Proving Termination of flowcharts programs

Laure Gonnord

Laboratoire de l'Informatique du Parallélisme (ENS Lyon)

http://laure.gonnord.org/pro/ — Laure.Gonnord@ens-lyon.fr

Joint work with Christophe Alias, Alain Darte, and Paul Feautrier (Compsys, ENS Lyon), Lucas Seguinot (ENS Bretagne), David Monniaux (Verimag, Grenoble) and Raphael-Ernani. Rodriguez (Univ Mineas Gerais Brasil).

Proving total correctness of programs

Total Correctness

Proving total correctness of programs

Total Correctness = Correctness +

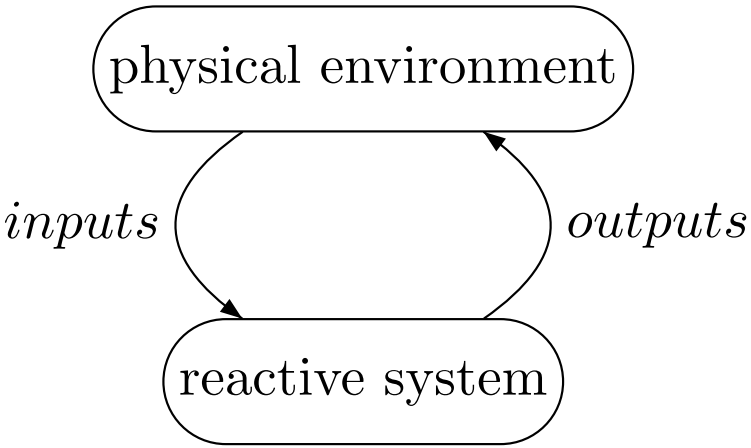
Proving total correctness of programs

Total Correctness = Correctness + Termination

- Proving correctness is useless without termination.

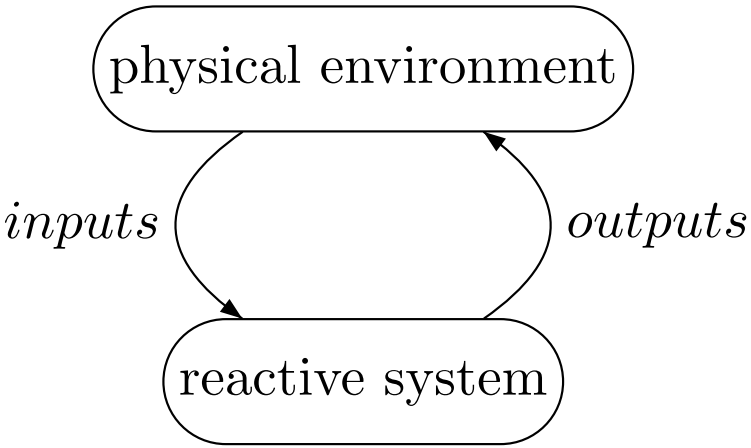
Proving validity of reactive systems software

(CC-BY-SA Captainm/Wikipedia)



Proving validity of reactive systems software

(CC-BY-SA Captainm/Wikipedia)



- Termination (+worst case exec. time) of each step of computation.

For fun !

But

Termination (HALTING PROBLEM) is undecidable !

But

Termination (HALTING PROBLEM) is undecidable !

- Use **conservative algorithms** : YES (+ witness) or “Don’t Know” (+ potential infinite path)

But

Termination (HALTING PROBLEM) is undecidable !

- Use **conservative algorithms** : YES (+ witness) or “Don’t Know” (+ potential infinite path)
- On **restricted** classes of programs.

Hoare rule [1969] for total correctness

Partial correctness :

$$\frac{\{\ \mathrm{P\ and\ B}\ \}\ \mathrm{S}\ \{\ \mathrm{P}\ \ \}}{\{\mathrm{P}\}\ \textbf{while}\ \mathrm{B}\ \textbf{do}\ \mathrm{S}\ \textbf{done}\ \{\mathrm{not(B)\ and\ P}\}}$$

Hoare rule [1969] for total correctness

Total correctness :

$$\frac{\{\ \mathrm{t{=}z\ and\ t{\in}\ D\ and\ \ P\ and\ B}\ \}\ \mathrm{S}\ \{\ \mathrm{P\ \ and\ t{<}z\ and\ t{\in}\ D}\ \} \qquad (D,<)\ \text{well-founded}}{\{\mathrm{P}\}\ \textbf{while}\ \mathrm{B}\ \textbf{do}\ \mathrm{S}\ \textbf{done}\ \{\mathrm{not(B)\ and\ P}\}}$$

Hoare rule [1969] for total correctness

Total correctness :

$$\frac{\{\ \mathrm{t{=}z\ and\ t{\in}\ D\ and\ \ P\ and\ B}\ \}\ \mathrm{S}\ \{\ \mathrm{P\ \ and\ t{<}z\ and\ t{\in}\ D}\ \} \qquad (D,<)\ \text{well-founded}}{\{\mathrm{P}\}\ \textbf{while}\ \mathrm{B}\ \textbf{do}\ \mathrm{S}\ \textbf{done}\ \{\mathrm{not(B)\ and\ P}\}}$$

- Find $(D,<)$ and t !

First easy example

```
assume(N>0);
i=N;
while(i>0) --i;
```

First easy example

```
assume(N>0);
i=N;
while(i>0) --i;
```

- $(\mathbb{N}, <)$ and $t = i$.

Restriction

In this talk, we only focus on :

Numerical (sequential) flowcharts programs

no thread, no recursive call, no function call, no list, no pointer. . . ...

- A great restriction, but still undecidable

Restriction

In this talk, we only focus on :

Numerical (sequential) flowcharts programs

no thread, no recursive call, no function call, no list, no pointer. . . ...

- A great restriction, but still undecidable
- We are able to synthesize **ranking functions** in some cases.

Agenda

- A (conservative) algorithm to find **affine ranking functions**.
- Scalability issues and other improvements.

Summary

(C) Program $\rightarrow$ Model of program$\rightarrow$ Affine ranking function

Summary

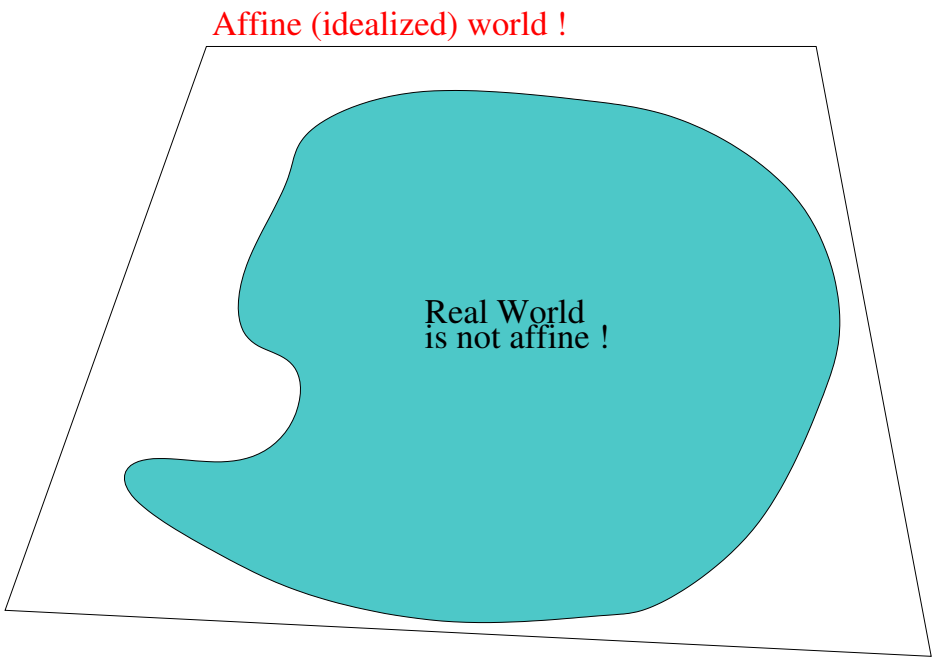
(C) Program $\rightarrow$ Model of program$\rightarrow$ Affine ranking function

