

Combining Widening and Acceleration in Linear Relation Analysis

Laure Gonnord, Nicolas Halbwachs

VERIMAG/CNRS
Grenoble, France



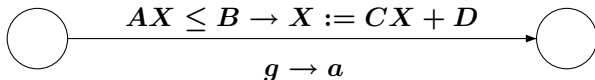
- 1 Introduction and Motivation
- 2 Simple loops
- 3 Two translation loops
- 4 Translations and reset loops
- 5 Implementation - ASPIC

Introduction

Context : Linear Relation Analysis (LRA) [Cousot&Halbwachs 78]

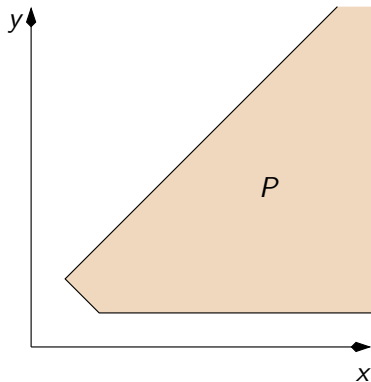
The analysis framework

- CFG with numerical variables and **affine** guards and actions :



- Forward and/or backward analysis to compute invariants.
- Polyhedral (over-) approximations of invariants sets. (Convex polyhedra in $\mathcal{P}(\mathbb{Q}^n)$ in **double description**).
- Possibly infinite sequences \Rightarrow **widening operator**.

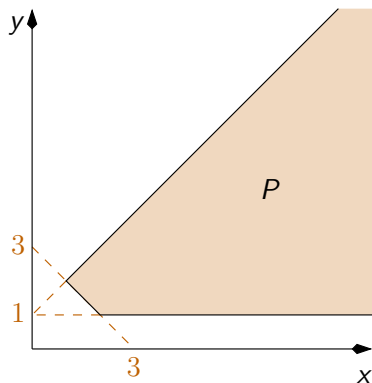
The double description



Adding rays operator

$$P \nearrow R = \left\{ x + \sum_{r_j \in R} \mu_j r_j \mid x \in P, \mu_j \in \mathbb{Q}^+ \right\}$$

The double description

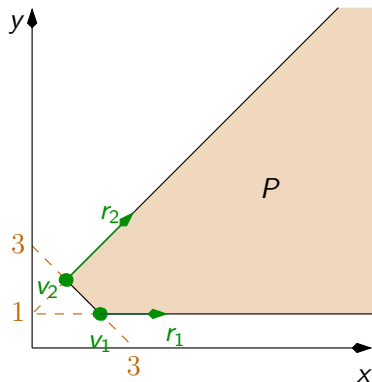


$$\begin{aligned}
 P &= \{(x, y) \mid \\
 &\quad 1 \leq y \leq x + 1 \wedge x + y \geq 3\} \\
 &= \text{cons}\{AX \leq b\}
 \end{aligned}$$

Adding rays operator

$$P \nearrow R = \{x + \sum_{r_j \in R} \mu_j r_j \mid x \in P, \mu_j \in \mathbb{Q}^+\}$$

The double description



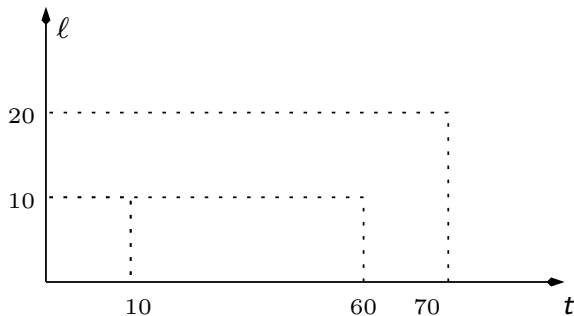
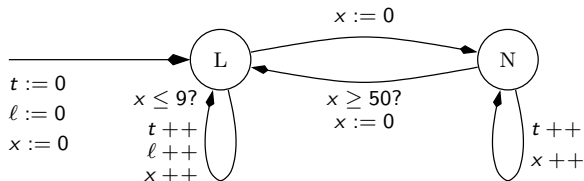
$$\begin{aligned}
 P &= \{(x, y) \mid \\
 &\quad 1 \leq y \leq x + 1 \wedge x + y \geq 3\} \\
 &= \text{cons}\{AX \leq b\}
 \end{aligned}$$

$$\begin{aligned}
 P &= \{\lambda v_1 + (1 - \lambda)v_2 + \mu_1 r_1 + \mu_2 r_2 \mid \\
 &\quad \lambda \in [0, 1], \mu_1, \mu_2 \geq 0\} \\
 &= \text{gen}(V, R)
 \end{aligned}$$

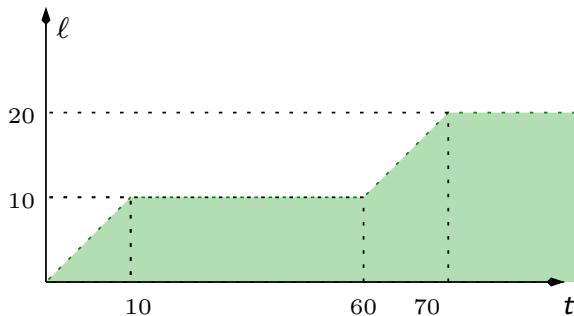
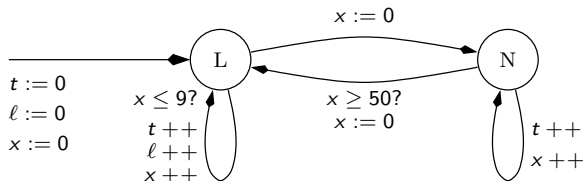
Adding rays operator

$$P \nearrow R = \{x + \sum_{r_j \in R} \mu_j r_j \mid x \in P, \mu_j \in \mathbb{Q}^+\}$$

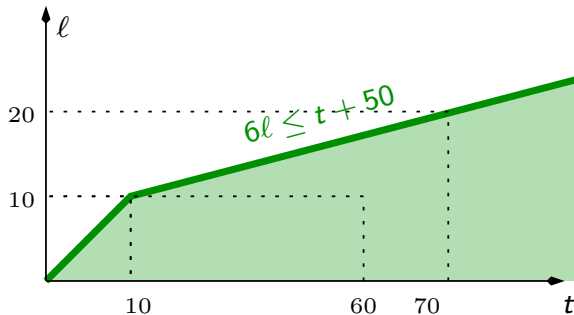
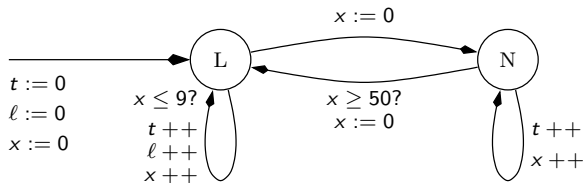
Example - Gaz Burner - 1



