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 Raphaël-Ernani. Rodriguez (Univ Mineas Gerais Brasil).
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm

4. Implementation and Experimental results
Proving total correctness of programs

Total Correctness
Termination proofs, what for?

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

Termination proofs, what for?

- Physical environment
- Reactive system
- Inputs
- Outputs

(CC-BY-SA Captainm/Wikipedia)
Proving validity of reactive systems software

Termination proofs, what for?

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)
Termination proofs, what for?

But

Termination (HALTING PROBLEM) is undecidable!

- Use **conservative algorithms**: YES (+ witness) or “Don’t Know” (+ potential infinite path)
- On **restricted** classes of programs.
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm

4. Implementation and Experimental results
Hoare rule [1969] for total correctness

Partial correctness:

\[
\{ P \text{ and } B \} \ S \ \{ P \} \n\]

\[
\{ P \} \textbf{while } B \textbf{ do } S \ \textbf{done} \ \{ \text{not}(B) \text{ and } P \} \n\]
Termination proofs, how?

Hoare rule [1969] for total correctness

Total correctness:

\[
\{ t=z \text{ and } t \in D \text{ and } P \text{ and } B \} \ S \ \{ P \text{ and } t < z \text{ and } t \in D \} \quad (D, <) \text{ well-founded}
\]

\[
\{ P \} \ \textbf{while } B \ \textbf{do } S \ \textbf{done} \ \{ \neg(B) \text{ and } P \}\]
Hoare rule [1969] for total correctness

Total correctness:

\[
\{ t=z \text{ and } t \in D \text{ and } P \text{ and } B \} \ S \ \{ P \text{ and } t < z \text{ and } t \in D \} \quad (D, <) \text{ well-founded}
\]

\[
\{P\} \ \textbf{while} \ B \ \textbf{do} \ S \ \textbf{done} \ \{\text{not}(B) \text{ and } P\}
\]

➤ Find \((D, <)\) and \(t\)!
First easy example

```plaintext
assume(N>0);
i=N;
while(i>0) --i;
```
First easy example

\[
\begin{align*}
\text{assume}(N>0); \\
i &= N; \\
\text{while}(i>0) &\quad --i;
\end{align*}
\]

\( (\mathbb{N}, <) \) 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
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.
Termination proofs, how?

Agenda

- A (conservative) algorithm to find affine ranking functions.
- Scalability issues and other improvements.
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm
   - Model of programs
   - From automata to invariants
   - An algorithm to compute 1D affine functions
   - An algorithm for multidimensional ranking functions

4. Implementation and Experimental results
Summary

(C) Program → Model of program → Affine ranking function
Summary

(C) Program $\rightarrow$ Model of program $\rightarrow$ Affine ranking function
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm
   - Model of programs
   - From automata to invariants
   - An algorithm to compute 1D affine functions
   - An algorithm for multidimensional ranking functions

4. Implementation and Experimental results
Objective

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

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

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

▶ **Pb** Compute an accurate (affine) model of a given C program?
Our model for programs