Objective

(C) Program $\rightarrow$ Model of program $\rightarrow$ Affine ranking function

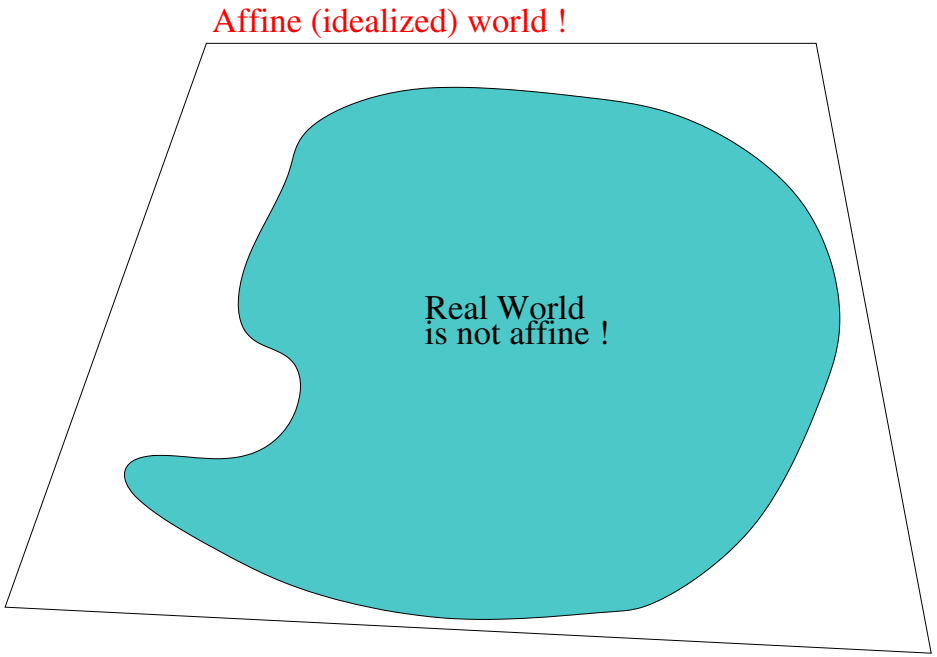
Objective

(C) Program $\rightarrow$ Affine Automaton $\rightarrow$ Affine ranking function



Objective

(C) Program → Affine Automaton → Affine ranking function

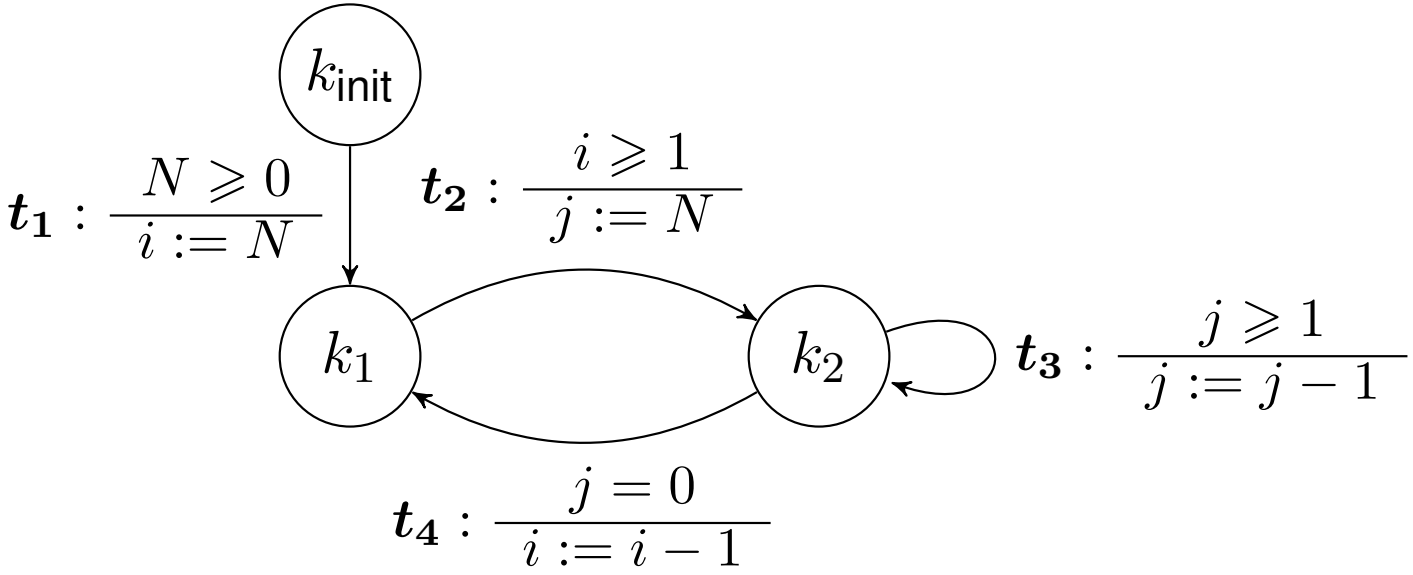


- **Pb** Compute an accurate (affine) model of a given C program ?

Our model for programs

Interpreted affine automata $(\mathcal{K}, n, k_{init}, \mathcal{T})$

- $\mathcal{K}$: control points
- n rational variables $\boldsymbol{x}$
- $k_{init} \in \mathcal{K}$ the initial control point
- $\mathcal{T}$ the set of transitions (k, g, a, k')

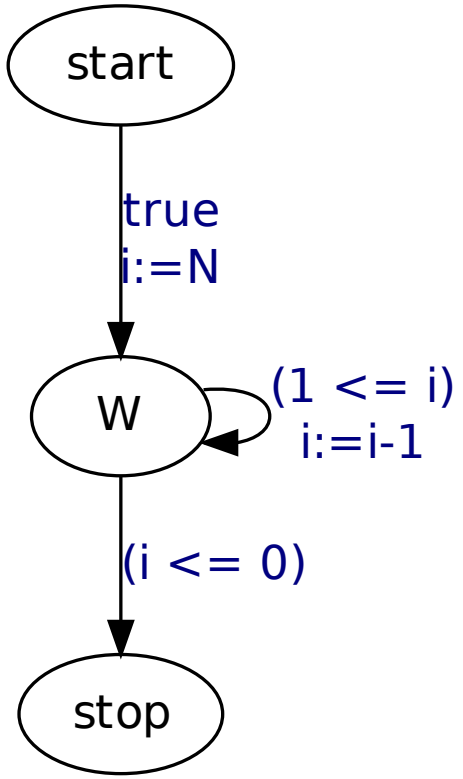


From a Program to an affine Automaton I

Program → Control flow graph is the first step of **compilation**.

```
assume(N>0);
i=N;
while(i>0) --i;
```

→

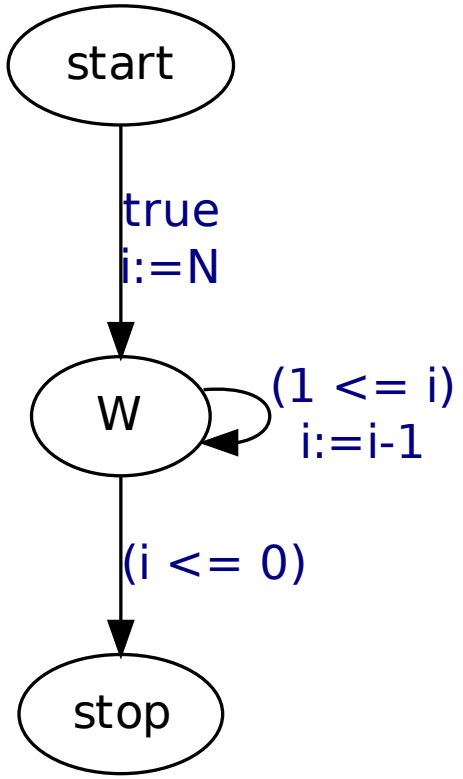


From a Program to an affine Automaton I

Program → Control flow graph is the first step of **compilation**.

```
assume(N>0);
i=N;
while(i>0) --i;
```

→



Basics : tests → branches ; for/while → loops.

From a Program to an affine Automaton II

The rest is not (just) ugly syntax stuff : we have also to abstract non numerical behaviors :

- Handle all C control-flow instructions (syntax !)
- Safe Abstractions for all data structures, arrays, pointers, floating point . . .
- How to handle programs with functions calls ?

▶ Home-made tool C2FSM ([FG10]. Demo.

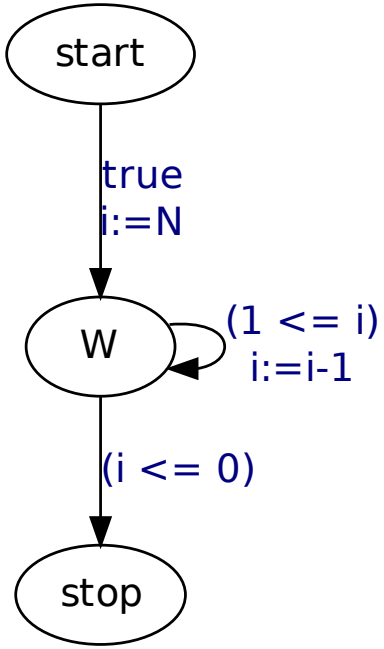
Termination for affine automata (I)

What is a **ranking function** for a given affine automaton ?

- A mapping from $(state, value)$ to a well-founded set
- Decreasing (strictly) on each transition.

