Program verification

Proving Termination of flowcharts programs

Laure Gonnord and David Monniaux

University of Lyon / LIP

October 20th, 2015

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

Plan

Plan

Goal : Safety

Prove that (some) loops terminate:

```
int main () {
    unsigned int i,j ;
    i=42 ; j=1515 ;
    while(i>0)i-- ; ✔
    while(j>=0)j++ ; ✘
}
```

- Fight against bugs.

Goal : Optimisation

Prove that (some) loops terminate:

```
int main () {
    unsigned int i,j ;
    i=42 ; j=1515 ;
    while(i>0) i-- ; ✔
    foo(j);
}
```

- Code motion (compiler optimisation).

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 :

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

Hoare rule [1969] for total correctness

Total correctness :

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

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

First easy example

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

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

Restriction

In this course, we will 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.

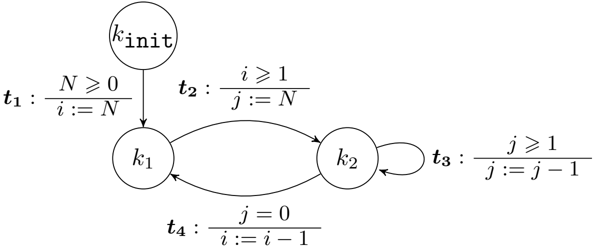
Agenda

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

Our model for programs

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

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



C program to automata + invariants

Preprocessing :

- Compilation + abstraction of non numerical behaviors. Not trivial.
- We also compute numerical invariants (polyhedra on each control point) (see course 3 on invariant generation).

Two papers

- SAS 2010 : Alias, Darte, Feautrier, Gonnord.
- PLDI 2015 : Gonnord, Monniaux, Radanne.

Plan

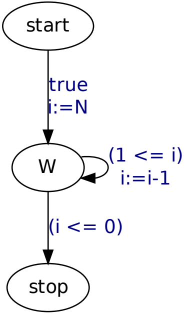
Termination for affine automata (I)

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: $(\mathbb{N}, <)$



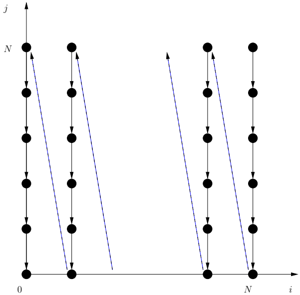
$$\rho(st, i, N) = \begin{cases} 2 + N_0 & \text{if } st = start \\ i + 1 & \text{if } st = W \\ 0 & \text{if } st = stop \end{cases}$$

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--;
}
```



Introduction

Problem statement

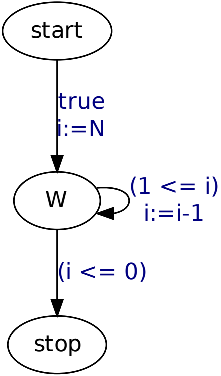
Given:

- 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;
```

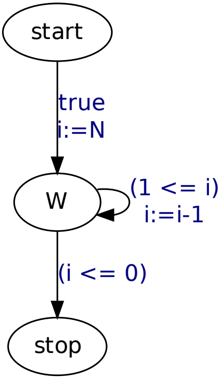


Searching for $\alpha_{pc,-} \in \mathbb{Q}$:

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

Finding a 1D-ranking function as an affine form

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



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

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

The constraints are :

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

Arglll, “forall” constraints

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

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

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

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

Arglll, "forall" constraints

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

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

$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x}) > 0$ 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** : $\rho(pc, \vec{x}) \geq 0$ on P_{pc}.

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

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

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 :

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

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 ***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 :

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

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 ***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** :

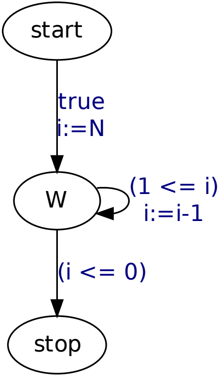
$$(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$$

also gives affine constraints.