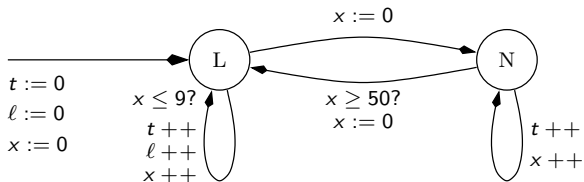
Example - Gaz Burner - 1



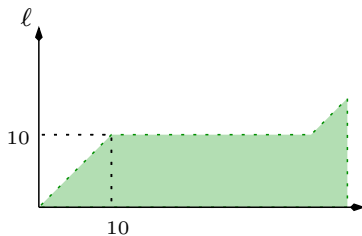
Example - Gaz Burner - 1



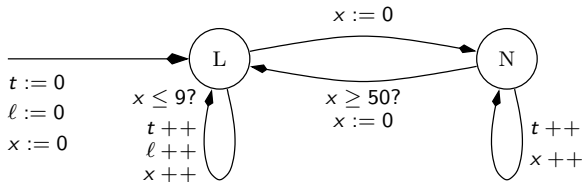
Example - Gaz Burner - 2



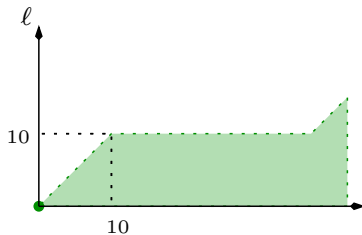
Standard LRA algorithm



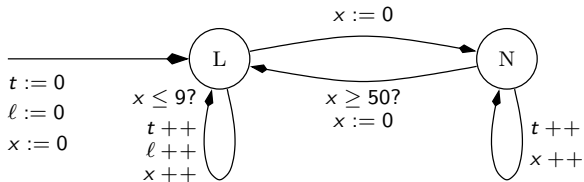
Example - Gaz Burner - 2



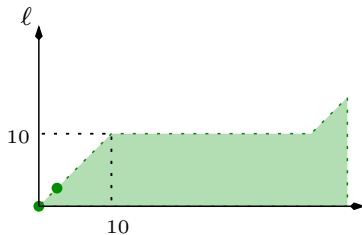
Standard LRA algorithm



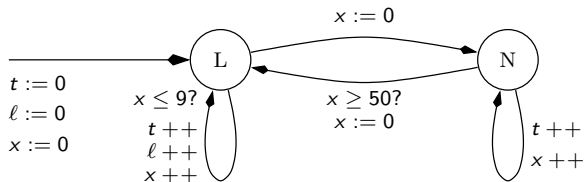
Example - Gaz Burner - 2



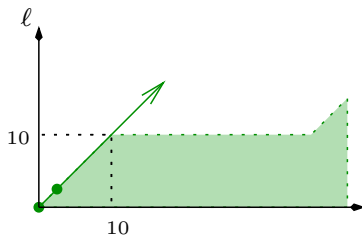
Standard LRA algorithm



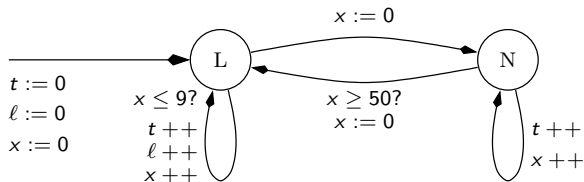
Example - Gaz Burner - 2



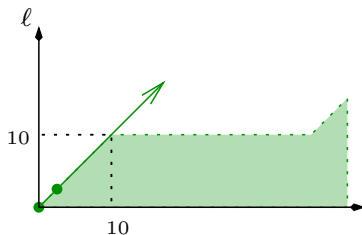
Standard LRA algorithm



Example - Gaz Burner - 2

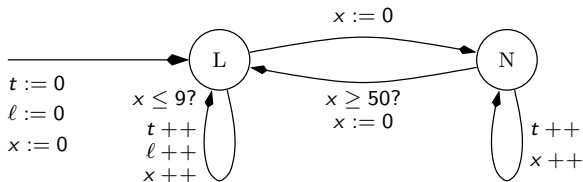


Standard LRA algorithm

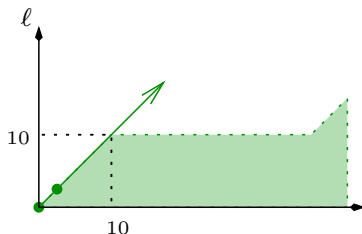


► **lack of precision** : descending sequence will not recover

Example - Gaz Burner - 2



Standard LRA algorithm



- ▶ **lack of precision** : descending sequence will not recover
- ▶ **delaying ?** 60 times !

Existing techniques (1)

Linear Relation Analysis (LRA) and Widening

- Approximate results
- Possible improvements :
 - **Delaying** the application of the widening, drawback : the cost.
 - **Changing** the operator. [Bagnara&Hill&Zafanella], drawback : no result on the global convergence.
 - Alternating widening and narrowing [Gopan&Reps : CAV 2006], drawback : the cost.

Existing techniques (2)

Acceleration

[Boigelot&Wolper, Common&Jurski, Finkel&Sutre&Leroux]

- Computing the **exact effect** of loops on integer sets.
- Encoding : Automata representing Presburger Formula
- **Drawbacks** : restricted class of programs, semi-algorithms, high complexity

▶ Our global objective : “abstract” acceleration **at low cost** for convex polyhedra combined with widening.

▶ More precisely : compute the convex hull of the exact reachable states.

Existing techniques (2)

Acceleration

[Boigelot&Wolper, Common&Jurski, Finkel&Sutre&Leroux]

- Computing the **exact effect** of loops on integer sets.
- Encoding : Automata representing Presburger Formula
- **Drawbacks** : restricted class of programs, semi-algorithms, high complexity

▶ **Our global objective** : “abstract” acceleration **at low cost** for convex polyhedra combined with widening.

▶ **More precisely** : compute the convex hull of the exact reachable states.

Existing techniques (2)

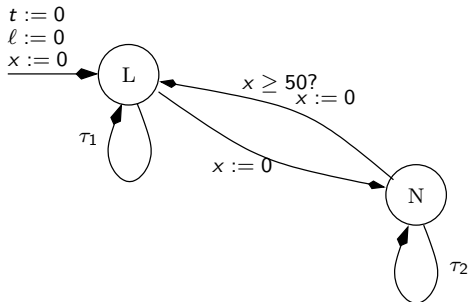
Acceleration

[Boigelot&Wolper, Common&Jurski, Finkel&Sutre&Leroux]

- Computing the **exact effect** of loops on integer sets.
 - Encoding : Automata representing Presburger Formula
 - **Drawbacks** : restricted class of programs, semi-algorithms, high complexity
- ▶ **Our global objective** : “abstract” acceleration **at low cost** for convex polyhedra combined with widening.
- ▶ **More precisely** : compute the convex hull of the exact reachable states.