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

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

\[
\begin{align*}
    t_1 : & \quad \frac{N \geq 0}{i := N} \\
    t_2 : & \quad \frac{i \geq 1}{j := N} \\
    t_3 : & \quad \frac{j \geq 1}{j := j - 1} \\
    t_4 : & \quad \frac{j = 0}{i := i - 1}
\end{align*}
\]
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;
```

Diagram:
- Start
- True
- \( i := N \)
- \( 1 \leq i \)
- \( i := i-1 \)
- \( i \leq 0 \)
- Stop
From a Program to an affine Automaton I

Program $\rightarrow$ Control flow graph is the first step of \textit{compilation}.

Program:

\begin{verbatim}
assume(N>0);
i=N;
while(i>0) --i;
\end{verbatim}

Basics: tests $\rightarrow$ branches; for/while $\rightarrow$ loops.
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 : \((\mathbb{N}^d, \prec_{lex})\)

\[ \rho(k, \vec{x}) = A_k \cdot \vec{x} + \vec{b}_k \]

//N>0
i = N;
while(i>0)
{
    j = N;
    while(j>0) j--;
    i--;
}
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm
   - Model of programs
   - From automata to invariants
   - An algorithm to compute 1D affine functions
   - An algorithm for multidimensional ranking functions

4. Implementation and Experimental results
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.

\[ t_1 : \frac{N \geq 0}{i := N} \]
\[ t_2 : \frac{i \geq 1}{j := N} \]
\[ t_3 : \frac{j \geq 1}{j := j - 1} \]
\[ t_4 : \frac{j = 0}{i := i - 1} \]

\[ k_1 : N \geq 0 \land i \geq 0 \land i \leq N \]
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:

![Diagram of polyhedron](image)

Tools: ASPIC([FG10]) or the promising Pagai (J. Henry, [HMM12])
From an affine Automaton to invariants

Abstract Interpretation in a nutshell:

System of equations:

\[
\begin{align*}
P_{init} &= all \\
P_{k_1} &= t_1(P_{init}) \cup t_4(P_{k_2}) \\
P_{k_2} &= t_2(P_{k_1}) \cup t_3(P_{k_2})
\end{align*}
\]
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:

- Affine (idealized) world!
- Real World is not affine!

Affine Invariant
What’s next?
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm
   - Model of programs
   - From automata to invariants
   - An algorithm to compute 1D affine functions
   - An algorithm for multidimensional ranking functions

4. Implementation and Experimental results
The basic algorithm
An algorithm to compute 1D affine functions

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

\[ \rho(start, \vec{x}) = \alpha_{start, 1} \cdot i + \alpha_{start, 2} \cdot N \]
\[ + \alpha_{start, 3} \cdot i_0 + \alpha_{start, 4} \cdot N_0 \]
\[ + \alpha_{start, 5} \]

\[ \rho(W, \vec{x}) = \alpha_{W, 1} \cdot i + \ldots \]

\[ \rho(stop, \vec{x}) = \alpha_{stop, 1} \cdot i + \ldots \]
Finding a 1D-ranking function as an affine form

Assume \(N > 0\);
\[ i := N; \]
while \((i > 0)\) --\(i\);

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

\[
\begin{align*}
\rho(\text{start}, \vec{x}) &= \alpha_{\text{start},1}.i + \alpha_{\text{start},2}.N \\
&+ \alpha_{\text{start},3}.i_0 + \alpha_{\text{start},4}.N_0 \\
&+ \alpha_{\text{start},5} \\
\rho(W, \vec{x}) &= \alpha_{W,1}.i + \ldots \\
\rho(\text{stop}, \vec{x}) &= \alpha_{\text{stop},1}.i + \ldots
\end{align*}
\]

The constraints are:

- For each control point: \(\rho(pc, \vec{x}) \geq 0\) 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

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

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

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

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

\[ \rho(pc, \vec{x}) \geq 0 \text{ on } P_W \text{ gives (control point } W) : \]

\[ \forall i, N \in P_W, \alpha_{W,1} i + \ldots \geq 0 \]

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

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

Unkowns are \( \alpha_{*,*} \). “Forall” in (possibly) \textbf{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: \( \rho(p_c, \vec{x}) \geq 0 \) on \( P_{pc} \).

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

\[
\rho(W, \vec{x}) = \lambda_{W,1}.(N_0 - 1) + \lambda_{W,2}.(N_0 - N) \\
+ \lambda_{W,3}.(N - N_0) + \lambda_{W,4}.i + \lambda_{W,3}.(N - i)
\]
Finding a 1D ranking function: linearization

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

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

$$\rho(W, \bar{x}) = \lambda_{W,1}(N_0 - 1) + \lambda_{W,2}(N_0 - N)$$
$$+ \lambda_{W,3}(N - N_0) + \lambda_{W,4}i + \lambda_{W,3}(N - i)$$

We were looking for $\rho(W, \bar{x})$ with the following “template”:

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

- Identifying coefficients for $i$: $\alpha_{W,1} = \lambda_{W,4} - \lambda_{W,3}, \ldots$
Finding a 1D ranking function: linearization

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

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

\[
\rho(W, \vec{x}) = \lambda_{W,1}.(N_0 - 1) + \lambda_{W,2}.(N_0 - N) \\
+ \lambda_{W,3}.(N - N_0) + \lambda_{W,4}.i + \lambda_{W,3}.(N - i)
\]

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

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