Termination for affine automata (II)

Monodimensional affine ranking function : $(\mathbb{N}, <)$



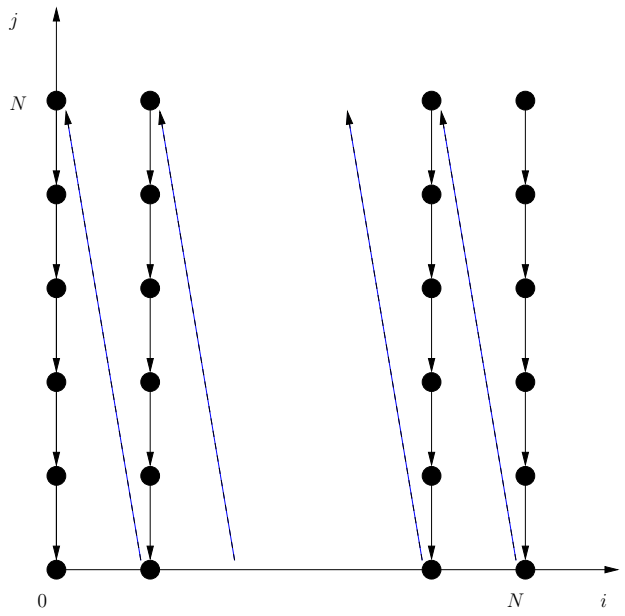
$$\rho(st, i, N) = \begin{cases} 2 + N_0 & \text{if } st = start \\ i + 1 & \text{if } st = W \\ 0 & \text{if } st = stop \end{cases}$$

Termination for affine automata (III)

Multidimensional affine ranking function : $(\mathbb{N}^d, <_{lex})$

$$\rho(k, \vec{x}) = A_k.\vec{x} + \vec{b_k}$$

```
//N>0
i = N;
while(i>0)
{
    j = N;
    while(j>0) j--;
    i--;
}
```



Recall the global objective

(C) Program $\rightarrow$ Affine automaton
$\rightarrow$ Affine ranking function

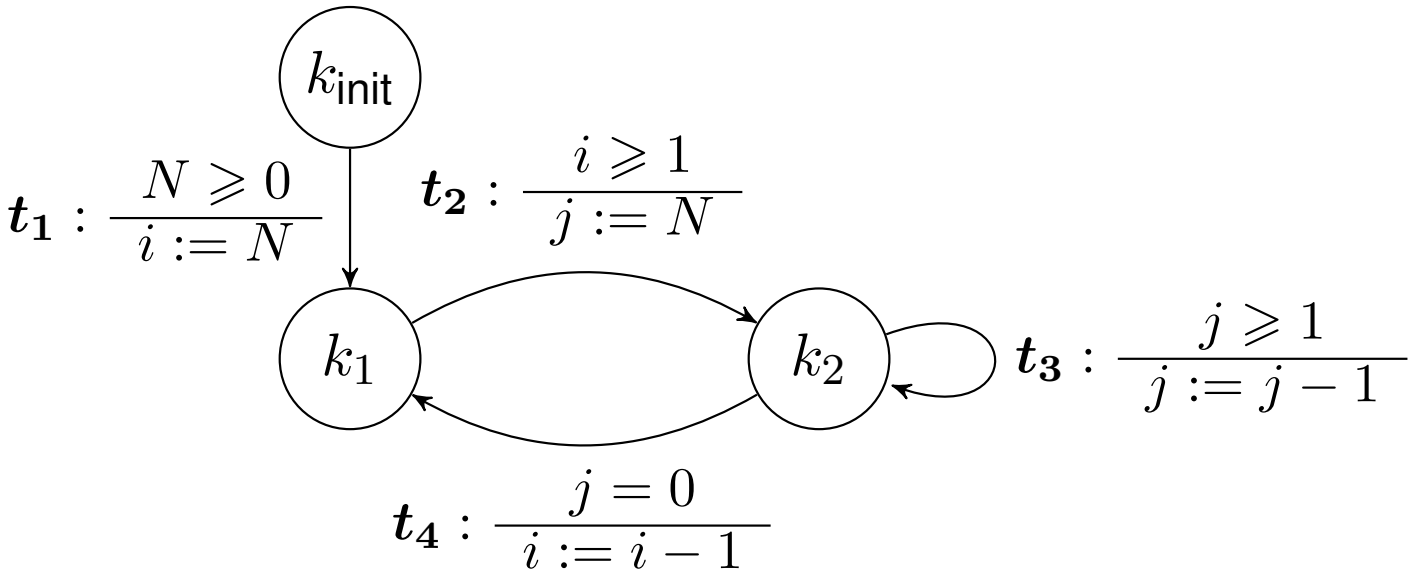
Recall the global objective

(C) Program $\rightarrow$ Affine automaton
$\rightarrow$Affine invariants
$\rightarrow$ Affine ranking function

Harry Potter, (chemin “de traverse”)

Invariants

Invariant = formula that over-approximates all the possible values of the variables.



$$k_1 : N \geqslant 0 \wedge i \geqslant 0 \wedge i \leqslant N$$

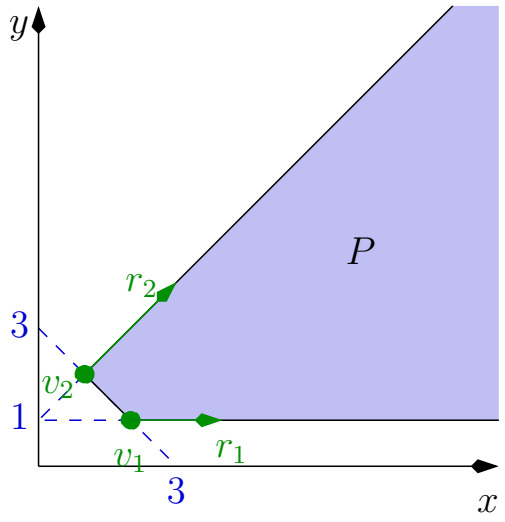
From an affine Automaton to invariants

Pb : how to compute numerical invariants ?

- Accurate numerical invariants.
- Polyhedra (conjunction of affine constraints).
- Within a reasonable amount of time.

From an affine Automaton to invariants

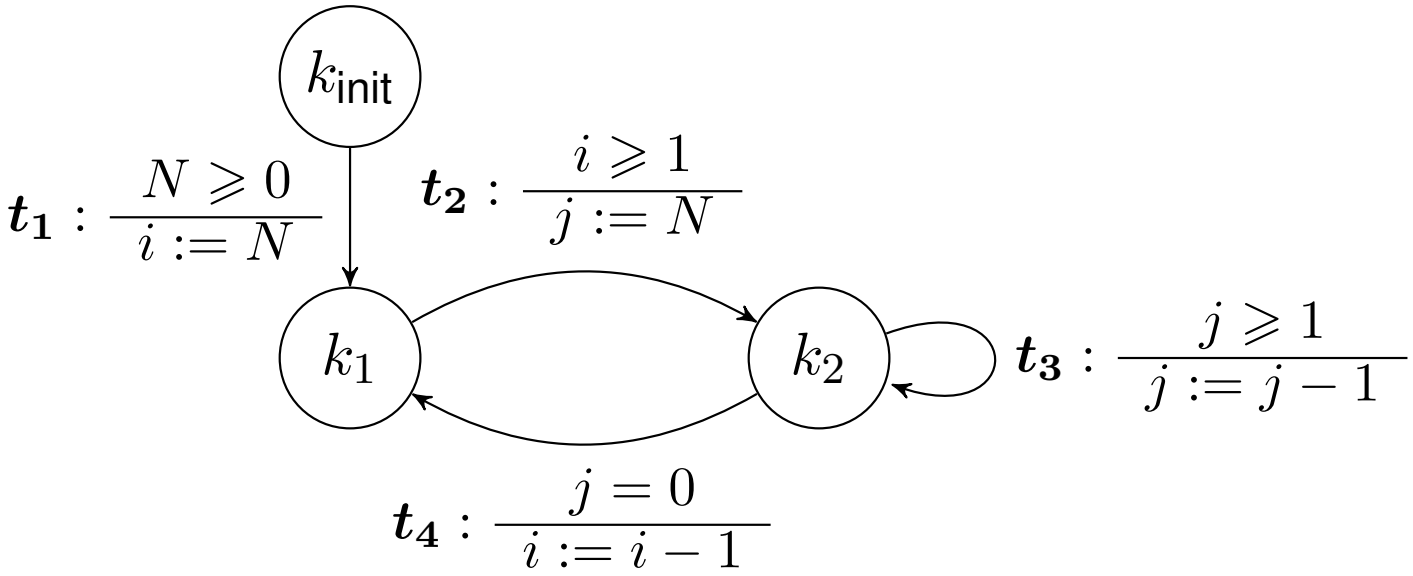
- We use **abstract interpretation** with the polyhedral abstract domain :



Tools : ASPIC([FG10]) or the promising Pagai (J. Henry, [HMM12])

From an affine Automaton to invariants

Abstract Interpretation in a nutshell :



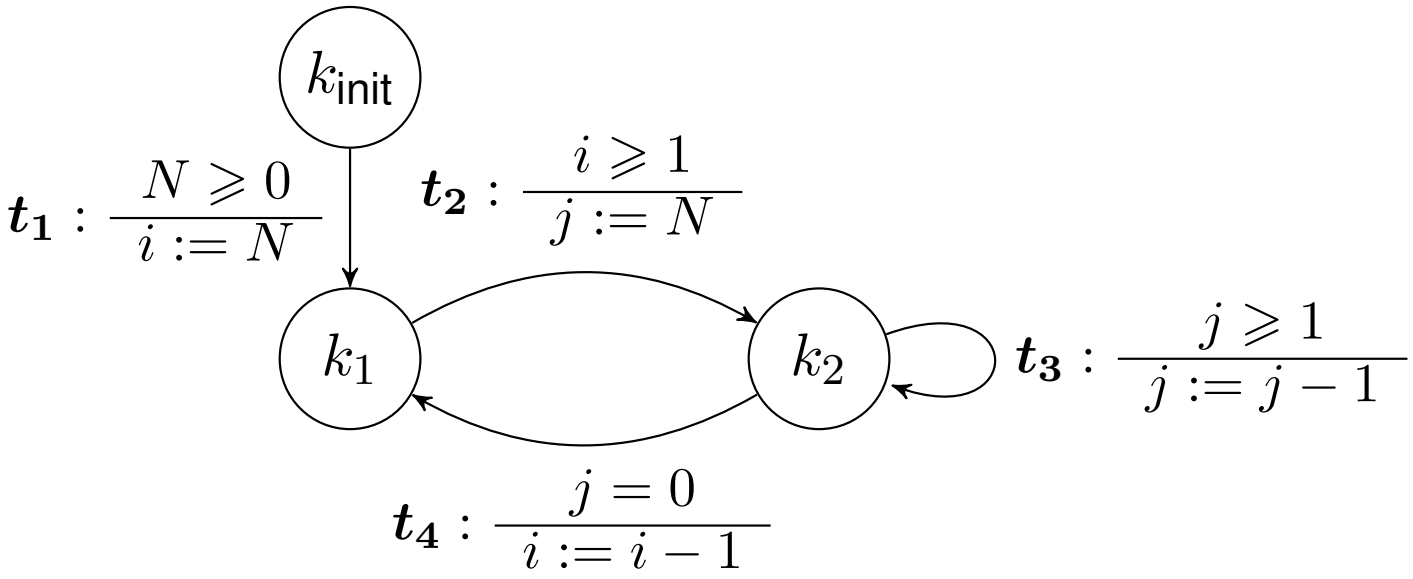
System of equations :