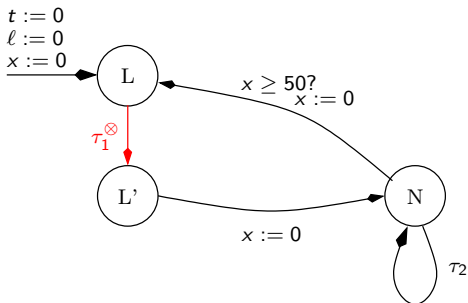
Acceleration within LRA

We want to replace the loops ($\tau_i : g_i \rightarrow a_i$)



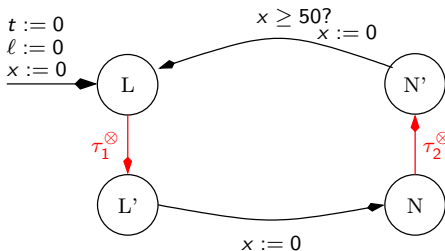
Acceleration within LRA

We want to replace the loops ($\tau_i : g_i \rightarrow a_i$)



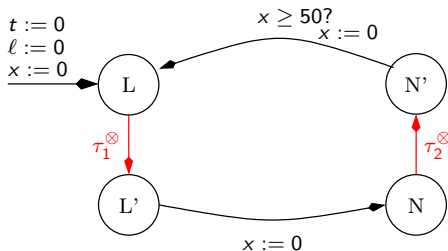
Acceleration within LRA

We want to replace the loops ($\tau_i : g_i \rightarrow a_i$)



Acceleration within LRA

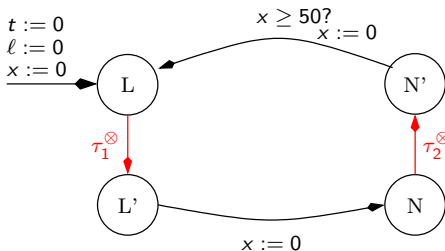
We want to replace the loops ($\tau_i : g_i \rightarrow a_i$)



► τ_i^\otimes summarizes the effect of **any** number of applications of τ_i

Acceleration within LRA

We want to replace the loops ($\tau_i : g_i \rightarrow a_i$)



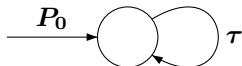
- ▶ τ_i^{\otimes} summarizes the effect of **any** number of applications of τ_i
- ▶ global loop : **accelerate** or **widen**

- 1 Introduction and Motivation
- 2 Simple loops
- 3 Two translation loops
- 4 Translations and reset loops
- 5 Implementation - ASPIC

Simple loops

We want to **characterise** $\tau^*(P_0) = \bigcup_{i \in \mathbb{N}} \tau^i(P_0)$, with :

$\tau(x) = \text{if } Ax \leq B \text{ then } Cx + D \text{ else } x$



Previous result : [Boigelot99, Leroux02] : if $\exists p, C^{2p} = C^p$ and φ a Presburger set, then $\tau^*(\varphi)$ is a computable Presburger set.

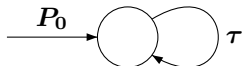
New result (not in the SAS paper) If $\exists p, C^{2p} = C^p$, then computing $\tau^*(P_0)$ reduces itself to computing $\tau'^*(P'_0)$:

- τ' a translation/reset transition.
- P'_0 a union of polyhedra.
- Polynomial Reduction.

Simple loops

We want to **characterise** $\tau^*(P_0) = \bigcup_{i \in \mathbb{N}} \tau^i(P_0)$, with :

$\tau(x) = \text{if } Ax \leq B \text{ then } Cx + D \text{ else } x$



Previous result : [Boigelot99, Leroux02] : if $\exists p, C^{2p} = C^p$ and φ a Presburger set, then $\tau^*(\varphi)$ is a computable Presburger set.

New result (not in the SAS paper) If $\exists p, C^{2p} = C^p$, then computing $\tau^*(P_0)$ reduces itself to computing $\tau'^*(P'_0)$:

- τ' a translation/reset transition.
- P'_0 a union of polyhedra.
- Polynomial Reduction.

Simple loops

We want to **characterise** $\tau^*(P_0) = \bigcup_{i \in \mathbb{N}} \tau^i(P_0)$, with :

$\tau(x) = \text{if } Ax \leq B \text{ then } Cx + D \text{ else } x$ 

Previous result : [Boigelot99, Leroux02] : if $\exists p, C^{2p} = C^p$ and φ a Presburger set, then $\tau^*(\varphi)$ is a computable Presburger set.

New result (not in the SAS paper) If $\exists p, C^{2p} = C^p$, then computing $\tau^*(P_0)$ reduces itself to computing $\tau'^*(P'_0)$:

- τ' a translation/reset transition.
- P'_0 a union of polyhedra.
- Polynomial Reduction.

Single translation loop

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

- (obvious) Proposition

$$\tau^*(P_0) = \{x \mid \exists i \in \mathbb{N}, \exists x_0 \in P_0, \\ Ax_0 \leq B, A(x - D) \leq B, x = x_0 + iD\} \cup P_0$$

Not convex

- Discrete vs Dense acceleration

$$\tau^{\otimes}(P_0) = \{x \mid \exists i \in \mathbb{Q}^+, \exists x_0 \in P_0, \\ Ax_0 \leq B, A(x - D) \leq B, x = x_0 + iD\} \sqcup P_0$$

Convex polyhedron

Single translation loop

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

- (obvious) Proposition

$$\tau^*(P_0) = \{x \mid \exists i \in \mathbb{N}, \exists x_0 \in P_0, \\ Ax_0 \leq B, A(x - D) \leq B, x = x_0 + iD\} \cup P_0$$

Not convex

- Discrete vs Dense acceleration

$$\tau^{\otimes}(P_0) = \{x \mid \exists i \in \mathbb{Q}^+, \exists x_0 \in P_0, \\ Ax_0 \leq B, A(x - D) \leq B, x = x_0 + iD\} \sqcup P_0$$

Convex polyhedron

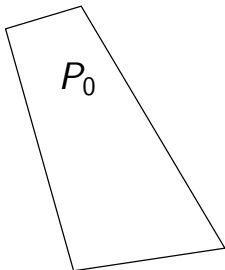
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



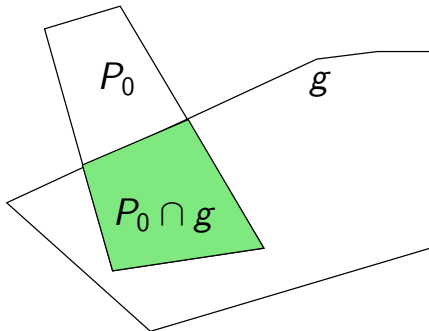
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



