout the paper,  $n$  will denote the number of numerical variables, the numerical states will be considered as elements of the affine space  $\mathbb{Q}^n$ <sup>2</sup>.

Let us recall that a (closed convex) polyhedron in  $\mathbb{Q}^n$  can be seen either as the set  $\{x \in \mathbb{Q}^n \mid Ax \leq B\}$  of solutions of a system of *linear constraints*  $Ax \leq B$  — where  $A$  is an  $m \times n$  matrix, for some  $m \geq 0$ , and  $B$  is an  $m$ -vector — or as the convex hull

$$\left\{ \sum_{v_i \in V} \lambda_i v_i + \sum_{r_j \in R} \mu_j r_j \mid \lambda_i, \mu_j \in \mathbb{Q}^+, \sum \lambda_i = 1 \right\}$$

of a *system of generators* — i.e., a finite set  $V \subset \mathbb{Q}^n$  of *vertices*, and a finite set  $R \subset \mathbb{Q}^n$  of *rays*.

If  $Ax \leq B$  is a system of constraints, we will often note simply  $\{Ax \leq B\}$  the polyhedron of its solutions. If  $P$  is a polyhedron and  $R \subset \mathbb{Q}^n$  is a finite set of vectors, we will note  $P \nearrow R$  the polyhedron  $\{x + \sum_{r_j \in R} \mu_j r_j \mid x \in P, \mu_j \in \mathbb{Q}^+\}$  obtained by adding to  $P$  all vectors in  $R$  as new rays.

<sup>2</sup> We consider  $\mathbb{Q}$  for computational reasons

## 2 A simple case: single translation loops

We first consider the case of single loops, i.e., single paths in the program control-flow graph looping back to a control point. We consider such a single path as a guarded command  $g \rightarrow a$ , where  $g$  is a condition on numerical variables, and  $a$  is a transformation of numerical variables. As usual in LRA, we restrict<sup>3</sup> ourselves to linear conditions ( $g(x) \Leftrightarrow Ax \leq B$ ) and linear transformations (say  $x := Cx + D$ , where  $C$  is an  $n \times n$  matrix,  $D$  is an  $n$ -vector). Let  $\tau$  be the corresponding function:  $\tau(x) = \text{if } Ax \leq B \text{ then } Cx + D \text{ else } x$ . We want to build the corresponding polyhedra transformer, i.e., to be able to compute the image  $P$  of a polyhedron  $P_0$  by an arbitrary number of applications of  $\tau$ :

$$x \in P \Leftrightarrow \exists i \in \mathbb{N}, \exists x_0 \in P_0, x = \tau^i(x_0)$$

i.e., if we define the sequence  $(x_k)$  by  $x_k = C^k x_0 + \sum_{j=0}^{k-1} C^j D$ :

$$x \in P \Leftrightarrow \exists i \in \mathbb{N}, \exists x_0 \in P_0, \forall j \in [0, i-1], Ax_j \leq B, \text{ and } x = x_i$$

In general, obtaining a general expression for  $C^k$  is too expensive, and the quantification over  $i$  and  $j$  cannot be computed. So, let us look at some cases where the computation is possible; in such cases, the loop will be said to be *acceleratable*:

- [Tiw04] considers the same kind of loops, and shows that their termination is decidable. The method uses algebraic characterisation of the  $C$  matrix, but does not provide any loop invariant.
- In [FL02], the linear functions  $\lambda x.Cx + D$  such that the cardinal of  $\{C^k, k \in \mathbb{N}\}$  is finite is pointed out to be a class that is acceleratable. But the upper-bound that is given is too large, and as far as we know, the complexity of the problem of finding whether a monoid generated by a (set of) matrix is finite or not is an open problem (it is known to be decidable [Hal97]).
- The case where  $C^2 = C$  is interesting, since it covers the loops which increment or decrement variables by constants, and/or set variables to constants.
- The simplest case is when  $C = \text{Id}$ , i.e., when all variables are incremented or decremented by constants. We call such loops *translation loops* and we first consider this simple case.

In the case of a translation loop, we get simply  $x_k = x_0 + kD$  and

$$x \in P \Leftrightarrow \exists i \in \mathbb{N}, \exists x_0 \in P_0, \forall j \in [0, i-1], A(x_0 + jD) \leq B, \text{ and } x = x_0 + iD$$

By convexity, the condition  $\forall j \in [0, i-1], A(x_0 + jD) \leq B$  reduces to  $Ax_0 \leq B \wedge A(x_0 - D) \leq B$ . Adding an arbitrary positive number of  $D$  is just adding  $D$  as a ray. Finally, we get:

$$P = \left( (P_0 \cap \{Ax \leq B\}) \nearrow \{D\} \right) \cap \{A(x - D) \leq B\}$$

*Remark 1.* In the last expression, we have lost the points of the initial polyhedron  $P_0$  that don't satisfy  $g$ . In the rest of the paper, without loss of generality, we assume that the initial polyhedron verifies the guard of the transition. If it is not the case, we first compute the intersection with the guard, and after all our computations, we make a convex hull with the initial polyhedron.

<sup>3</sup> Other cases are over-approximated.