$$\begin{cases} P_{init} & = & all \\ \boldsymbol{P_{k_1}} & = & t_1(P_{init}) \cup t_4(\boldsymbol{P_{k_2}}) \\ \boldsymbol{P_{k_2}} & = & t_2(P_{k_1}) \cup t_3(\boldsymbol{P_{k_2}}) \end{cases}$$

.

From an affine Automaton to invariants

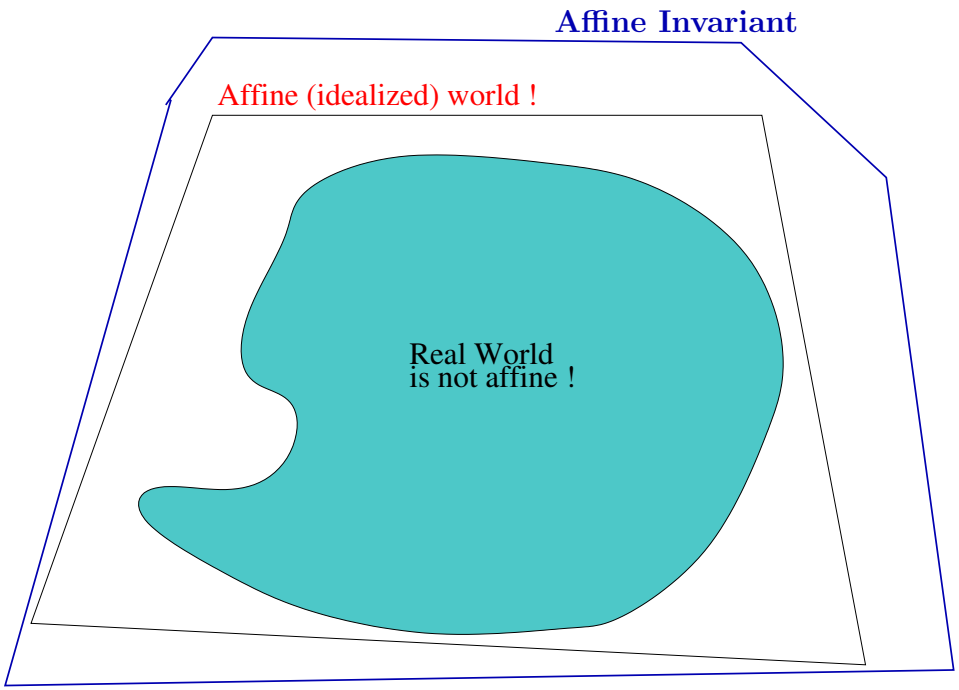
Abstract Interpretation in a nutshell :



- Fixpoint system with affine guards and actions. A Kleene iteration + a special iterator (widening) provides **overapproximated** polyhedral invariants.

Conclusion of this part

We are able to compute over-approximations of the numerical behavior of C programs :



What's next ?

Introduction

Problem statement

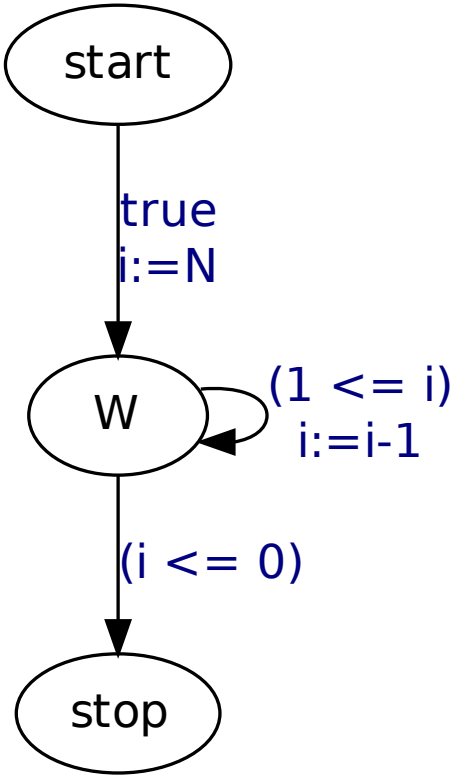
Given :

- An affine automaton.
- Some affine invariants on each control point.

Find a 1D (affine) ranking function.

Finding a 1D-ranking function as an affine form

```
assume(N>0);
i=N;
while(i>0) --i;
```

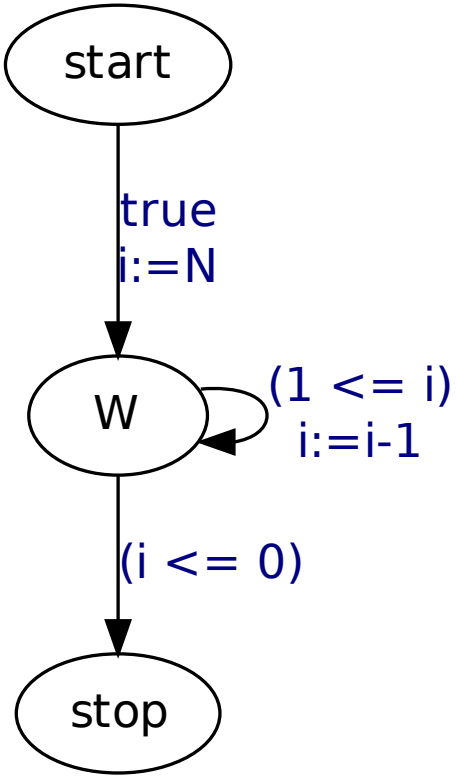


Searching for $\alpha_{pc,-} \in \mathbb{Q}$:

$$\begin{aligned}
\rho(start, \vec{x}) &= \alpha_{start,1}.\mathbf{i} + \alpha_{start,2}.\mathbf{N} \\
&+ \alpha_{start,3}.\mathbf{i_0} + \alpha_{start,4}.\mathbf{N_0} \\
&+ \alpha_{start,5} \\
\rho(W, \vec{x}) &= \alpha_{W,1}.\mathbf{i} + \ldots \\
\rho(stop, \vec{x}) &= \alpha_{stop,1}.\mathbf{i} + \ldots
\end{aligned}$$

Finding a 1D-ranking function as an affine form

```
assume(N>0);
i=N;
while(i>0) --i;
```



Searching for $\alpha_{pc,-} \in \mathbb{Q}$:

$$\begin{aligned}
\rho(start, \vec{x}) &= \alpha_{start,1}.\mathbf{i} + \alpha_{start,2}.\mathbf{N} \\
&+ \alpha_{start,3}.\mathbf{i_0} + \alpha_{start,4}.\mathbf{N_0} \\
&+ \alpha_{start,5} \\
\rho(W, \vec{x}) &= \alpha_{W,1}.\mathbf{i} + \ldots \\
\rho(stop, \vec{x}) &= \alpha_{stop,1}.\mathbf{i} + \ldots
\end{aligned}$$

The constraints are :

- For each control point : $\rho(pc, \vec{x}) \geqslant 0$ on P_{pc}
- For each transition $(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x}) > 0$

ArgIII, “forall” constraints

$\rho(pc, \vec{x}) \geqslant 0$ on P_W gives (control point W) :

$$\forall i, N \in P_W, \alpha_{W,1}.\mathbf{i} + \ldots \geqslant 0$$

$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x}) > 0$ for the “loop transition” :

$$\forall i, N, i', N' \in P_{transition}, \alpha_{W,1}(i' - i) + \ldots > 0$$

Argll, “forall” constraints

$\rho(pc, \vec{x}) \geqslant 0$ on P_W gives (control point W) :

$$\forall i, N \in P_W, \alpha_{W,1}.\mathbf{i} + \ldots \geqslant 0$$

$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x}) > 0$ for the “loop transition” :

$$\forall i, N, i', N' \in P_{transition}, \alpha_{W,1}(i' - i) + \ldots > 0$$

Unknowns are $\alpha_{*,*}$. “Forall” in (possibly) **infinite domains** ! ?

A very useful theorem

Farkas Lemma

An affine form which is positive on a (convex) polyhedron can be expressed as a linear combination of the polyhedron's constraints.

Finding a 1D ranking function : linearization

1- Constraints for **control points** : $\rho(pc, \vec{x}) \geqslant 0$ on P_{pc}.