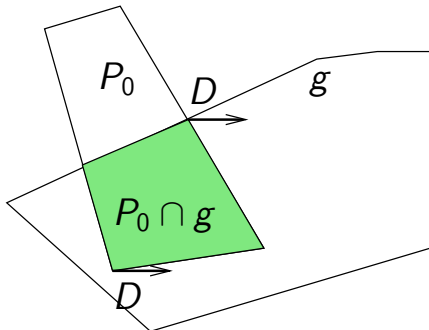
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



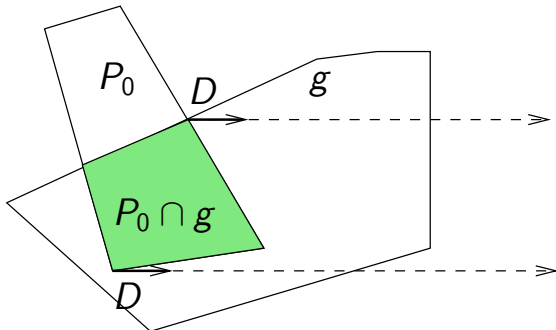
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



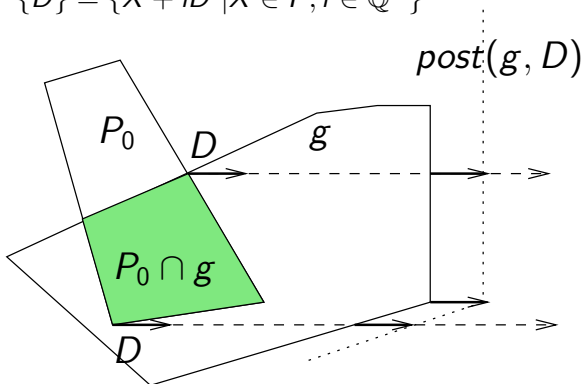
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



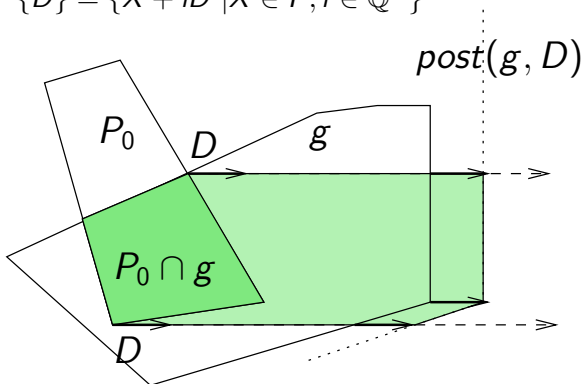
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



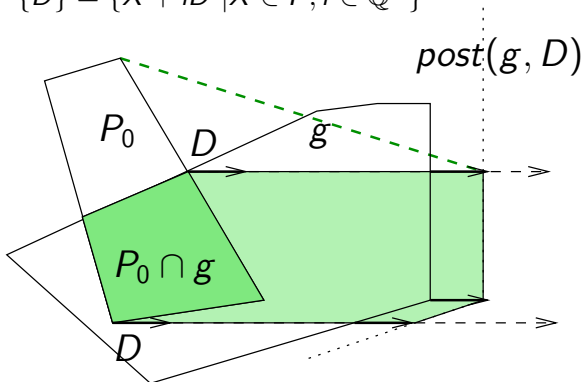
Single translation loop (2)

$$\tau(x) = \text{if } Ax \leq B \text{ then } x + D \text{ else } x$$

Computation : Adding rays :

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

where $P \nearrow \{D\} = \{X + iD \mid X \in P, i \in \mathbb{Q}^+\}$



Finite Monoids simple loops

What do we lose?

$$\tau = \begin{cases} g = (x \leq 7) \\ a = (x := x + 2) \end{cases}$$

$$P_0 = \{x = 0\}$$

$$\tau^*(P_0) = \{0 \leq x \leq 8\}$$

$$\begin{aligned} \tau^{\otimes}(P_0) &= P_0 \nearrow (1) \cap (x - 2 \leq 7) \\ &= \{0 \leq x \leq 9\} \end{aligned}$$

New result (not in the SAS paper) If $\exists p, C^p = C^{2p}$, we can compute in polynomial time an over approximation of $\tau^*(P_0)$. We note it $\tau^{\otimes}(P_0)$.

Question Is this the “best” (convex) overapproximation?

Finite Monoids simple loops

What do we lose?

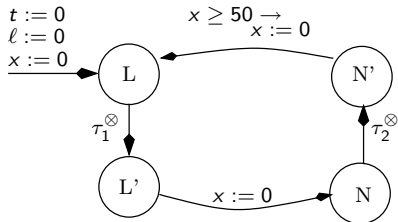
$$\tau = \begin{cases} g = (x \leq 7) \\ a = (x := x + 2) \end{cases} \quad \begin{array}{l} \tau^*(P_0) = \{0 \leq x \leq 8\} \\ \tau^\otimes(P_0) = P_0 \nearrow (1) \cap (x - 2 \leq 7) \\ = \{0 \leq x \leq 9\} \end{array}$$

$$P_0 = \{x = 0\}$$

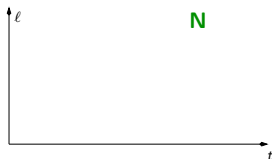
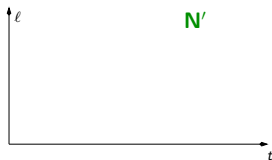
New result (not in the SAS paper) If $\exists p, C^p = C^{2p}$, we can compute in polynomial time an over approximation of $\tau^*(P_0)$. We note it $\tau^\otimes(P_0)$.

Question Is this the “best” (convex) overapproximation?

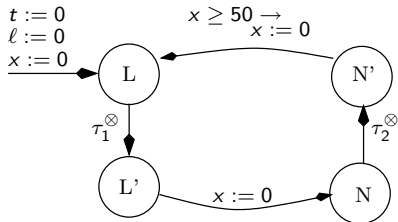
Ex. : gaz burner with acceleration



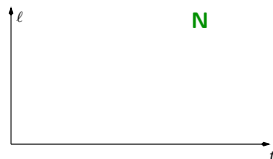
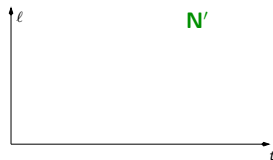
- $\tau_1^\otimes = \text{"add-ray (1, 1, 1) as long as } x \leq 10\text{"}$
- $\tau_2^\otimes = \text{"add-ray (1, 0, 1)"}$