- **Identifying coefficients for \( i \):** \( \alpha_{W,1} = \lambda_{W,4} - \lambda_{W,3}, \ldots \)
- **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.
The basic algorithm

An algorithm to compute 1D affine functions

Finding a 1D ranking function - linearization and solving

2- Decreasing transitions:

\[(x' - x) \in t \Rightarrow \rho(dest, x') - \rho(src, x') > 0\]

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.
Termination proofs, what for?

Termination proofs, how?

The basic algorithm
- Model of programs
- From automata to invariants
- An algorithm to compute 1D affine functions
- An algorithm for multidimensional ranking functions

Implementation and Experimental results
The main idea

Idea

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

\[
\rho = \begin{pmatrix}
\rho_1 \\
\rho_2 \\
\vdots \\
\rho_d
\end{pmatrix}
\]
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 \( \sigma \), 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 \( \sigma \), not increasing for any transition, and decreasing for as many transitions as possible.
  - Let \( \rho_i = \sigma ; i = i + 1 \); \((i^{th} \text{ dimension})\)
  - 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:

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

\(\rightarrow\)

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

with \(0 \leq \epsilon_t \leq 1\)
Modification of the constraint system

Pb. How do we implement “decreasing for as many transitions as possible” in the LP instance?

Decreasing transitions constraints:

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

\[\rightarrow\]

\[(\vec{x}' - \vec{x}) \in t \Rightarrow \rho(\text{dest}, \vec{x}') - \rho(\text{src}, \vec{x}') \geq \epsilon_t\]

with \(0 \leq \epsilon_t \leq 1\)

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:

-1 < i \leq N, -1 < j \leq N, N > 0, N = N_o
Example - 2

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

Invariant for `whiles`:

\[-1 < i \leq N, -1 < j \leq N, N > 0, N = N_o\]
Example - 2

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

Invariant for `whiles`:

\[-1 < i \leq N, -1 < j \leq N, N > 0, N = N_0\]
Example - 2

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

Invariant for whiles :

\[-1 < i \leq N, -1 < j \leq N, N > 0, N = N_0\]
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 :

\[ WCCC \leq \text{card} \left( \bigcup_{k} \rho(k, P_k) \right) \leq \sum_{k} \text{card}(\rho(k, P_k)) \]

- Use counting integer points algorithms

\[ WCCC \leq \#\rho(\text{start}, P_{\text{start}}) + \#\rho(\text{whiles}, P_{\text{whiles}}) \]

\[ = 1 + \#\{(i, j) | \ldots\} \]

\[ = N_0^2 + \ldots \]

- Demo !
1. Termination proofs, what for?
2. Termination proofs, how?
3. The basic algorithm
4. Implementation and Experimental results
   - First experimental results
   - Scalability issues
1. Termination proofs, what for?
2. Termination proofs, how?
3. The basic algorithm
4. Implementation and Experimental results
   - First experimental results
   - Scalability issues
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>LOCs</th>
<th>Time (c2fsm/analysis)</th>
<th>dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>20</td>
<td>1.0/0.4</td>
<td>3</td>
</tr>
<tr>
<td>insertion</td>
<td>12</td>
<td>0.6/0.22</td>
<td>3</td>
</tr>
<tr>
<td>bubble</td>
<td>22</td>
<td>1.2/0.4</td>
<td>3</td>
</tr>
<tr>
<td>shell</td>
<td>23</td>
<td>1.0/1.1</td>
<td>4</td>
</tr>
<tr>
<td>heap</td>
<td>45</td>
<td>3.0/2.8</td>
<td>3</td>
</tr>
</tbody>
</table>

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 literature.
- 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**!
1. Termination proofs, what for?

2. Termination proofs, how?

3. The basic algorithm

4. Implementation and Experimental results
   - First experimental results
   - Scalability issues
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.
Global idea

Work on smaller instances of programs.
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

(a) Before slicing

```java
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 }
```

(b) After slicing

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

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 incrementally 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.

More details ▶ Appendix
Work on the initial algorithm 2/2

First experimental results:

<table>
<thead>
<tr>
<th>Example</th>
<th>Ranking function</th>
<th>Rank tool</th>
<th>Smterm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#LP</td>
<td>Avg. #lines/#cols</td>
<td>#LP</td>
</tr>
<tr>
<td>easy1</td>
<td>41 - x</td>
<td>1</td>
<td>334/155</td>
</tr>
<tr>
<td>easy2</td>
<td>z</td>
<td>2</td>
<td>86/42</td>
</tr>
<tr>
<td>wcet2</td>
<td>-11i - j + 65</td>
<td>2</td>
<td>225/94</td>
</tr>
<tr>
<td>exmini</td>
<td>102 - i - j + k</td>
<td>2</td>
<td>140/65</td>
</tr>
<tr>
<td>cousot9</td>
<td>( \binom{i}{j} )</td>
<td>3</td>
<td>180/75</td>
</tr>
</tbody>
</table>
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.
Guillaume Andrieu, Christophe Alias, and Laure Gonnord, *SToP: Scalable Termination analysis of (C) Programs (tool presentation)*, Tapas 2012 (Deauville, France), December 2012.