Here (for W) $P_W = \{N_0 > 0, N = N_0, 0 \leqslant i \leqslant N\}$ thus :

$$\begin{aligned} \rho(W, \vec{x}) &= \lambda_{W,1}.(N_0 - 1) + \lambda_{W,2}.(N_0 - N) \\ &+ \lambda_{W,3}.(N - N_0) + \lambda_{W,4}.\mathbf{i} + \lambda_{W,3}.(N - \mathbf{i}) \end{aligned}$$

Finding a 1D ranking function : linearization

1- Constraints for **control points** : $\rho(pc, \vec{x}) \geqslant 0$ on P_{pc}.

Here (for W) $P_W = \{N_0 > 0, N = N_0, 0 \leqslant i \leqslant N\}$ thus :

$$\begin{aligned} \rho(W, \vec{x}) &= \lambda_{W,1}.(N_0 - 1) + \lambda_{W,2}.(N_0 - N) \\ &+ \lambda_{W,3}.(N - N_0) + \lambda_{W,4}.\mathbf{i} + \lambda_{W,3}.(N - \mathbf{i}) \end{aligned}$$

We were looking for $\rho(W, \vec{x})$ with the following "template" :

$$\rho(W, \vec{x}) = \alpha_{W,1}.\mathbf{i} + \alpha_{W,2}.N + \alpha_{W,3}.i_0 + \alpha_{W,4}.N_0 + \alpha_{W,3}$$

- Identifying coefficients for $\boldsymbol{i}$: $\alpha_{W,1} = \lambda_{W,4} - \lambda_{W,3}, \ldots$

Finding a 1D ranking function : linearization

1- Constraints for **control points** : $\rho(pc, \vec{x}) \geqslant 0$ on P_{pc}.

Here (for W) $P_W = \{N_0 > 0, N = N_0, 0 \leqslant i \leqslant N\}$ thus :

$$\begin{aligned} \rho(W, \vec{x}) &= \lambda_{W,1}.(N_0 - 1) + \lambda_{W,2}.(N_0 - N) \\ &+ \lambda_{W,3}.(N - N_0) + \lambda_{W,4}.\mathbf{i} + \lambda_{W,3}.(N - \mathbf{i}) \end{aligned}$$

We were looking for $\rho(W, \vec{x})$ with the following "template" :

$$\rho(W, \vec{x}) = \alpha_{W,1}.\mathbf{i} + \alpha_{W,2}.N + \alpha_{W,3}.i_0 + \alpha_{W,4}.N_0 + \alpha_{W,3}$$

- Identifying coefficients for $\boldsymbol{i}$: $\alpha_{W,1} = \lambda_{W,4} - \lambda_{W,3}, \ldots$
- We solved the **for all** problem.

Finding a 1D ranking function - linearization and solving

2- **Decreasing transitions** :

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$$

also gives affine constraints.

Finding a 1D ranking function - linearization and solving

2- **Decreasing transitions** :

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$$

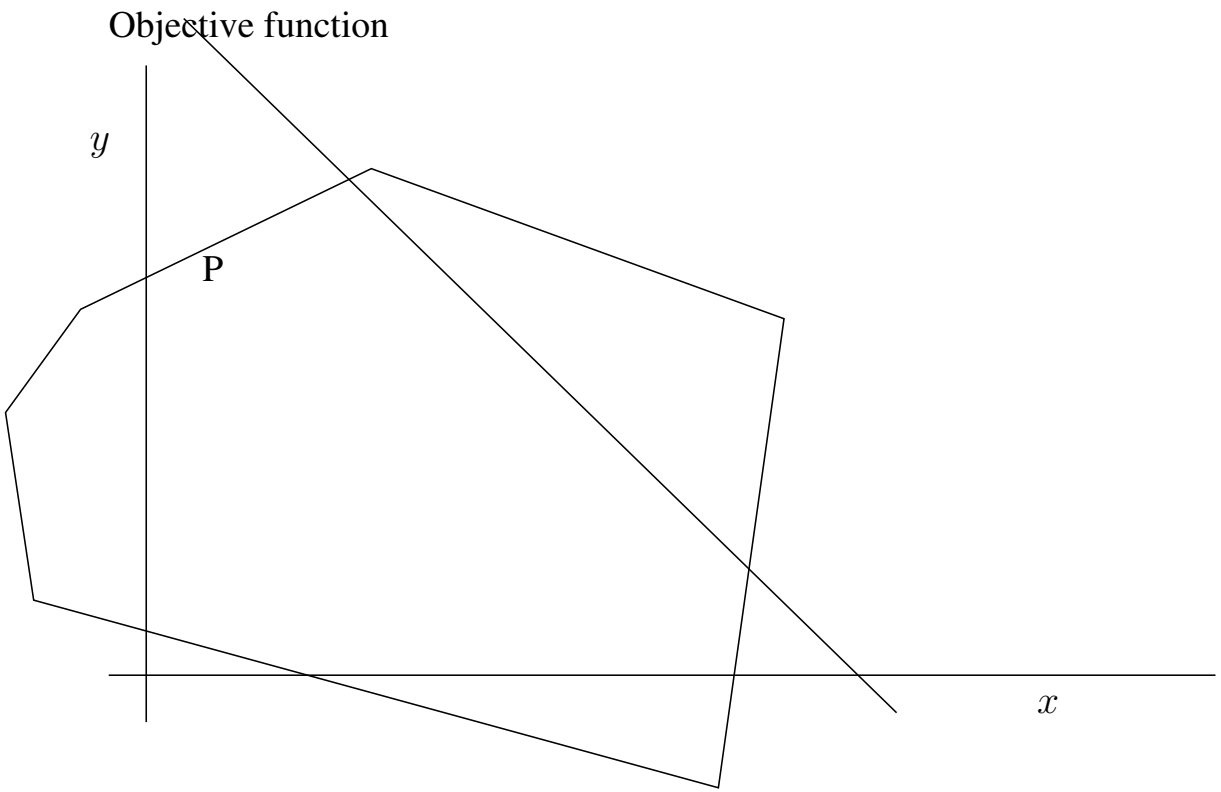
also gives affine constraints.

- A set of affine constraints. A **Linear Programming solver** gives a model, which solves the problem.

Digression

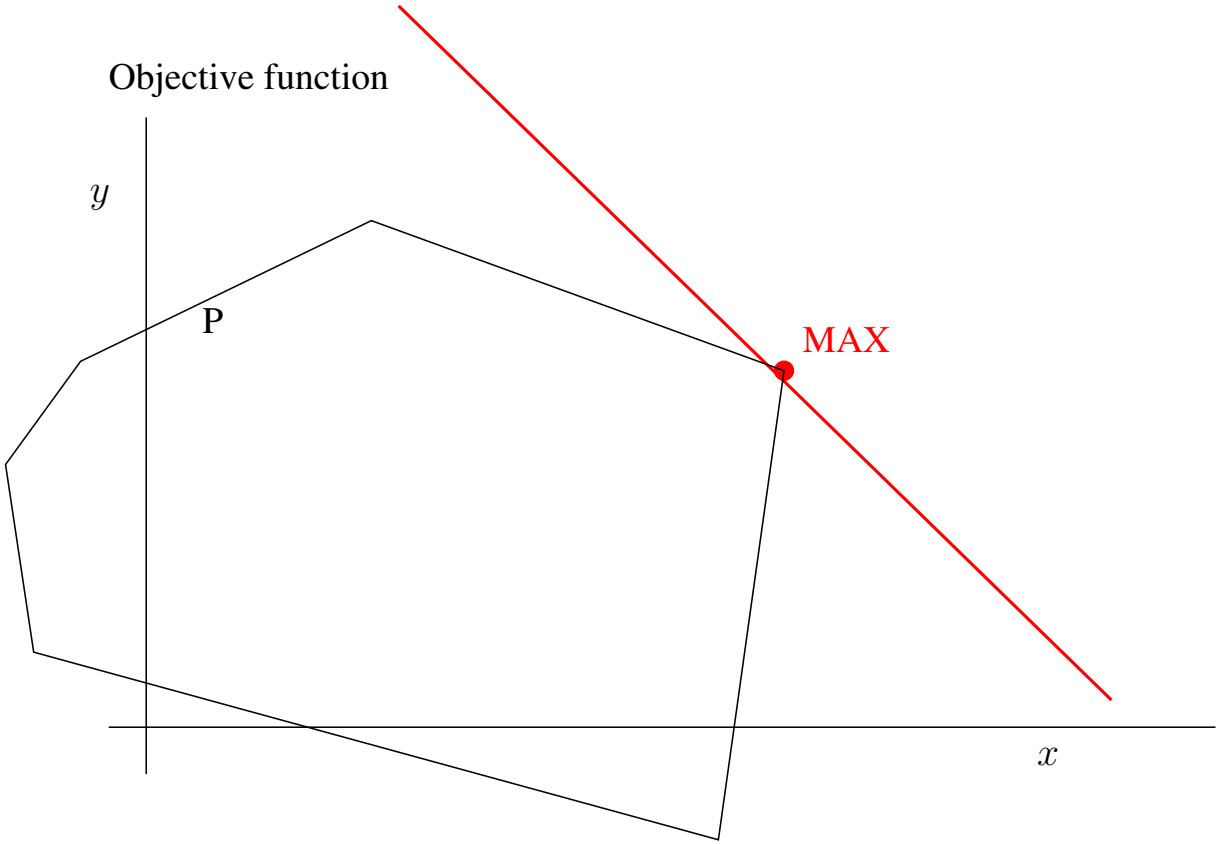
LP solving

Maximise $x + y$ on P :



LP solving

Maximise $x + y$ on P :



LP solving

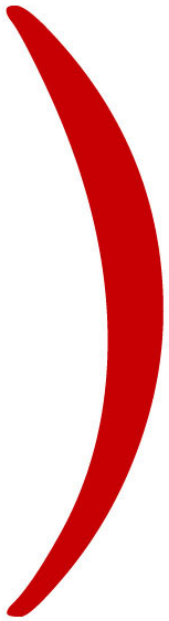
No objective function $\rightarrow$ gives a point iff P is not empty.