*Example:* This allows us to compute the effect of the two simple loops in the gas burner example (Fig. 3). Starting from  $P_L^{(0)} = \{t = \ell = 0\}$ , we first apply  $\tau_1^\otimes$  as in Fig. 4 and get  $P'_L{}^{(0)} = \{0 \leq t = \ell \leq 10\}$ . Then, in location  $N$ , we have  $P_N^{(0)} = \{0 \leq t = \ell \leq 10\}$  which is accelerated into  $P'_N{}^{(0)} = \{0 \leq \ell \leq 10, \ell \leq t\}$ , and the transition back to  $L$  gives  $Q = \{0 \leq \ell \leq 10, \ell + 50 \leq t\}$ , so<sup>4</sup>

$$P_L^{(1)} = P_L^{(0)} \nabla \left( P'_L{}^{(0)} \sqcup Q \right) = \{t = \ell = 0\} \nabla \{0 \leq \ell \leq 10, 6\ell \leq t\} = \{0 \leq 6\ell \leq t\}$$

Applying again  $\tau_1^\otimes$ , we get  $P'_N{}^{(1)} = \{0 \leq \ell \leq t, 6\ell \leq t - 50\}$  which is the correct limit .

### 3 Abstract acceleration

We are now able to make our objective more precise: we want to precisely characterize, when possible, the effect of a loop on a polyhedron. Of course, with respect to the exact effect of the loop, we will have to take a convex hull. Moreover, we are faced with a problem of arithmetic, since the effect of a loop is obtained by applying  $k$  times its body *where  $k$  is an integer*. To avoid the complexity of exact arithmetic, we will perform, as usual, a dense approximation. To summarize, in the case of a simple translation loop, instead of computing the exact effect of the loop:

$$\tau^*(P_0) = \{x \mid \exists i \in \mathbb{N}, \exists x_0 \in P_0, g(x_0) \wedge g(x - D), x = x_0 + iD\} \cup P_0$$

we compute its *abstract acceleration*:

$$\tau^\otimes(P_0) = \bigsqcup \{x \mid \exists i \in \mathbb{Q}^+, \exists x_0 \in P_0, g(x_0) \wedge g(x - D), x = x_0 + iD\} \sqcup P_0$$

We now are able to prove the following proposition :

**Proposition 1.** *Let  $\tau : Ax \leq B \rightarrow x := Cx + D$ . Then*

$$\tau^\otimes(P_0 \cap Ax \leq B) = \left( (P_0 \cap \{Ax \leq B\}) \nearrow \{D\} \right) \cap \{A(x - D) \leq B\}$$

*Sketch of Proof :* Let  $P = \left( (P_0 \cap \{Ax \leq B\}) \nearrow \{D\} \right) \cap \{A(x - D) \leq B\}$ . Then

$$\begin{aligned} x \in P &\Leftrightarrow \left( \exists i \in \mathbb{Q}^+, \exists x_0 \in (P_0 \cap \{Ax \leq B\}), x = x_0 + iD \text{ and } A(x - D) \leq B \right) \\ &\Leftrightarrow x \in \tau^\otimes(P_0) \end{aligned}$$

□

It is also useful to define the *rational* iteration of a translation loop :

**Definition 1.** *Let  $i \in \mathbb{Q}^+$ , then we note :*

$$\tau^i(P_0) = \{x \mid \exists x_0 \in P_0, x = x_0 + iD \wedge g(x_0) \wedge g(x - D)\}$$

<sup>4</sup>  $\sqcup, \nabla$  respectively denote the convex hull and widening operators.

## 4 Two translation loops

In the presence of several translation loops, the situation becomes more complex. For instance, the control graph is not necessarily flat, and exact acceleration techniques no longer apply.

In order to separate the difficulties, we will first assume, at least conceptually, that the control graph is partitioned according to the combination of guards: in a given location, either both guards are satisfied, or only one or the other is satisfied. Once this partitioning is performed, we are left with the problem of accelerating the loops *as long as both guards are satisfied*.

Let us note  $\tau_{1,2}^{\otimes}(P_0)$  the image of an initial polyhedron  $P_0$  by two translation loops  $\tau_i : g_i \rightarrow x := x + D_i$ , ( $i = 1, 2$ ) as long as  $g_1 \wedge g_2$  is satisfied. It is made of all the points  $x$  that can be reached from  $P_0 \cap g_1 \cap g_2$  by successive rational applications of  $\tau_1$  and  $\tau_2$  and staying in  $g_1 \cap g_2$  :

$$\begin{aligned} x \in \tau_{1,2}^{\otimes}(P_0) \text{ iff } & \exists x_0 \in P_0 \cap g_1 \cap g_2, \\ & \exists x_1, x_2, \dots, x_\ell \in g_1 \cap g_2, \exists x'_1, x'_2, \dots, x'_\ell \in g_1 \cap g_2, \\ & \exists i_1, i_2, \dots, i_\ell, i'_1, i'_2, \dots, i'_\ell \in \mathbb{Q}^+, \\ & \text{such that } x = x'_\ell, \text{ and } x_j = \tau_1^{i_j}(x'_{j-1}), x'_j = \tau_2^{i'_j}(x_j), j = 1.. \ell \end{aligned}$$

The following proposition gives a way of computing  $\tau_{1,2}^{\otimes}(P_0)$ :

**Proposition 2.** *Let  $\tau_i$  be  $g_i \rightarrow x := x + D_i$ , ( $i = 1, 2$ ) then,*

- *if  $\exists x_0 \in P_0 \cap g_1 \cap g_2$ ,  $\exists \varepsilon > 0$  such that either  $x_0 + \varepsilon D_1 \in g_1 \cap g_2$  or  $x_0 + \varepsilon D_2 \in g_1 \cap g_2$  (i.e., there is at least one point in  $P_0$  where at least one transition can be “rationally” applied and stay in  $g_1 \cap g_2$ ), then*

$$\tau_{1,2}^{\otimes}(P_0) = ((P_0 \cap g_1 \cap g_2) \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2$$

- *otherwise,  $\tau_{1,2}^{\otimes}(P_0) = P_0 \cap g_1 \cap g_2$*

*Remark 2.* The first condition on  $P_0 \cap g_1 \cap g_2$  comes from the fact that the rational application of  $\tau_1$  or  $\tau_2$  must be initialised. This condition can easily be checked by a simplex method.