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

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 :

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

$\rightarrow$

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

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 :

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

$\rightarrow$

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

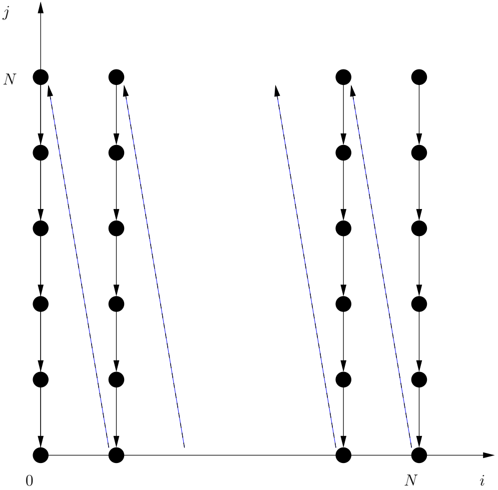
with $0 \leq \epsilon_t \leq 1$

And the **Objective function**:

$$\text{Maximize } \sum_t \epsilon_t$$

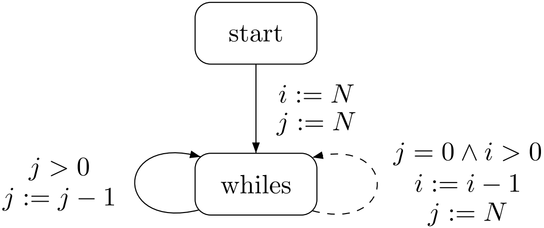
Example - 1

```
//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--;
}
```

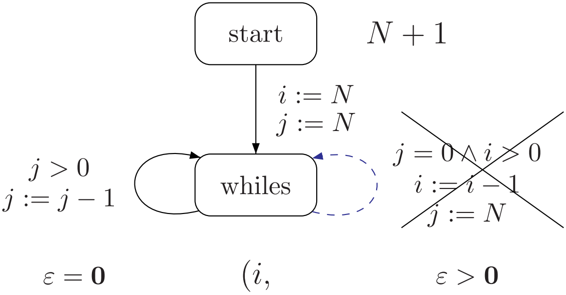


Invariant for `whiles` :

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

Example - 2

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

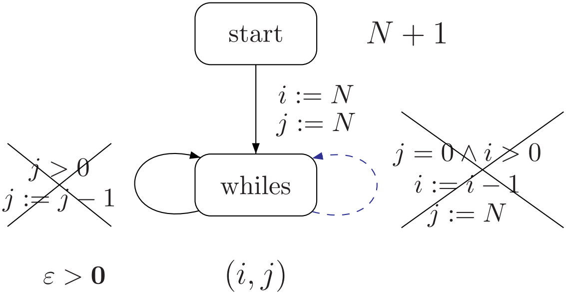


Invariant for `whiles` :

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

Example - 2

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

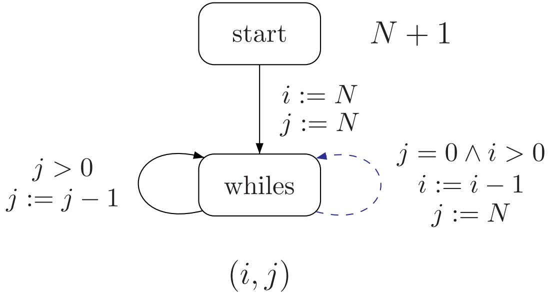


Invariant for `whiles` :

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

Example - 2

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



Invariant for `whiles` :

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

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.

Bonus ! Computing a "WCET"

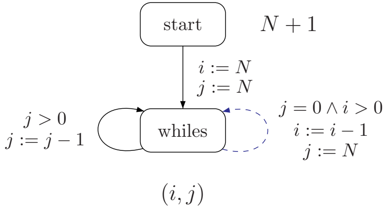
Worst-case computational complexity (WCCC): maximum number of transitions fired by the automaton:

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

- Use **counting integer points** algorithms

$$\begin{aligned} WCCC \leq\ & \#\rho(\text{start}, P_{\text{start}}) \\ & + \#\rho(\text{whiles}, P_{\text{whiles}}) \end{aligned}$$

$$\begin{aligned} &= 1 + \#\{(i,j) \mid \ldots\} \\ &= N_0^2 + \ldots \end{aligned}$$



Our toolsuite "Rank"

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

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.

Plan

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

Global idea

Work on smaller instances of programs.

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 [Stop tool].

Work on the initial algorithm

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.

- These ideas lead to PLDI'15 and the tool Termite

Plan

Contributions of the PLDI paper

- A technique to prove that (some) loops terminate:
  - Automatic generation of **ranking functions**
  - Based on Linear Programming.
  - Focus on scalability: incremental construction of LP instances.
- Implemented as a standalone tool: TERMITE
  - Capable of proving 119 on 129 programs of TERMCOMP benchmark.
  - Competitive with other state-of-the-art tools.
  - Publicly available on github.