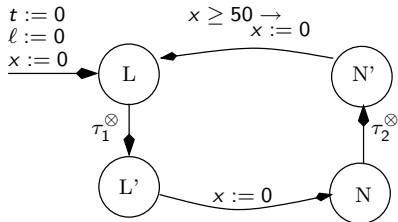
Ex. : gaz burner with acceleration



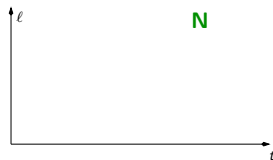
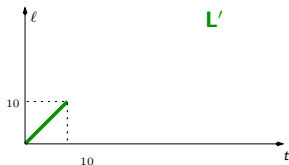
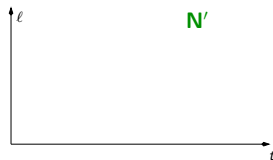
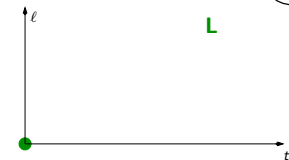
- $\tau_1^\otimes = \text{"add-ray (1, 1, 1) as long as } x \leq 10\text{"}$
- $\tau_2^\otimes = \text{"add-ray (1, 0, 1)"}$



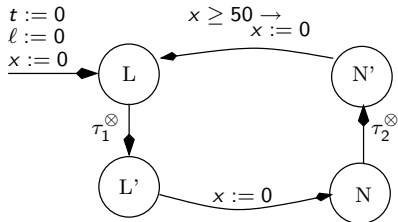
Ex. : gaz burner with acceleration



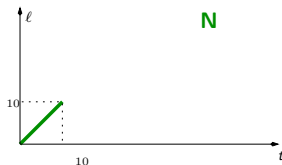
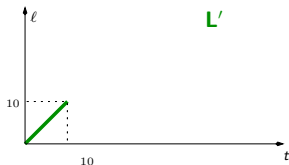
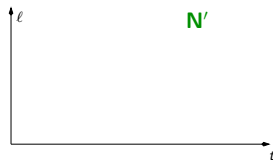
- τ_1^{\otimes} = “add-ray (1, 1, 1) as long as $x \leq 10$ ”
- τ_2^{\otimes} = “add-ray (1, 0, 1)”



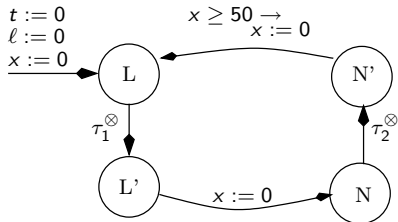
Ex. : gaz burner with acceleration



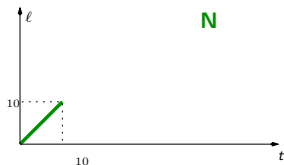
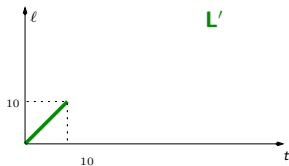
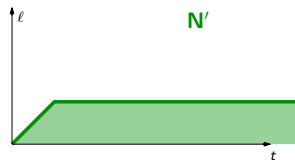
- τ_1^\otimes = “add-ray (1, 1, 1) as long as $x \leq 10$ ”
- τ_2^\otimes = “add-ray (1, 0, 1)”



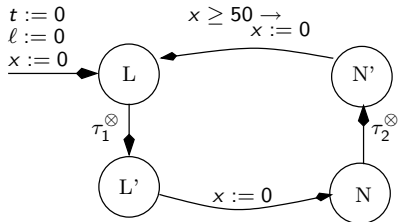
Ex. : gaz burner with acceleration



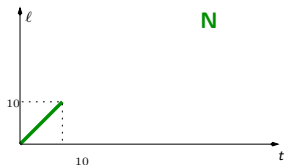
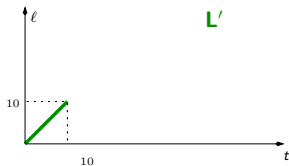
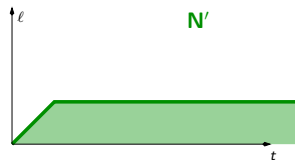
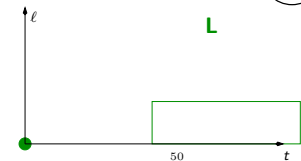
- τ_1^\otimes = “add-ray (1, 1, 1) as long as $x \leq 10$ ”
- τ_2^\otimes = “add-ray (1, 0, 1)”



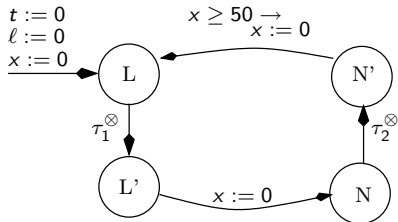
Ex. : gaz burner with acceleration



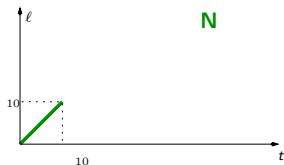
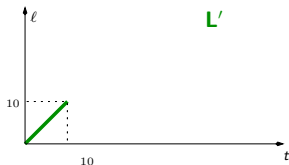
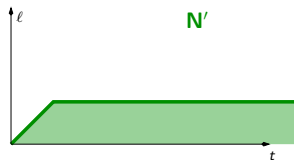
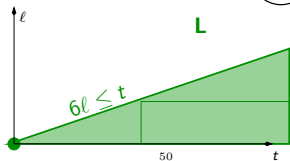
- τ_1^\otimes = "add-ray (1, 1, 1) as long as $x \leq 10$ "
- τ_2^\otimes = "add-ray (1, 0, 1)"



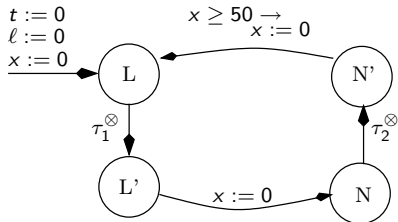
Ex. : gaz burner with acceleration



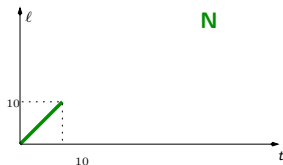
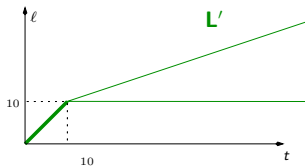
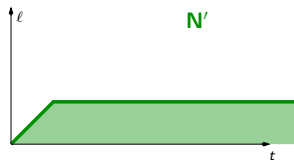
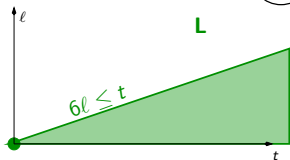
- $\tau_1^\otimes = \text{"add-ray (1, 1, 1) as long as } x \leq 10\text{"}$
- $\tau_2^\otimes = \text{"add-ray (1, 0, 1)"}$