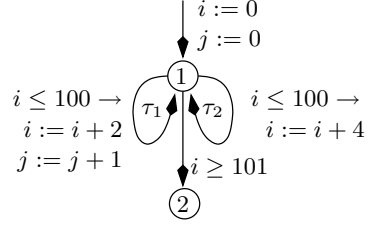
Of course, we don't really partition the control graph, which would involve a combinational explosion in the presence of several loops. But, if we use the combined acceleration computed as in Proposition 2, we compute the (approximate) solution of  $P = P_0 \sqcup \tau_{1,2}^{\otimes}(P) \sqcup \tau_1^{\otimes}(P) \sqcup \tau_2^{\otimes}(P)$  using widening if necessary. It often happens that  $P_0 \sqcup \tau_{1,2}^{\otimes}(P_0) \sqcup \tau_1^{\otimes}(\tau_{1,2}^{\otimes}(P_0)) \sqcup \tau_2^{\otimes}(\tau_{1,2}^{\otimes}(P_0))$  is a (post-) fixpoint, and that widening does not have to be used. Of course, this is one strategy among others, but it gives good experimental results. We could also compute for example the set  $P_0 \sqcup \tau_{1,2}^{\otimes}(P_0) \sqcup \tau_2^{\otimes}(\tau_1^{\otimes}(P_0)) \sqcup \tau_1^{\otimes}(\tau_2^{\otimes}(P_0))$ , or other combinations (like, e.g., in Fast [BFLP03]).

```

i := j := 0;
while (1) i <= 100 do
  if ... then i:=i+2; j:=j+1;
  else i:=i+4;
end
(2)

```

**Fig. 5.** The program



**Fig. 6.** The associated CFG

*Example* As a very simple application, we can now deal with the old basic example of [Hal79] without using any widening:

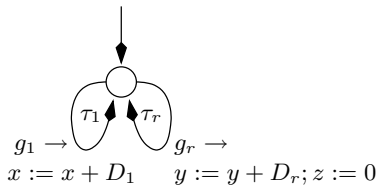
In the “program” of Fig 5, we abstract the “if-then-else” statement by the non-deterministic choice of two simple loops around the control point number 1, getting the control graph of figure 6. Applying our result (the two transition can be applied), we first compute  $\tau_{1,2}^{\otimes}(\{i = j = 0\}) = \{(0, 0)\} \nearrow \{(2, 1), (4, 0)\} \cap \{i \leq 100\} = \{0 \leq 2j \leq i \leq 100\}$ . Then we compute :

- $\tau_1^{\otimes}(\tau_{1,2}^{\otimes}(P_0)) = \{2j \geq 2, 2j \leq i, i \leq 102\}$
- $\tau_2^{\otimes}(\tau_{1,2}^{\otimes}(P_0)) = \{2j \geq 0, 2j < i \leq 104\}$
- The convex hull of the three polyhedra,  $\{0 \leq 2j \leq i, i \leq 104, i + 2j \leq 204\}$ .

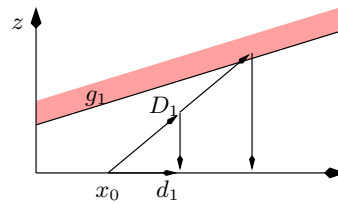
This last set is stable by the application of  $\tau_1$  or  $\tau_2$ , so the convergence is reached and we can propagate the obtained result to location 2, where we get:  $\{0 \leq 2j \leq i, 101 \leq i \leq 104, i + 2j \leq 204\}$ .

## 5 Combining translation and reset

The next case that we will consider is the combination of translation loops with loops where some variables are set to constants. Without loss of generality, we assume that these variables are simply reset to 0. This situation and the corresponding notations are represented in Fig. 7: we assume  $x = (y, z)$ ; in the first loop, all the variables are translated, while in the second one, only the variables  $y$  are translated and the variables  $z$  are set to 0. We will consider simple cases first.



**Fig. 7.** Transition and reset loops



**Fig. 8.** Complete reset

## 5.1 Complete reset

A first simple case is when the second loop performs only resetting, i.e., when  $D_r = 0$ . Let us note  $d_1 = D_1 \downarrow [z = 0]$  the projection of  $D_1$  on the subspace  $z = 0$ . We assume also that  $g_r = true$ . Then, the evolution of variables from a point  $x_0$  can be represented in the plane  $(D_1, d_1)$  as in Fig. 8 (in this figure,  $x_0 \nearrow \{D_1\}$  intersects  $g_1$ , but otherwise, the same expression is still valid). In this case, we obviously have:

**Proposition 3.** *If  $\tau_1 : g_1 \rightarrow x := x + D_1$  and  $\tau_2 : true \rightarrow z := 0$  then:*

$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, d_1\} \cap g_1(x - D_1)$$

*Sketch of Proof :* First let us remark that  $\tau_r^* = \tau_r$  ( $\tau_r$  is only a projection).