Proving termination: ranking functions

Non strict Linear ranking function

- Non increasing along the transitions
- Positive
- Linear

Strict linear ranking function: decreasing by ≥ 1.

```
int main () {
    unsigned int i,j ;
    i=42 ; j=1515 ;
    while(i>0)i-- ;
}
```

$$\rho = i + 1$$

Existing techniques: drawbacks / solutions

Existing techniques: build a system of constraints and solve:

$$Size = O(\#vars \times \#Bblocks \times \#transitions)$$

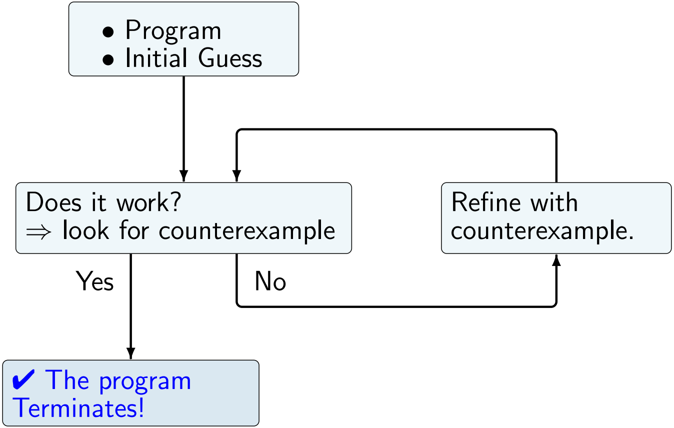
- scalability: all basic blocks $\rightsquigarrow$ big constraint systems
- precision: ρ must decrease at **each** transition.

Our technique:

- only considers **a cut-set** of basic blocks.
- considers loops as single transitions.

▶ **We do not compute all paths** explicitly (CEGAR-based algorithm).

Our key insight : incremental generation of constraints

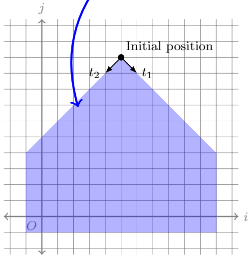


Sub-problem

Given a single loop $\tau = t_1 \vee t_2$:

$$\boldsymbol{t_2} : \frac{0 \leq i \wedge 0 \leq j}{\begin{array}{c} i := i - 1 \\ j := j - 1 \end{array}} \qquad k_0 \qquad \boldsymbol{t_1} : \frac{i \leq 10 \wedge 0 \leq j}{\begin{array}{c} i := i + 1 \\ j := j - 1 \end{array}}$$

$+$ an **invariant** $\mathcal{I}$, compute $\rho = \lambda \mathbf{x} + \ell$ an affine function:



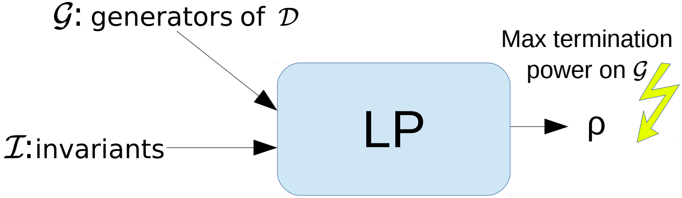
- Positive on $\mathcal{I}$.
- Decreasing on τ.

w.l.o.g we suppose $i_0 = 10, j_0 = 15$

$\mathbf{x} = \begin{pmatrix} i \\ j \end{pmatrix}$ is the vector of variables.

Solving the problem

Thanks to linearity + Farkas' Lemma we are able to define:



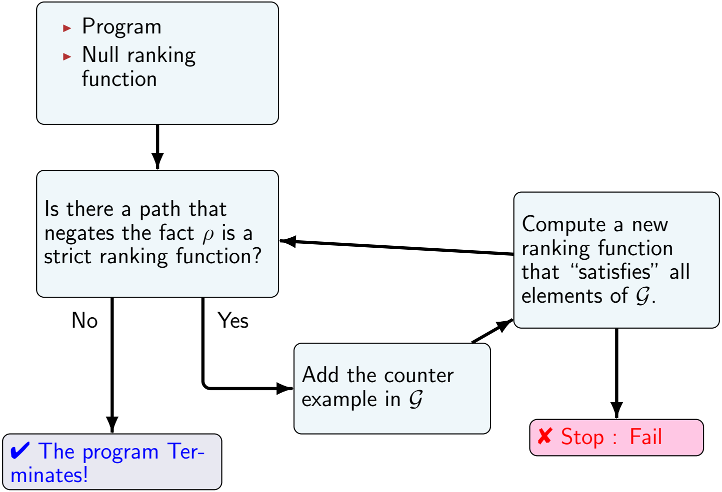
with $\mathcal{D}$ the set of reachable one-step differences:

$$\mathcal{D} = \{\mathbf{x} - \mathbf{x}' \mid \mathbf{x}, \mathbf{x}' \in \mathcal{I}, (\mathbf{x}, \mathbf{x}') \in \tau\}$$