LP solving

No objective function $\rightarrow$ gives a point iff P is not empty.

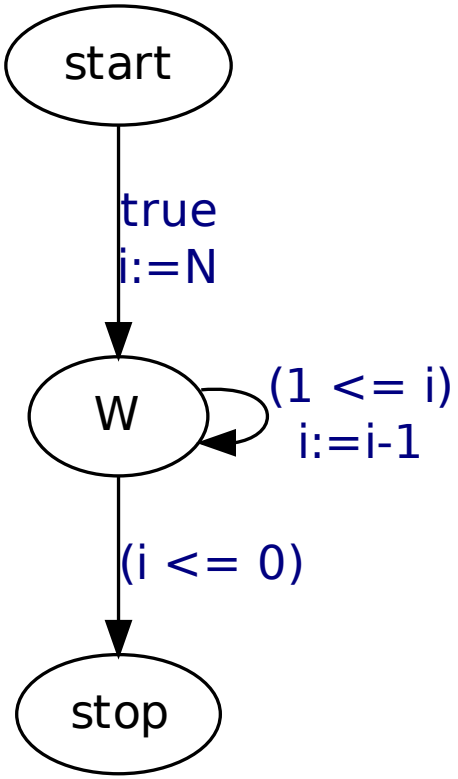
- Software : CPLEX, PIP, ...

End of digression



Finding a 1D ranking function - example/demo

```
assume(N>0);
i=N;
while(i>0) --i;
```



We find :

```
state start:
2+N__o

state W:
1+i

state stop:
0
```

But

Scoop : all programs are not **linear** !

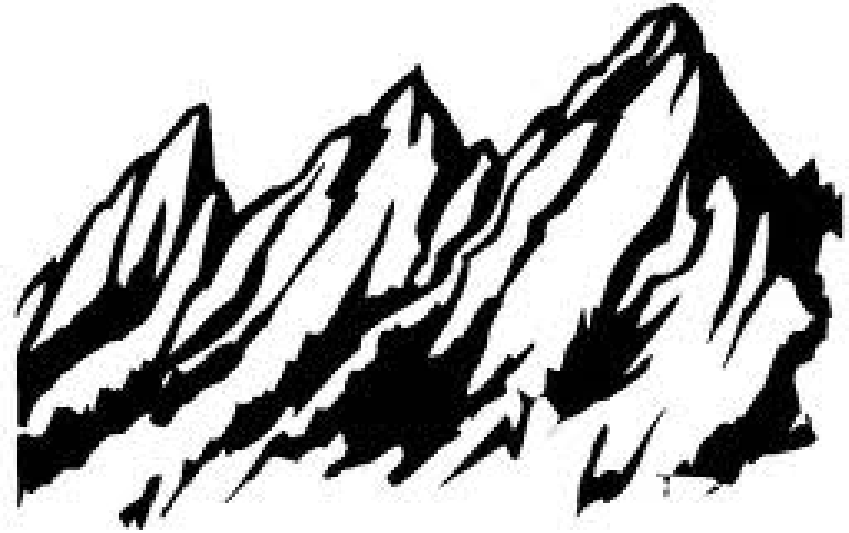
- Synthesize **multidimensional** ranking functions.

The main idea

Idea

A multidimensional affine function is a **vector** of monodimensional (**partial**) ranking functions.

$$\rho = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \dots \\ \rho_d \end{pmatrix}$$



Finding a ranking function - nD

The multidimensional-case, a **greedy algorithm**

- $i = 0$; $T = \mathcal{T}$, set of all transitions.
- While T is not empty do
  - Find a 1D affine function σ, not increasing for any transition, and decreasing for as many transitions as possible.
  - Let $\rho_i = \sigma$; $i = i + 1$; (i^{th} dimension)
  - If no transition is decreasing, **return false**.
  - Remove from T all decreasing transitions.
- $d = i$, **return true**.

Finding a ranking function - nD

The multidimensional-case, a **greedy algorithm**

- $i = 0$; $T = \mathcal{T}$, set of all transitions.
- While T is not empty do
  - Find a 1D affine function σ, not increasing for any transition, and **decreasing for as many transitions as possible.**
  - Let $\rho_i = \sigma$; $i = i + 1$; (i^{th} dimension)
  - If no transition is decreasing, **return false**.
  - Remove from T all decreasing transitions.
- $d = i$, **return true**.

Modification of the constraint system

Pb How do we implement "**decreasing for as many transitions as possible**" in the LP instance ?

Decreasing transitions constraints :

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$$

$\rightarrow$

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) \geqslant \epsilon_t$$

with $0 \leqslant \epsilon_t \leqslant 1$

Modification of the constraint system

Pb How do we implement "**decreasing for as many transitions as possible**" in the LP instance ?

Decreasing transitions constraints :

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$$

$\rightarrow$

$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) \geqslant \epsilon_t$$

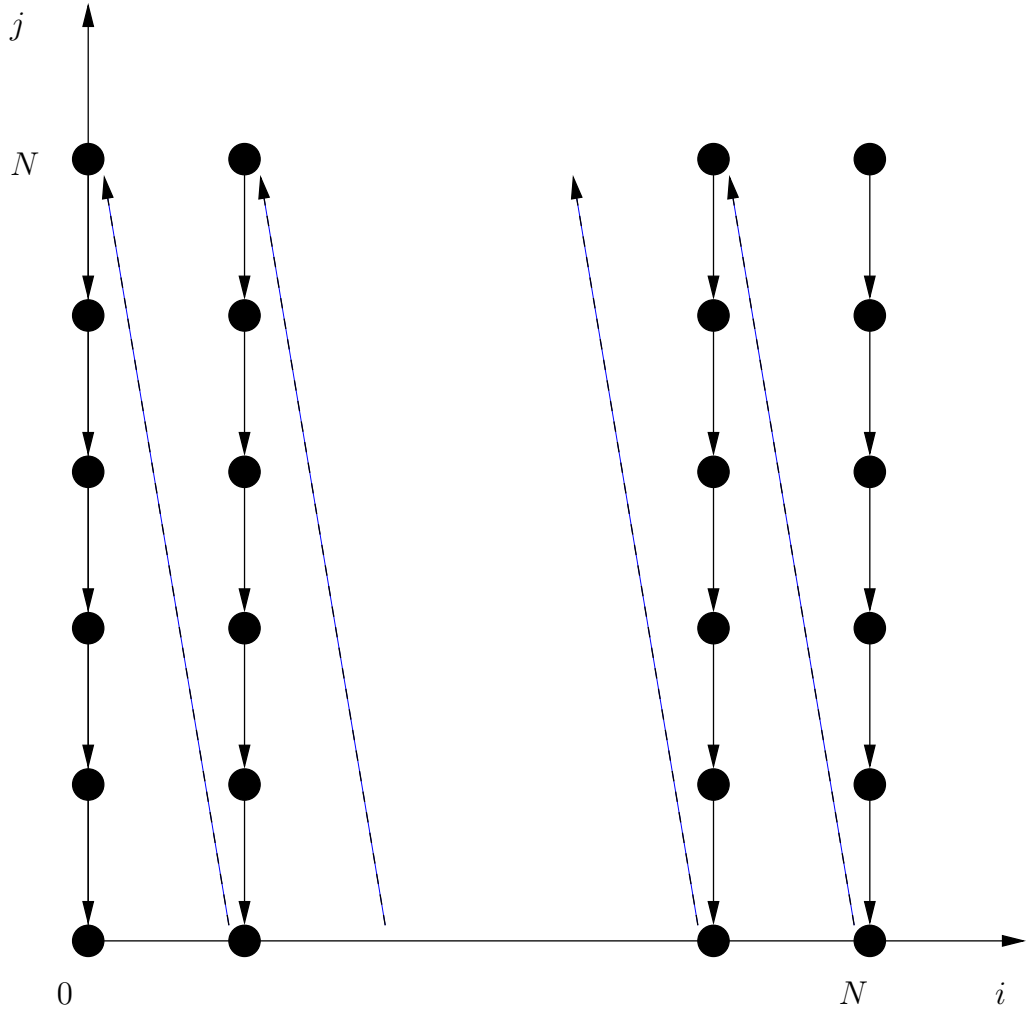
with $0 \leqslant \epsilon_t \leqslant 1$

And the **Objective function** :