- $\subseteq$  : If  $x \in (\tau_1 + \tau_r)^*(x_0)$  (the exact computation), then there has been a succession of  $\tau_1$  and  $\tau_r$ , that can be summarized by the following chain (the  $i_j$  are in  $\mathbb{N}$ ) :

$$\begin{aligned} x_0 \begin{pmatrix} y_0 \\ 0 \end{pmatrix} &\xrightarrow{\tau_1^{i_1}} x_0 + i_1 D_1 \xrightarrow{\tau_r} \begin{pmatrix} y_0 + i_1 d_1 \\ 0 \end{pmatrix} \xrightarrow{\tau_1^{i_2}} x_0 + (i_1 + i_2) D_1 \\ &\xrightarrow{\tau_r} \begin{pmatrix} y_0 + (i_1 + i_2) d_1 \\ 0 \end{pmatrix} \rightarrow \dots \end{aligned}$$

So if the chain ends with a  $\tau_r$ , then there exists  $I_1 \in \mathbb{N}$  such that  $x = x_0 + I_1 d_1$  and  $g_1(x - D_1)$  (it comes from the fact we have  $g_1(x_0 + (I_1 - 1)D_1)$ ). If the chains ends with an iteration of  $\tau_1$ , then  $x = x_0 + I_1 d_1 + I_2 D_1$ , with  $g(x - D_1)$ . As the abstract acceleration is the relaxation of the exact computation, we are done.

- $\supseteq$  : If  $x = x_0 + I_1 d_1 + I_2 D_1$  and  $g_1(x - D_1)$ , then we can obtain the point  $x$  by applying the following “rational” chain :  $x = \tau_1^{I_2} \left( (\tau_r(\tau_1(x_0)))^{I_1} \right)$ . Indeed, any application of  $\tau_1$  followed by an application of  $\tau_r$  from an initial point  $x_0 = (y_0, 0)$  leads to the point  $(y_0 + d_1, 0)$ , which allows us to define the rational alternation of  $\tau_r$  and  $\tau_1$  as if it were an application of a single translation of vector  $d_1$ . So applying  $I_1$  times (possibly 0) this alternation, we obtain the point  $(y_0 + I_1 d_1, 0)$ . Then we end by applying  $\tau_2^{I_2}$ . □

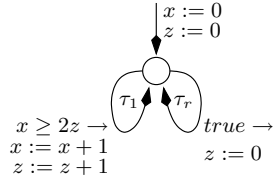
This simple case generalizes naturally:

- for  $D_r \neq 0$ , if  $D_r$  belongs to the plane  $(D_1, d_1)$
- for  $g_r \neq true$  if  $\{x_0\} \nearrow \{D_1\} \cap g_1$  intersects  $g_r$ .

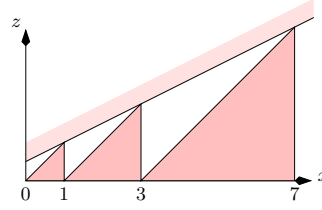
*Remark 3.* Notice that we can easily produce reachable domains that cannot be described by Presburger formulas, as shown by the figures 9 and 10 (where the exact set is  $\exists k \geq 0, (2^{k-1} - 1 \leq x \leq 2^k - 1 \wedge z \geq 0 \wedge x \geq 2z - 1)$ ), which means that standard exact acceleration techniques cannot work. In this case, the previous proposition leads to the abstract acceleration :  $\{x \geq z \geq 0, x \geq 2z - 1\}$ .

## 5.2 Partial reset

Now, we consider the case when  $D_r \neq 0$  does not belong to the plane  $(D_1, d_1)$  (i.e., there are variables which are incremented in the second loop, while being unchanged by the first loop). As before, we assume that  $g_r = true$ , but we also

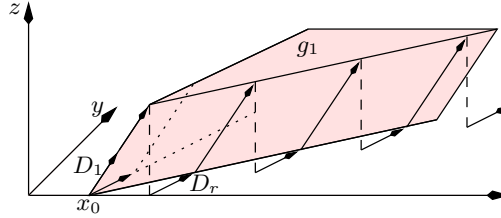


**Fig. 9.** The CFG



**Fig. 10.** The corresponding behaviour

assume that  $g_1$  is of the form  $z \leq K$ , i.e., is parallel to the hyperplane  $z = 0$  (see Remark 5 about this restriction). Moreover, we consider an initial polyhedron  $P_0$  included in the hyperplane  $z = 0$ . Now, the variables can evolve from a point  $x_0$  as shown on Fig. 11.



**Fig. 11.** Partial reset

From  $x_0$ ,  $\tau_1$  can be applied at most, say,  $k_{max}$  times, where  $k_{max}$  is the minimum over all reset variables  $z_i$  of the expressions  $\lfloor K/D_{1z_i} + 1 \rfloor$ , which we denote  $k_{max} = \lfloor K/D_{1z} + 1 \rfloor$ <sup>5</sup>. At any time meanwhile,  $\tau_r$  can occur, resetting  $z$  and translating the result according to  $D_r$  in the plane  $\{z = 0\}$ . So, after some applications of  $\tau_1$  followed by one application of  $\tau_r$ , we have  $x = x_0 + kd_1 + D_r$ , where  $d_1 = D_1 \downarrow [z = 0]$  as before, and  $0 \leq k \leq k_{max}$ . Then from any such  $x$ , the same transformation can occur. One can easily show that the resulting domain is given by the following proposition:

**Proposition 4.** Let  $\tau_1$  be of the form  $(z \leq K) \rightarrow x := x + D_1$  and  $\tau_r$  be  $true \rightarrow y := y + D_{ry}; z := 0$ . Let  $P_0 \subset \{z = 0\}$ . Let  $u_z = (0, \dots, 0, 1, \dots, 1)$  where we have 1s for the  $z$  components, and 0 elsewhere. Let  $d_1 = D_1 \downarrow [z = 0]$  and  $D_r = \begin{pmatrix} D_{ry} \\ 0 \end{pmatrix}$ . Then:

- if  $D_1 \cdot u_z < 0$  then  $(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, d_1, D_r\}$
- else, let  $k_{max} = \lfloor K/D_{1z} + 1 \rfloor$ , we have:

$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r, k_{max}d_1 + D_r\} \cap g_1(x - D_1)$$

<sup>5</sup> Notice that, here, we precisely take arithmetic into account, since it can be done at reasonable cost.