- ρ positive, decreasing on $\mathcal{G}$, and **stricly decreasing on a maximal subset of $\mathcal{G}$**

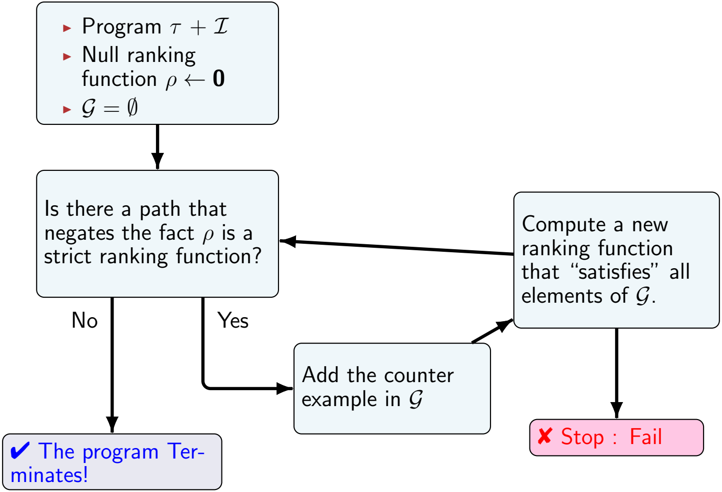
Simple algorithm for one control point

Idea : we construct $\mathcal{G}$ lazily.



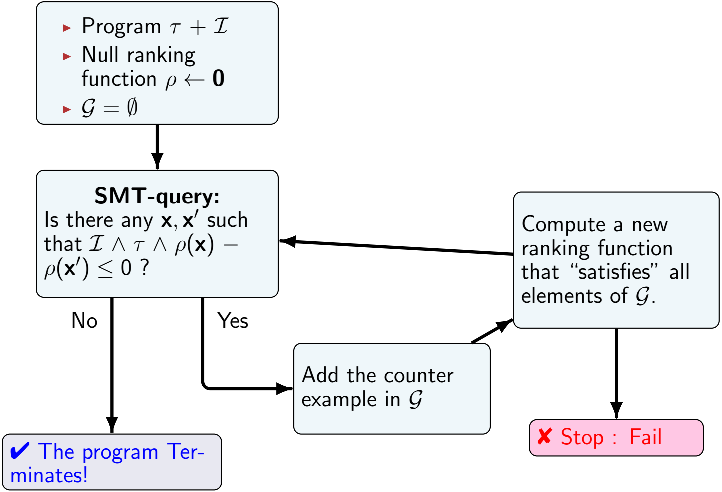
Simple algorithm for one control point

Idea : we construct $\mathcal{G}$ lazily.



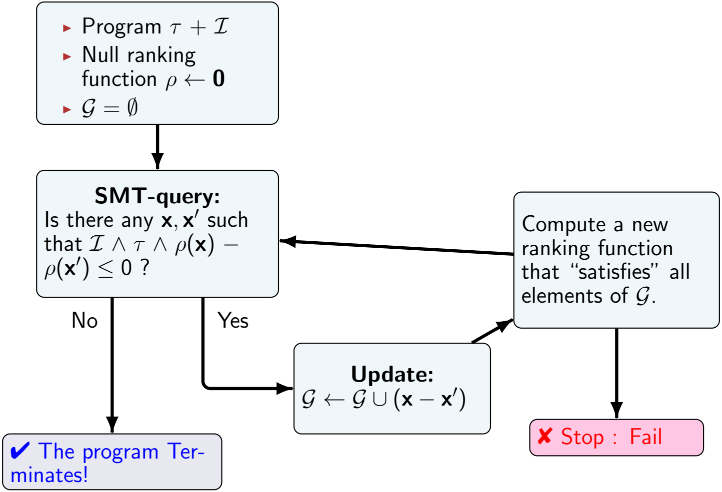
Simple algorithm for one control point

Idea : we construct $\mathcal{G}$ lazily.