Maximize $\sum_t \epsilon_t$

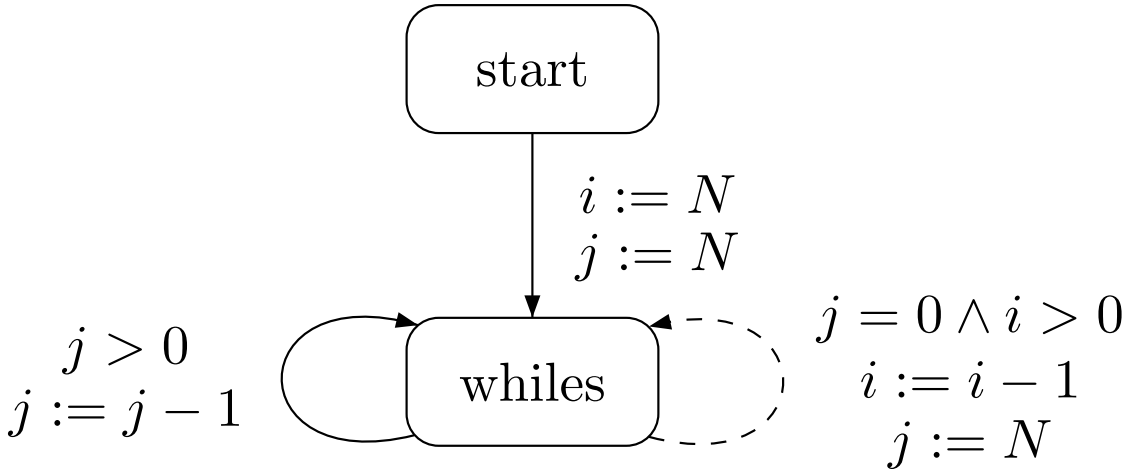
Example - 1

```
//N>0
i = N;
while(i>0)
{
    j = N;
    while(j>0) j--;
    i--;
}
```



Example - 2

```
//N>0
i = N;
while(i>0){
 j = N;
 while(j>0) j--;
 i--;
}
```

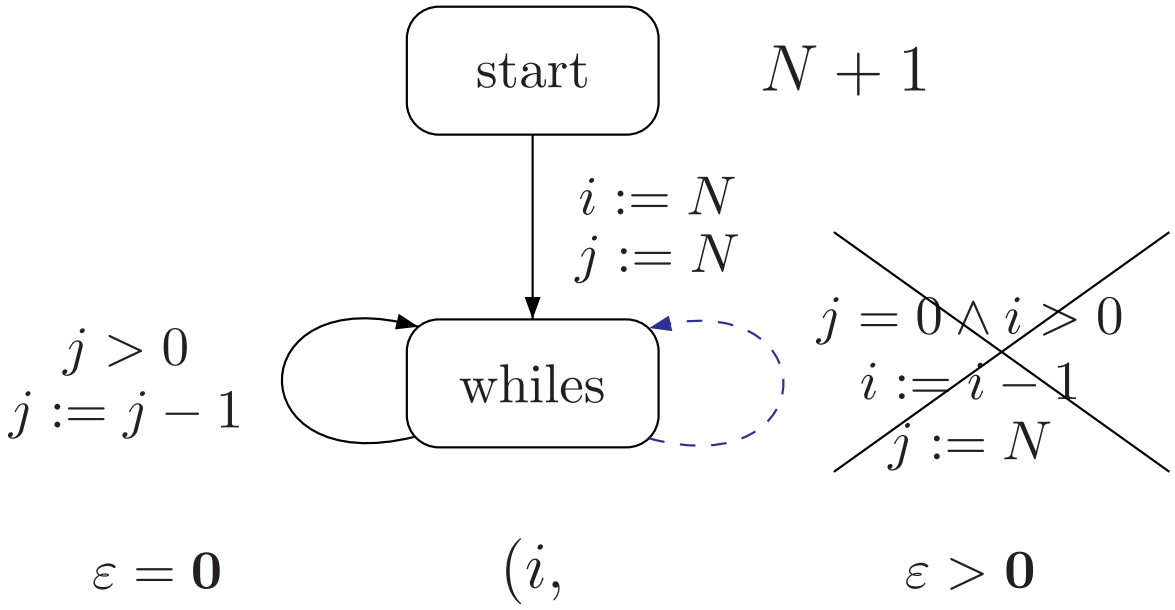


Invariant for `whiles` :

$$\mathbf{-1 < i \leqslant N, -1 < j \leqslant N}, N > 0, N = N_o$$

Example - 2

```
//N>0
i = N;
while(i>0){
 j = N;
 while(j>0) j--;
 i--;
}
```

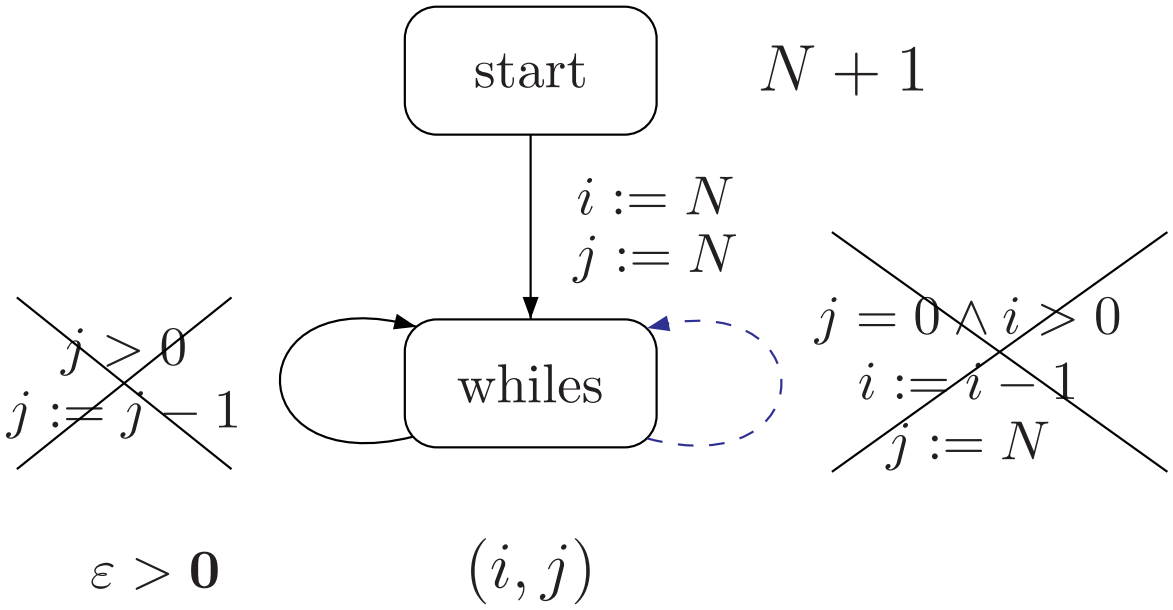


Invariant for `whiles` :

$$-\mathbf{1} < \mathbf{i} \leqslant \mathbf{N}, -\mathbf{1} < \mathbf{j} \leqslant \mathbf{N}, N > 0, N = N_o$$

Example - 2

```
//N>0
i = N;
while(i>0){
 j = N;
 while(j>0) j--;
 i--;
}
```

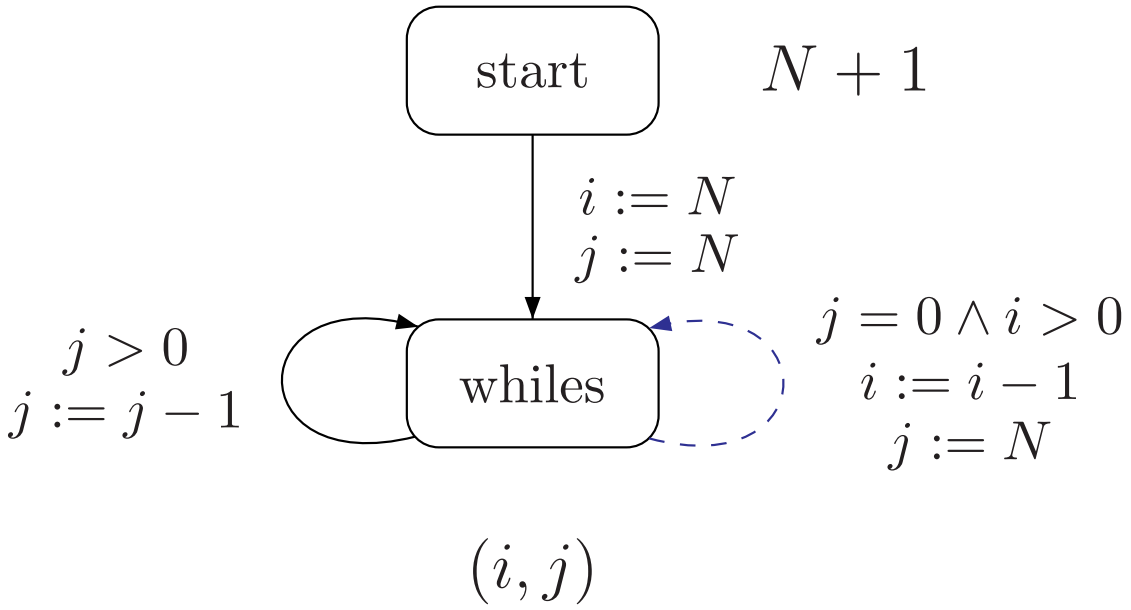


Invariant for `whiles` :

$$\mathbf{-1 < i \leqslant N, -1 < j \leqslant N}, N > 0, N = N_o$$

Example - 2

```
//N>0
i = N;
while(i>0){
 j = N;
 while(j>0) j--;
 i--;
}
```



Invariant for `whiles` :

$$\mathbf{-1 < i \leqslant N, -1 < j \leqslant N}, N > 0, N = N_o$$

An additional result !