*Remark 4.* The scalar product  $D_1 \cdot u_z$  characterizes whether or not  $P_0 \nearrow \{D_1\}$  intersects  $g_1$ .

*Sketch of Proof :* Without loss of generality, we can assume that each time we apply  $\tau_r^i$  or  $\tau_1^i$ , we have  $i > 0$ .

- In the first case,  $P_0 \nearrow \{D_1\}$  does not intersect  $g_1$ . It means that the guard  $g_1$  is always satisfied :

$$x_0 \xrightarrow{\tau_1^{i_1}} \xrightarrow{\tau_r^{i'_1}} x_0 + i_1 d_1 + i'_1 D_r \xrightarrow{\tau_1^{i_2}} \xrightarrow{\tau_r^{i'_2}} x_0 + (i_1 + i_2) d_1 + (i'_1 + i'_2) D_r \rightarrow \dots$$

Then if the chain ends with some  $\tau_r$ ,  $x = x_0 + I d_1 + I' D_r$  (in particular,  $z = 0$ ). If the chain ends with some  $\tau_1$ , we obtain  $x = x_0 + I d_1 + I' D_r + I'' D_1$ , with no bound for  $I, I', I''$ .

- In the second case, the number of iteration of  $\tau_1$  following one (or more) application(s) of  $\tau_r$  is at most  $k_{max}$ . We write a similar chain, except that we have the property  $\forall j, 0 \leq i_j \leq k_{max}$ . If the chains ends with  $\tau_r^+$ , we obtain  $x = x_0 + (i_1 + i_2 + \dots + i_n) d_1 + (i'_1 + \dots + i'_n) D_r$ . Let  $I = i_1 + i_2 + \dots + i_n$ . Taking the Euclidean division of  $I$  by  $k_{max}$ , we get  $I = q k_{max} + r$  with  $r \leq k_{max}$  and  $q \leq n$ . Then  $x = x_0 + (q \cdot k_{max} + r_1) d_1 + (q + r_2) D_r$  with  $r_1, r_2 \geq 0$ , and finally  $x = x_0 + q(k_{max} d_1 + D_r) + r_1 d_1 + r_2 D_r$ , which is the good form.  $x$  also satisfied  $g_1(x - D_1)$  because all the plane  $\{z = 0\}$  satisfy  $g_1$ , and  $-D_1$  moves away  $x$  from the guard  $g_1$ .

If the chains ends with  $\tau_1^*$ , we add some  $i_{n+1} D_1$ , but we must satisfy  $g_1$  before the last but one application, hence  $g_1(x - D_1)$ .

These arguments justify the left-to-the right inclusions. The proof of the two other inclusions are very similar to the proof of Proposition 4.  $\square$

*Example:* we consider a very simple reactive program [HPR97], supposed to model a speedometer under the assumption that the speed is less than 4 meters/second: the speedometer perceives either an elapsed second from some clock, in which case the time  $t$  is incremented while the instantaneous speed  $s$  (which counts the number of meters occurring during each second) is reset to 0, or a “meter” sensor, in which case both the distance  $d$  and the instantaneous speed  $s$  are incremented; this “meter” event can only occur when  $s \leq 3$ , because of the assumption on the speed.

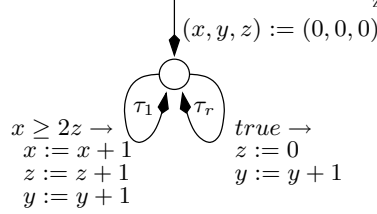
```
t := d := s := 0 ;
while true do
1: if second then
    t := t+1 ; s:= 0 ;
    else if meter and s<=3 then
        d := d+1 ; s := s+1 ;
    end
```

With the notations of Fig. 7, we have  $x = (t, d, s), y = (t, d), z = (s), u_z = (0, 0, 1), g_1 = (s \leq 3), D_1 = (0, 1, 1), D_r = (1, 0, 0)$ , and  $x_0 = (0, 0, 0)$ . We have  $u_z \cdot D_1 = 1 \geq 0$  hence we compute  $k_{max} = 4$  and  $d_1 = (0, 1, 0)$ , so  $k_{max} d_1 + D_r = (1, 4, 0)$ , and finally:

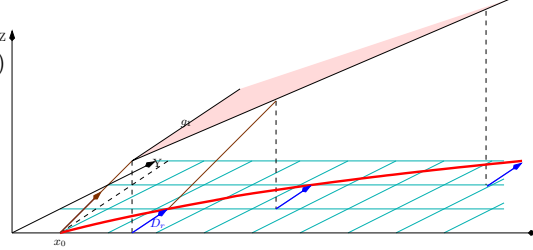
$$\begin{aligned} (\tau_1 + \tau_r)^\otimes(x_0) &= (0, 0, 0) \nearrow \{(0, 1, 1), (1, 0, 0), (1, 4, 0)\} \cap \{s \leq 4\} \\ &= \{t \geq 0, 0 \leq s \leq 4, 0 \leq d \leq 4t + s\} \end{aligned}$$

so we get at once the precise result, which is not easy to obtain with widening (in [HPR97], we needed 3 iterations, and a “limited widening”).