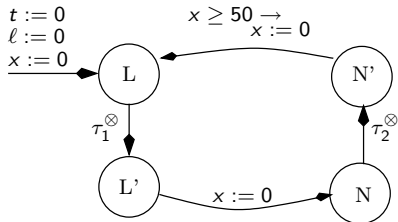
Ex. : gaz burner with acceleration



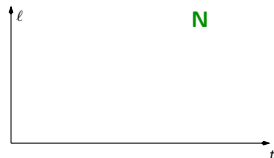
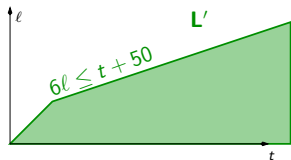
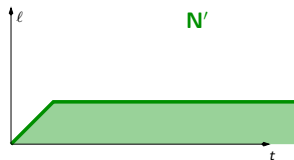
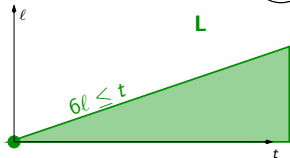
- τ_1^\otimes = “add-ray (1, 1, 1) as long as $x \leq 10$ ”
- τ_2^\otimes = “add-ray (1, 0, 1)”



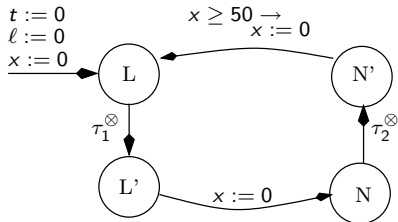
Ex. : gaz burner with acceleration



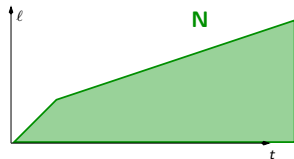
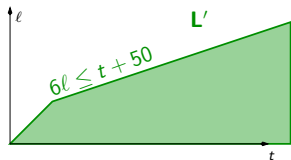
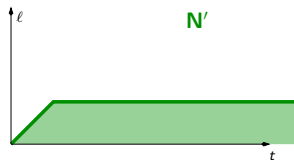
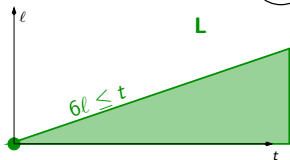
- $\tau_1^\otimes = \text{"add-ray (1, 1, 1) as long as } x \leq 10\text{"}$
- $\tau_2^\otimes = \text{"add-ray (1, 0, 1)"}$



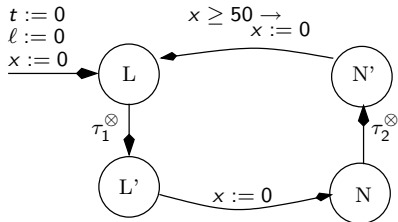
Ex. : gaz burner with acceleration



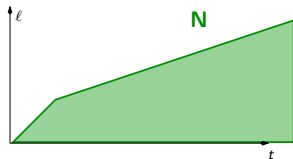
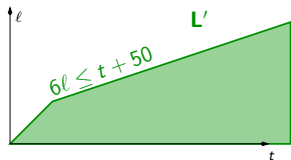
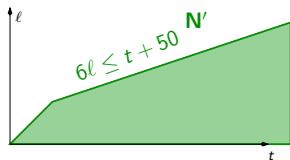
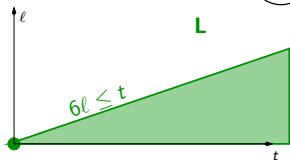
- $\tau_1^\otimes =$ “add-ray (1, 1, 1) as long as $x \leq 10$ ”
- $\tau_2^\otimes =$ “add-ray (1, 0, 1)”



Ex. : gaz burner with acceleration



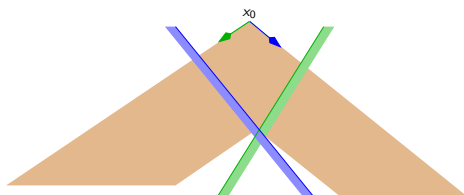
- $\tau_1^\otimes = \text{"add-ray (1, 1, 1) as long as } x \leq 10\text{"}$
- $\tau_2^\otimes = \text{"add-ray (1, 0, 1)"}$



- 1 Introduction and Motivation
- 2 Simple loops
- 3 Two translation loops
- 4 Translations and reset loops
- 5 Implementation - ASPIC

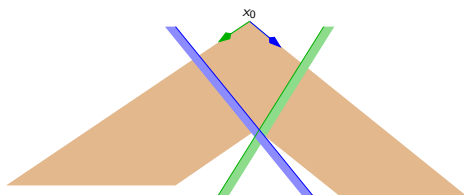
Two loops - First remarks

$(\tau_1 + \tau_2)^*(P_0)$ is not necessarily **convex** :

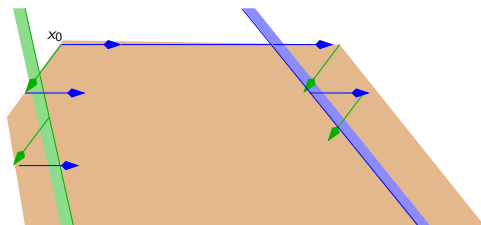


Two loops - First remarks

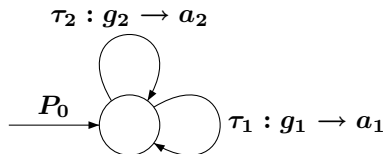
$(\tau_1 + \tau_2)^*(P_0)$ is not necessarily **convex** :



There can be quite complex **oscillations**



Two simple translation loops – new results



- ▶ **New result** for $g_i = G_i X_i \leq c_i$: an algorithm to compute an over-approximation of $(\tau_1 + \tau_2)^*(P_0)$ (convex polyhedron).
- ▶ In other cases, add a new loop computing P_1 :

$$P_1 = P_0 \sqcup \tau_1^{\otimes}(P') \sqcup \tau_2^{\otimes}(P')$$

where :

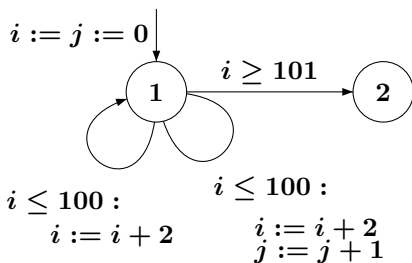
$$P' = (P_0 \cap g_1 \cap g_2 \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2$$

Old Cousot&Halbwachs78 example

```

i := 0;
j := 0;
while i <= 100 do
1   if ? then i := i + 2
    else i := i + 2; j := j + 1
    fi
od
2

```

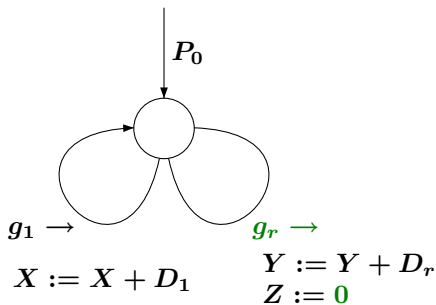


Immediate result at control point 1

$$\begin{aligned}
 (0, 0) \nearrow \{(2, 0), (2, 1)\} \cap (i \leq 100) \\
 = 0 \leq 2j \leq i \leq 100
 \end{aligned}$$