Theorem (Completeness of greedy algorithm w.r.t. invariants)

If an affine interpreted automaton, with associated invariants, has a multi-dimensional affine ranking function, then the greedy algorithm generates one such ranking.
Moreover, the dimension of the generated ranking is minimal.

Summary of this part

From (arbitrary) flowchart programs :

- Compute an affine abstraction.
- Compute invariants on each control point.
- Compute and solve linear programming problems from the graph and its invariants.

Conference paper : [ADFG10]

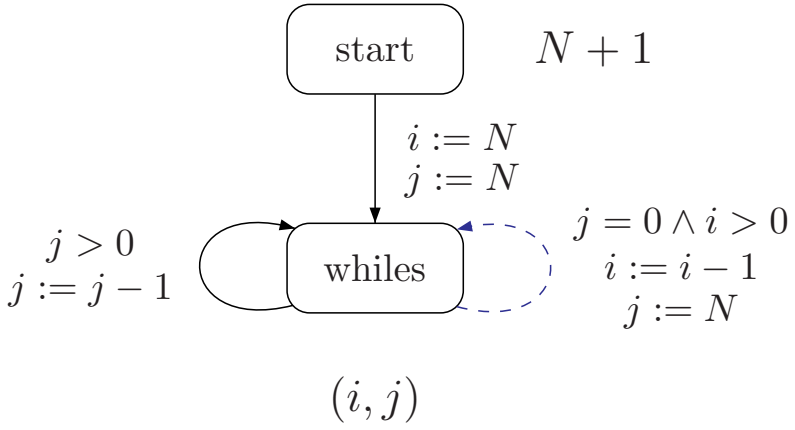
Bonus ! Computing a “WCET”

Worst-case computational complexity (WCCC) : maximum number of transitions fired by the automaton :

$$WCCC \leqslant card\Big(\bigcup_k \rho(k, P_k)\Big) \leqslant \sum_k card(\rho(k, P_k))$$

- Use **counting integer points** algorithms

$$\begin{aligned} WCCC &\leqslant \#\rho(\text{start}, P_{\text{start}}) \\ &\quad + \#\rho(\text{whiles}, P_{\text{whiles}}) \\ &= 1 + \#\{(i,j) \mid \ldots\} \\ &= N_0^2 + \ldots \end{aligned}$$



- Demo !

Our toolsuite “Rank”

1. C2FSM for the front-end
2. ASPIC for the invariants
3. RANK for the computation of the ranking function.

Published in [FG10] and [ADFG13].

Available for demo at the url :

`http://compsys-tools.ens-lyon.fr/`

Some experimental results

Sorting arrays :

Name	LOCs	Time(c2fsm/analysis) [1]	dim
selection	20	1.0/0.4	3
insertion	12	0.6/0.22	3
bubble	22	1.2/0.4	3
shell	23	1.0/1.1	4
heap	45	3.0/2.8	3

1. user time in seconds on a Pentium 2GHz with 1Gbyte RAM

Some comments on experimental results

- The algorithm works well on small challenging programs from the litterature.
- The form of the automaton has a strong impact on the invariants.
- The precision of invariants is **crucial**.

But the size of the LP instances grows exponentially and the solvers cannot deal with too much variables

```
ex2 : 10 loc / automaton : 10 vars, 5 transitions
-- > 3LP, average  180L/75 cols
heapsort : 30 loc / automaton : 12 vars, 10 transitions
--> fail.
```

Our algorithm

... does not **scale** !

2 ways of improvement

Two main directions of work :

- Divide and conquer : slice, cut, and go.
- Work on the 'practical' complexity of the initial algorithm.

Divide and conquer 1

Global idea

Work on smaller instances of programs.

"We already have quite a few people who know how to divide. So essentially, we're now looking for people who know how to conquer."

Divide and conquer 2

We use classical (static methods for safety) :

- slicing : we designed a specialized slicing for termination
- compute **context information**
- cut into **kernels** with preconditions
- prove termination on kernels.

▶ With C. Alias and G. Andrieu ([AAG12]

Divide and conquer 3

```
1 if(up == 0){
2   i = 1;
3   while(i <= n){
4     a[i] = a[i+n];
5     j = j+1;
6     i = i+1;
7   }
8 }
```

(a) Before slicing

```
1 if (up == 0) {
2   i = 1;
3   while(i <= n){
4     i = (i + 1);
5   }
6 }
```

(b) After slicing

Work on the initial algorithm 1/2

Even after slicing/summarizing all programs are not tractable with the first (monodimensional) algorithm.

- Idea 1 : work only on cutsets and on a compact version of the graph (Henry/Monniaux)

Work on the initial algorithm 1/2

Even after slicing/summarizing all programs are not tractable with the first (monodimensional) algorithm.

- Idea 1 : work only on cutsets and on a compact version of the graph (Henry/Monniaux)
- Idea 2 : Construct lazily the (dual) LP programs with counter examples computed with an SMT-solver. The size of LP programs does not depend on the complexity of the transitions.

▶ With D. Monniaux and L. Seguinot. Implementation in progress.

More details Appendix

Work on the initial algorithm 1/2

Even after slicing/summarizing all programs are not tractable with the first (monodimensional) algorithm.

- Idea 1 : work only on cutsets and on a compact version of the graph (Henry/Monniaux)
- Idea 2 : Construct incrementaly the (dual) LP programs with counter examples computed with an SMT-solver. The size of LP programs does not depend on the complexity of the transitions.

▶ With D. Monniaux and L. Seguinot. Implementation in progress.

More details Appendix

Work on the initial algorithm 2/2

First experimental results :

Example	Ranking function	Rank tool		Smterm			
		#LP	Avg. #lines/#cols	#LP	Avg. #lines/#cols	#SMT	Avg. size
$easy1$	$41 - x$	1	$334/155$	1	$3/6$	3	21
$easy2$	z	2	$86/42$	1	$3/5$	3	16
$wcet2$	$-11i - j + 65$	2	$225/94$	2	$4/6$	4	20
$exmini$	$102 - i - j + k$	2	$140/65$	3	$6/8$	5	16
$cousot9$	$\binom{i}{j}$	3	$180/75$	5	$6/8$	6	25

Conclusion 1/2

Biblio

An **excellent** survey on termination techniques [BAGM12]

Current / Future work :

- Implement and validate the SMT-based approach.
- Extract kernels from big programs and contribute to termination benchmarks.
- Non termination preconditions.
- Cooperation between techniques : computing invariants and proving termination at the same time ?

Conclusion 2/2

Termination/Compilation/Parallelism :

- A lot of undecidable (thus cool !) problems
- Reuse/extend models and algorithms from the literature
- Real-world- oriented research

Conclusion 2/2

Termination/Compilation/Parallelism :

- A lot of undecidable (thus cool !) problems
- Reuse/extend models and algorithms from the literature
- Benchmarks- oriented research

Termination ?

Synthetizing hardware : while2for transformation.

References I

Guillaume Andrieu, Christophe Alias, and Laure Gonnord, *SToP : Scalable Termination analysis of (C) Programs (tool presentation)*, Tapas 2012 (Deauville, France), December 2012.

C. Alias, A. Darte, P. Feautrier, and L. Gonnord, *Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs*, 17th International Static Analysis Symposium, SAS'10 (Perpignan, France), September 2010.

Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord, *Rank : a tool to check program termination and computational complexity*, Constraints in Software Testing Verification and Analysis (Luxembourg), March 2013.

Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud, *On the termination of integer loops*, ACM Trans. Program. Lang. Syst. **34** (2012), no. 4, 16.

Amir M. Ben-Amram and Chin Soon Lee, *Program termination analysis in polynomial time*, ACM Transactions on Programming Languages and Systems (TOPLAS) **29** (2007), no. 1, 5.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma, *Termination analysis of integer linear loops*, 16th International Conference on Concurrency Theory (CONCUR'05), LNCS, vol. 3653, Springer, August 2005, pp. 488–502.

Paul Feautrier and Laure Gonnord, *Accelerated Invariant Generation for C Programs with Aspic and C2fsm*, Tools for Automatic Program AnalysiS (TAPAS2010) (Perpignan, France), 2010 (English).

Sumit Gulwani, Sagar Jain, and Eric Koskinen, *Control-flow refinement and progress invariants for bound analysis*, SIGPLAN Not. **44** (2009), no. 6, 375–385.

References II

Sumit Gulwani and Florian Zuleger, *The reachability-bound problem*, ACM symposium on programming language design and implementation (PLDI), ACM, 2010, pp. 292–304.

Julien Henry, David Monniaux, and Matthieu Moy, *PAGAI : a path sensitive static analyzer*, Tools for Automatic Program AnalysiS (TAPAS 2012) (Deauville, France), September 2012 (Anglais).

Andreas Podelski and Andrey Rybalchenko, *A complete method for the synthesis of linear ranking functions*, in Steffen and Levi [SL04], pp. 239–251.

Bernhard Steffen and Giorgio Levi (eds.), *Verification, model checking, and abstract interpretation, 5th international conference, vmcai 2004, venice, january 11-13, 2004, proceedings*, LNCS, vol. 2937, Springer, 2004.


That's all Folks!