*Remark 5.* If the guard is not of the form  $z \leq K$ , the behaviour can be non linear. In the following example, one border of the reachable domain is a parabola:



**Fig. 12.** The CFG



**Fig. 13.** The corresponding behaviour

### 5.3 Weakening the assumptions

The previously considered case may appear quite specific. However, we can easily suppress or alleviate some of our assumptions:

- If  $P_0$  is not included in  $\{z = 0\}$ , first compute  $P'_0 = \tau_r(\tau_1^\otimes(P_0))$ , which is included in  $\{z = 0\}$  (since it results from an application of  $\tau_r$ ).
- We can extend Proposition 4 in order to take into account a guard of the form  $z \bowtie K_r$  ( $\bowtie \in \{\leq, =, \geq\}$ ) for the second loop, using Proposition 5 below.
- Finally, Section 5.4 will give an example of using these results to combine a reset loop with more than one translation loop.

**Proposition 5.** Let  $\tau_1, \tau_r$  be respectively of the form:

$$\tau_1 : (z \leq K_1) \rightarrow x := x + D_1 \text{ and } \tau_r : (z \bowtie K_r) \rightarrow y := y + D_r; z := 0$$

where  $\bowtie \in \{\leq, =, \geq\}$ . Assume  $P_0 \subset \{z = 0\}$  and  $K_1 > 0$  and  $D_1 \cdot u_z > 0$ . Let us note  $k_{max1} = \lfloor K_1/D_1z + 1 \rfloor$ ,  $d_1 = D_1 \downarrow [z = 0]$ , and  $k_{maxr} = \lfloor K_r/D_1z + 1 \rfloor$ . Then:

- if  $\bowtie$  is “ $\leq$ ” then
  - if  $K_r < 0$  then ( $\tau_r$  never applies)
$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1\} \cap g_1(x - D_1)$$
  - if  $K_r > K_1$  then
$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r, D_r + k_{max1}d_1\} \cap g_1(x - D_1)$$
  - if  $K_1 \geq K_r > 0$  then
$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r, D_r + k_{maxr}d_1\} \cap g_1(x - D_1)$$
- if  $\bowtie$  is “ $=$ ” then
  - if  $K_1 \geq K_r > 0$  and  $\exists k, K_r = kD_1z$  then
$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r + kd_1\} \cap g_1(x - D_1)$$
  - else ( $\tau_r$  never applies)
$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1\} \cap g_1(x - D_1)$$

– if  $\bowtie$  is “ $\geq$ ” then

- if  $K_r > K_1$  and  $K_r > 0$  then ( $\tau_r$  never applies)

$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1\} \cap g_1(x - D_1)$$

- if  $K_1 \geq K_r \geq 0$ , then

$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r + k_{max1}d_1, D_r + k_{maxr}d_1\} \cap g_1(x - D_1)$$

- if  $K_r < 0$  then

$$(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, D_r, D_r + k_{max1}d_1\} \cap g_1(x - D_1)$$

*Remark 6.* If  $D_1 \cdot u_z < 0$  under the same assumptions as in Proposition 5,  $g_1$  is always true, and we easily get:  $(\tau_1 + \tau_r)^\otimes(P_0) = P_0 \nearrow \{D_1, d_1, D_r\}$

*Sketch of Proof :* The demonstration is mostly like the preceding one, except that we take the second guard into account. In particular, if the second guard is of the form  $z = K_2$ , then we must check whether or not the *real* set  $\{x + iD_1, i \in \mathbb{N}\}$  intersects  $g_2$ .  $\square$

*Remark 7.* Because  $g_1 = x \leq K_1$  and  $P_0 \subset \{z = 0\}$ , the successive images of any point  $x_0 \in P_0$  are  $\{x_0 + kD_1 \mid 0 \leq k \leq k_{max}\}$  with  $k_{max}$  independent of  $x_0$ . Let us consider the ray  $D_r + k_{max}d_1$  of Proposition 4. If we had an algorithm to compute directly  $\sqcup\tau_1^*(x_0)$  (the polyhedron representing the convex hull of the exact computation of all  $x_0 + kD_1$ , with  $0 \leq k \leq k_{max1}$ ), the Proposition 4 ensures that we can use the following algorithm to compute  $D_r + k_{max}d_1$  (in this case, the two algorithms are equivalent):

1. Select one point  $x_0 \in P_0$ .
2. Compute the segment  $S = [x_0, x_0^M] = \sqcup\tau_1^*(x_0)$  (exact computation).
3. Compute  $D_r + k_{max}d_1 = \tau_r(x_0^M) - x_0$ .

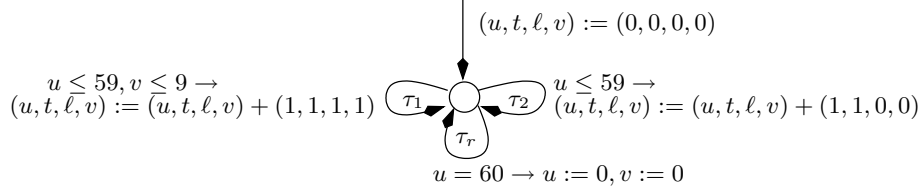
Now, in Proposition 5, if we want to compute the rays  $D_r + k_{max1}d_1$  and  $D_r + k_{maxr}d_1$  (when necessary), we must obtain (if they exist) the “real” points of the set  $S \cap g_r$  (i.e., the points that are reachable with  $\tau_1$  in  $i$  steps where  $i \in \mathbb{N}$ ). Notice that we get an algorithm that does not care about the relative values of  $K_1$  and  $K_r$ . Notice also that the extremal points of  $S \cap g_r$  can sometimes be computed directly if all the  $D_{1z_i}$  belong to  $\{-1, 0, 1\}$ , because the successive images of  $x_0$  are points with integer coordinates. We will apply this remark in the example below.