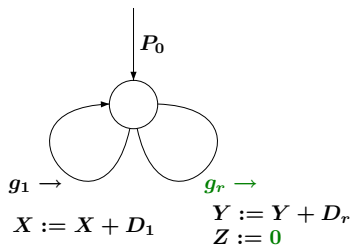
- 1 Introduction and Motivation
- 2 Simple loops
- 3 Two translation loops
- 4 Translations and reset loops**
- 5 Implementation - ASPIC

Two loops with reset (or constant assignments)

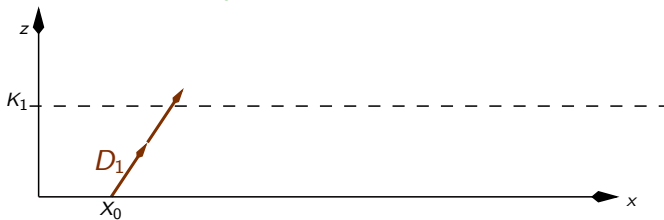


We suppose $P_0 \subseteq \{Z = 0\}$

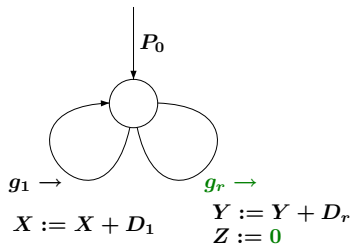
Unconditional simple translation/reset (2)



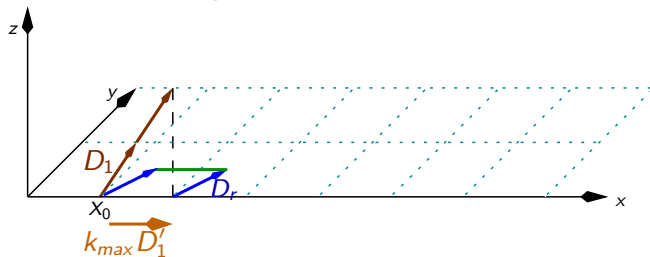
$$g_1 : Z \leq K_1, g_r = true$$



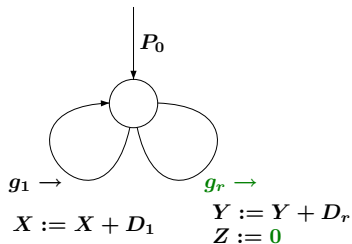
Unconditional simple translation/reset (2)



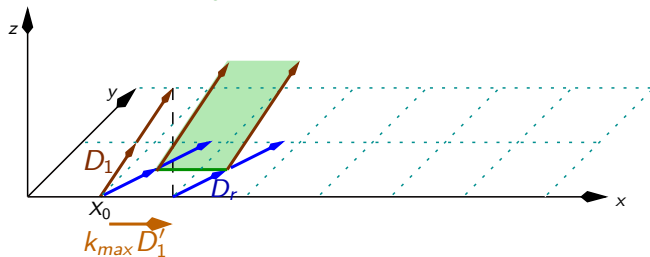
$$g_1 : Z \leq K_1, g_r = true$$



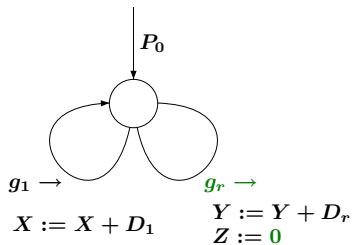
Unconditional simple translation/reset (2)



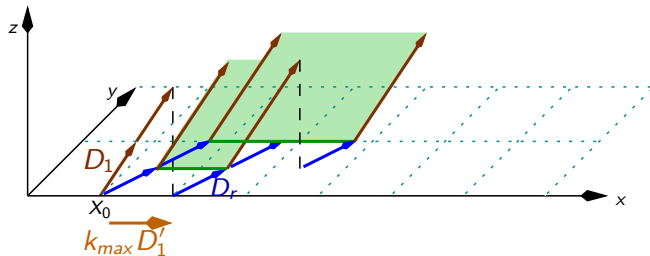
$$g_1 : Z \leq K_1, g_r = true$$



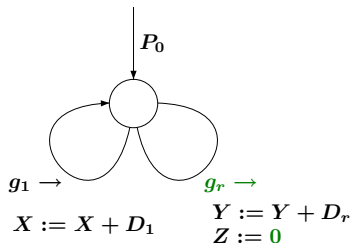
Unconditional simple translation/reset (2)



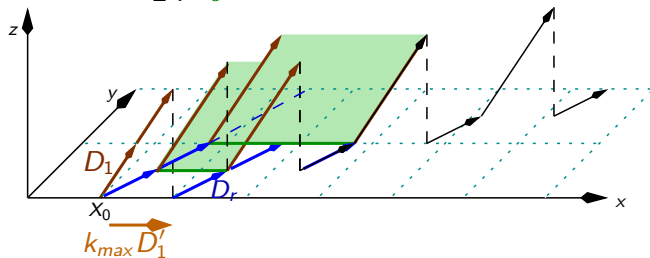
$$g_1 : Z \leq K_1, g_r = true$$



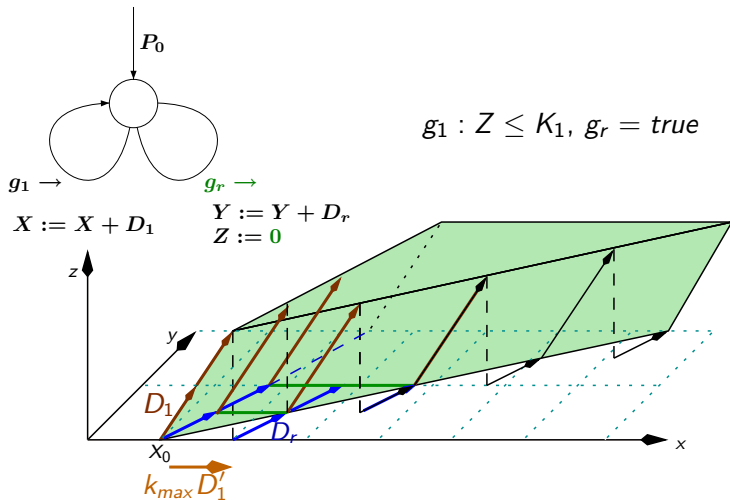
Unconditional simple translation/reset (2)