#### 5.4 An example with more translation loops and resets

We saw in the previous paragraph that the key property of  $\tau_1$  (the unique translation loop) is that the number of its iterations when  $z = 0$  is bounded by a constant  $k_{max1}$ . Let us now consider the case of two translation loops  $\tau_1$  and  $\tau_2$  combined with a reset loop  $\tau_r : z \leq K_r \rightarrow y := y + D_r, z := 0$ .

If we had a similar bound property for  $(\tau_1 + \tau_2)$ , we could have a similar expression for  $(\tau_1 + \tau_2 + \tau_r)^\otimes$ . It is the case if both of the two guards  $g_1$  and  $g_2$  only constrain variables in  $z$ . This condition guarantees that all points in  $\{z = 0\}$  have “parallel futures”.

*Yet another gas burner example:* We consider a modified version of the gas burner example, where it is only assumed that, in each consecutive 60-second interval, the cumulated leaking time is at most 10 seconds. A new variable  $v$  must be introduced to count the cumulated leaking time since the last time  $u$  has been reset to zero (see Fig 14). Now, we adapt the algorithm of the previous section :



**Fig. 14.** A more complex version of the gas burner

- **Step 1** At the beginning, the polyhedron associated with the control point is  $P^{(0)} = \{u = t = \ell = v = 0\}$ . We first compute  $(\tau_1 + \tau_2)^\otimes(P^{(0)})$ , applying the strategy given in Section 4 :  $\tau_{12}^\otimes(P^{(0)}) = (0, 0, 0, 0) \nearrow \{(1, 1, 1, 1); (1, 1, 0, 0)\} \cap \{u \leq 59; v \leq 9\} = \{0 \leq \ell = v \leq 10; \ell \leq u = t \leq 59\}$ ;  $\tau_1^\otimes(\tau_{12}^\otimes(P^{(0)})) = \{\ell \leq t, u = t \leq 60, 0 \leq v = \ell \leq 10\}$ ;  $\tau_2^\otimes(\tau_{12}^\otimes(P^{(0)})) = \tau_1^\otimes(\tau_{12}^\otimes(P^{(0)}))$ , so finally we get the exact set  $(\tau_1 + \tau_2)^\otimes(P^{(0)}) = \{\ell \leq t, u = t \leq 60, 0 \leq v = \ell \leq 10\}$ .
- Then, we intersect this last set with  $g_r = \{u = 60\}$ <sup>6</sup>, and we get the extremal points  $(60, 60, 10, 10)$  and  $(60, 60, 0, 0)$ , thus  $r_1 = \tau_r(60, 60, 10, 10) - (0, 0, 0, 0) = (0, 60, 10, 0)$  and  $r_2 = (0, 60, 0, 0)$ . Then,  $P^{(1)} = (\tau_1 + \tau_2)^\otimes(P^{(0)}) \nearrow \{(0, 60, 10, 0), (0, 60, 0, 0)\} \cap \{u \leq 60; v \leq 10\} = \{v \leq \ell, u \leq 60, 0 \leq v \leq 10, v \leq u, u + 6\ell \leq t + 6v\}$ .
- **Step 2** We compute  $P^{(2)} = (\tau_1 + \tau_2)^\otimes(P^{(1)})$  with the same method, replacing  $P^{(0)}$  by  $P^{(1)}$ . We quickly remark that  $(\tau_1 + \tau_2)^\otimes(P^{(2)}) \subseteq P^{(2)}$ ,  $\tau_1^\otimes(P^{(2)}) \subseteq P^{(2)}$ ,  $\tau_2^\otimes(P^{(2)}) \subseteq P_2$ , hence we get the invariant:<sup>7</sup>  $\{v \leq \ell, u \leq 60, 0 \leq v \leq 10, v \leq u, u + 6\ell \leq t + 6v\}$ , whose projection on  $\{t, \ell\}$  gives  $\{0 \leq \ell \leq t, 6\ell \leq t + 50\}$ .

## 6 Related work and conclusion

This work is a new attempt at decreasing the imprecision due to the widening in Linear Relation Analysis. The initial widening operator of [CH78] was promptly improved in [Hal79], which proposed the operator often called “standard widening”. More recently, [BHRZ03] proposed several ways of improving the standard widening, in the sense that the result of *a single application* of the new operators

<sup>6</sup> Here it is not necessary to bother about arithmetic, because our actions are just incrementations, see Remark 7.

<sup>7</sup> We are sure that we have the exact one because we have no loss of precision due to arithmetic

is guaranteed to be smaller than the one computed with the standard widening. Although these new operators seem to be really better in practice — in the sense that, in many cases they provide more precise limits without significant loss of performance —, there are counterexamples (like the speedometer of Section 5.2) showing that it is not always the case. [GR06] is a nice attempt to improve the precision by carefully alternating increasing widened sequences and descending (possibly narrowed) sequences. The approach has more general applications than Linear Relation Analysis, but could be combined with ours for LRA.

Instead of improving widening, we tried to complement it with some kind of acceleration, whenever possible. The essential difference between acceleration and widening is that widening is only based on the successive results of the abstract semantics of the program (i.e.,  $x_n$  and  $f(x_n)$  are used to compute  $x_{n+1} = x_n \nabla f(x_n)$ ), while acceleration looks at the function  $f$  itself to build  $f^*$ . Among the techniques that take the abstract function into account, one can mention the “widening with thresholds” [BCC<sup>+</sup>03] or “widening upto” [HPR97], where the conditions involving a loop exit are used to limit the extrapolation.

Of course, we were strongly inspired by exact acceleration techniques [BW94, WB98, CJ98, FS00, BFLP03]. However, we don’t want to pay the price of exact computations, and we want to obtain general analysis techniques. So, we use only an *abstract* acceleration, keeping the polyhedral approximation, and we still combine it with usual widening, to preserve the generality of the method. In [SW04], a class of programs is identified for which the abstract solution of the abstract semantics *in the lattice of intervals* can be computed exactly, without widening. Our goals are similar, but in the richer and more complex lattice of polyhedra. The closest approach to ours is probably the one applied in PIPS [IJT91, Iri05]. First, in this work, the abstract function is naturally taken into account, because PIPS applies a modular relational analysis — i.e., it computes the relation between initial and final values of variables of a program fragment. Then, a kind of abstract acceleration is applied, based on discrete differentiation and integration. However, this technique is not combined with widening, which is not used in PIPS.

When our abstract acceleration applies alone, it is guaranteed to provide better results than widening — in fact, it provides the best possible results in term of polyhedra. Used in combination with widening, it generally improves the precision of the analysis — we don’t have any counterexample so far — because it precisely foresees some future behaviors. In spite of apparently strong hypotheses, the abstract acceleration applies quite often in programs with counters, and our first experiments show significant improvements, both in precision and in performance.

A very first implementation of our technique is available. The detection of accelerable loops is not very elaborate, for the time being: the strongly connected subcomponents (SCSC) of the control graph are identified using Bourdoncle’s classical extension [Bou92] of Tarjan algorithm [Tar72]. Then the SCSC are considered bottom-up (starting from the deepest ones): in each of them the paths looping around the input node are checked w.r.t. our acceleration criteria,

and possibly replaced by meta-transitions [Boi99]. Of course, since the number of such paths can be large, we are not obliged to consider all of them.

Future work include of course further experiments of the proposed techniques, which are also likely to be extended towards more general cases. In particular, loops which may exchange values between variables could be considered.

## References

- [ACH<sup>+</sup>95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 138:3–34, January 1995.
- [BBC<sup>+</sup>00] N. Bjorner, A. Browne, M. Colon, B. Finkbeiner, Z. Manna, H. Sipma, and T. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16:227–270, 2000.
- [BBM97] N. Bjorner, I. Anca Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997.
- [BCC<sup>+</sup>03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI 2003, ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation*, pages 196–207, San Diego (Ca.), June 2003.
- [BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast acceleration of symbolic transition systems. In *CAV'03*, pages 118–121, Boulder (Colorado), July 2003. LNCS 2725, Springer-Verlag.
- [BHRZ03] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin.
- [Boi99] B. Boigelot. Symbolic methods for exploring infinite state spaces. Phd thesis, Université de Liège, 1999.
- [Bou92] F. Bourdoncle. Sémantique des langages impératifs d'ordre supérieur et interprétation abstraite. Thesis Ecole Polytechnique, Paris, 1992.
- [BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*, Stanford (Ca.), 1994. LNCS 818, Springer Verlag.
- [CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fix-points. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.
- [CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.
- [CHR91] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5), 1991.
- [CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, Vancouver (B.C.), 1998. LNCS 1427, Springer Verlag.

- [DRS01] N. Dor, M. Rodeh, and M. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In P. Cousot, editor, *SAS'01*, Paris, July 2001. LNCS 2126.
- [FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proceedings of the 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FSTTCS'2002)*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156, Kanpur, India, December 2002. Springer.
- [FS00] A. Finkel and G. Sutre. An algorithm constructing the semilinear post\* for 2-dim reset/transfer vass. In *25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000)*, Bratislava, Slovakia, August 2000. LNCS 1893, Springer Verlag.
- [GR06] D. Gopan and T. Reps. Lookahead widening. To appear in *Computer-Aided Verification*, 2006.
- [Hal79] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de troisième cycle, University of Grenoble, March 1979.
- [Hal97] Vesa Halava. Decidable and undecidable problems in matrix theory. Technical Report TUCS-TR-127, University of Turku, 30, 1997.
- [HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
- [HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.
- [IJT91] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization: An overview of the PIPS project. In *ACM Int. Conf. on Supercomputing, ICS'91, Köln*, 1991.
- [Iri05] F. Irigoin. Detecting affine loop invariants using modular static analysis. Technical Report A/367/CRI, Centre de Recherche en Informatique, Ecole des Mines de Paris, July 2005.
- [SSM04] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constraint-based linear relations analysis. In *International Symposium on Static Analysis, SAS'2004*, pages 53–68. LNCS 3148, Springer Verlag, 2004.
- [SW04] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In *TACAS'04*, pages 280–295, Barcelona, 2004.
- [Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
- [Tiw04] A. Tiwari. Termination of linear programs. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV*, volume 3114 of *LNCS*, pages 70–82. Springer, July 2004.
- [WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*, pages 88–97, Vancouver, June 1998. LNCS 1427, Springer-Verlag.