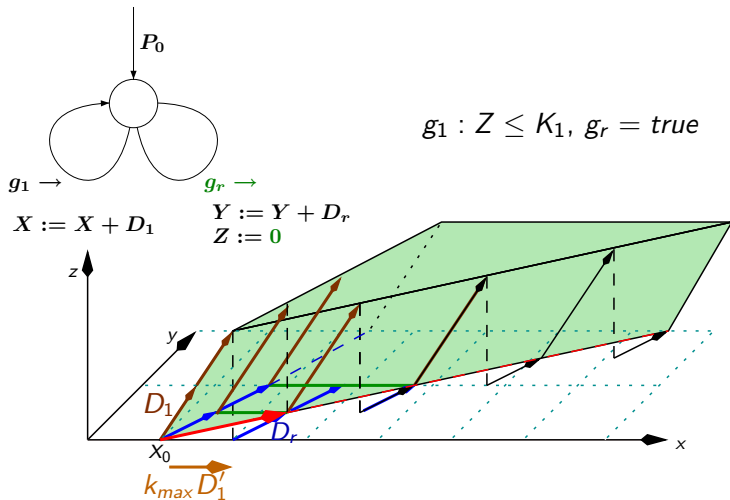
$$g_1 : Z \leq K_1, g_r = true$$



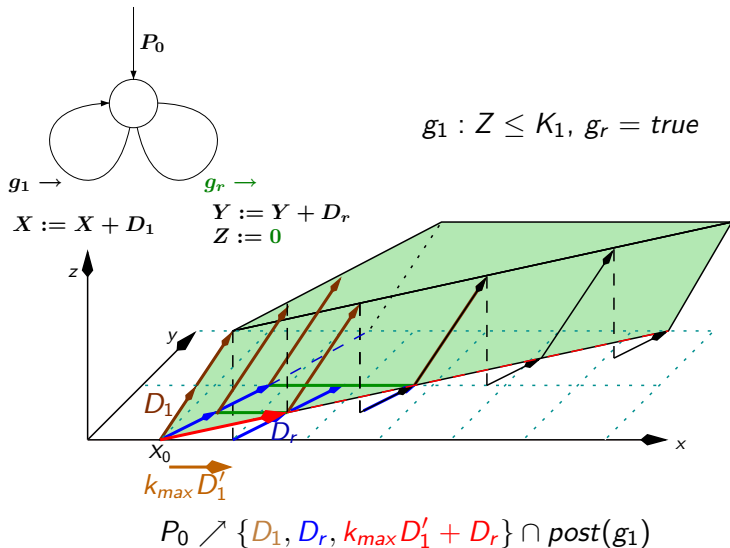
Unconditional simple translation/reset (2)



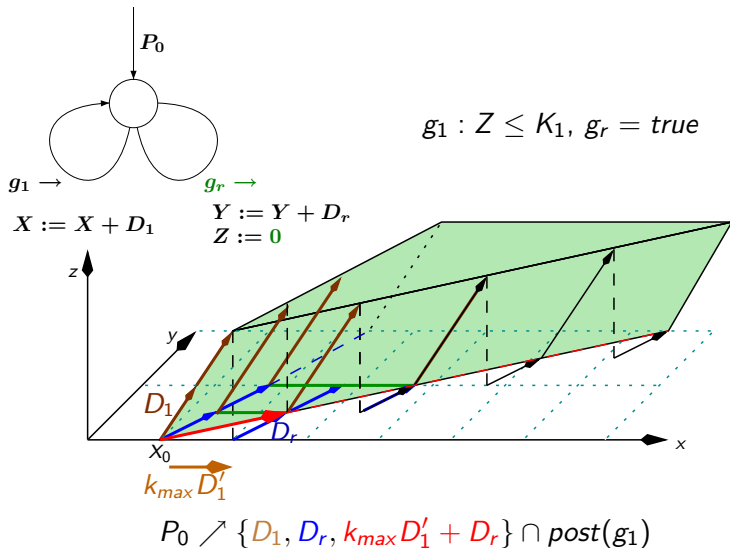
Unconditional simple translation/reset (2)



Unconditional simple translation/reset (2)



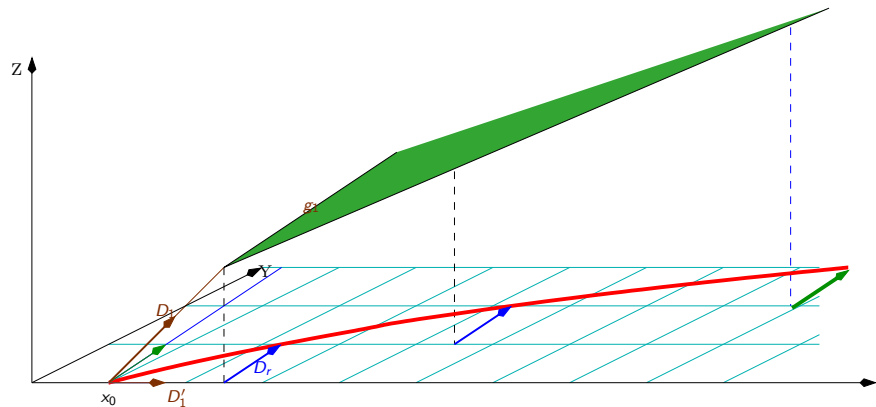
Unconditional simple translation/reset (2)



► More cases in the SAS paper.

Unconditional simple translation/reset (3)

But ... Only works if $g_1 = Z \leq K_1$ (modulo variable change)

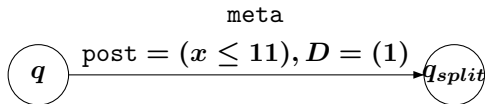
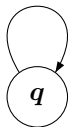


- 1 Introduction and Motivation
- 2 Simple loops
- 3 Two translation loops
- 4 Translations and reset loops
- 5 Implementation - ASPIC

Overview of ASPIC

- A Fixpoint engine for the polyhedral lattice [B. Jeannet]
- Creation of **meta-transitions** if possible :

$$x \leq 10 \rightarrow x++$$



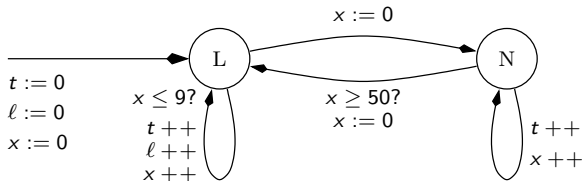
- Classical LRA with forward computation, widening and **meta-transitions**.

Aspic demo

ASPIC : Accelerated Symbolic Polyhedral Invariant Computation

<http://www-verimag.imag.fr/~gonnord/aspic/aspic.html>

demo.



Currently working on

- More general cases of loops.
- Lustre input.
- Sets of benchmarks coming from :
 - Sensor networks : consumption properties (Maraninchi/Samper)
 - "Programs with list are counter automata" (CAV 06, Iosif/Bozga/Bouajjani/Habermehl/Moro/Vojnar)
 - ...

Future work

- More general cases of loops.
- Compare/Combine with “Lookahead widening” (CAV 06, Gopan/Reps)
- Implement in NBAC tool [B.Jeannet]