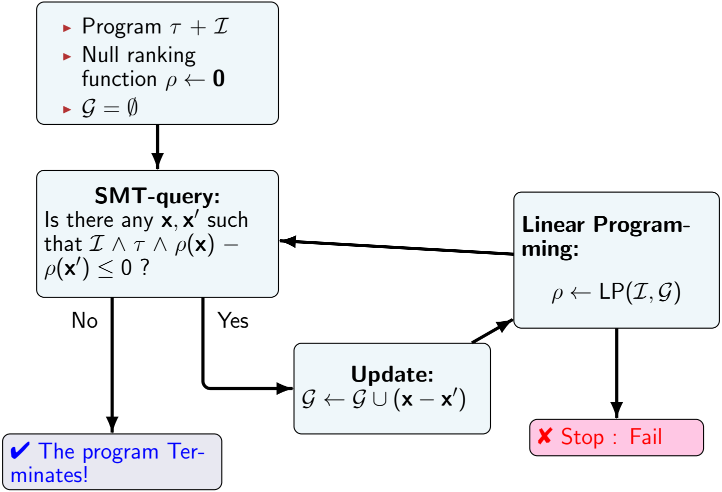
Simple algorithm for one control point

Idea : we construct $\mathcal{G}$ lazily.

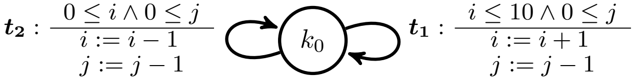


Simple algorithm for one control point

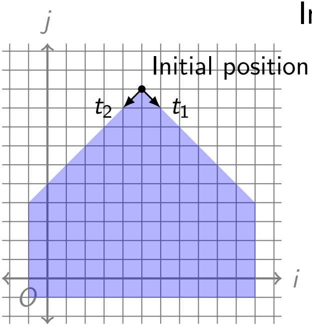
Idea : we construct $\mathcal{G}$ lazily.



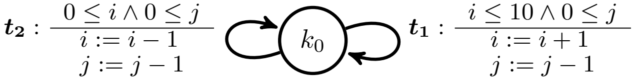
On the example

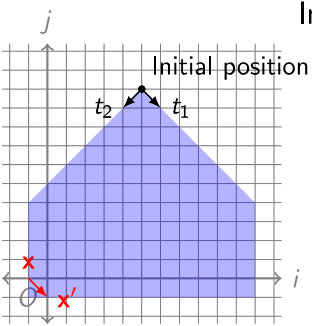


Initial $\rho \leftarrow \mathbf{0}$:



On the example



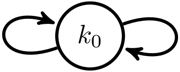


Initial $\rho \leftarrow \mathbf{0}$:

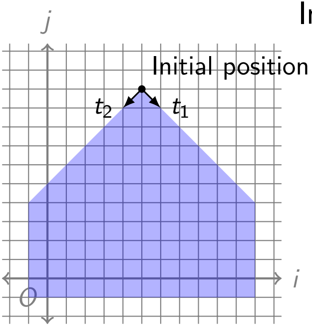
- **SMT**
  $(i, j) = (-1, 0), (i', j') = (0, -1)$ is a counterex for ρ (from t_1)
  $\rightsquigarrow \mathcal{G} \leftarrow \{(-1, 1)\}$.

On the example

$$t_2 : \frac{0 \le i \wedge 0 \le j}{\begin{array}{c} i := i - 1 \\ j := j - 1 \end{array}}$$



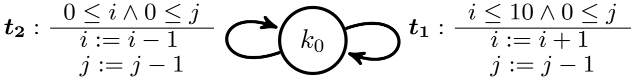
$$t_1 : \frac{i \le 10 \wedge 0 \le j}{\begin{array}{c} i := i + 1 \\ j := j - 1 \end{array}}$$

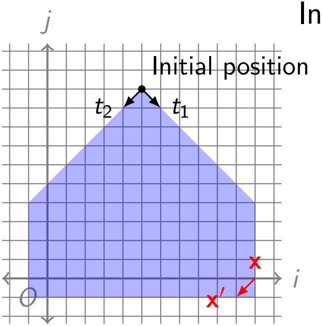


Initial $\rho \leftarrow \mathbf{0}$:

- **SMT**
  $(i, j) = (-1, 0), (i', j') = (0, -1)$ is a counterex for ρ (from t_1)
  $\rightsquigarrow \mathcal{G} \leftarrow \{(-1, 1)\}$.
- **LP** new candidate: $\rho = 11 - i$.

On the example

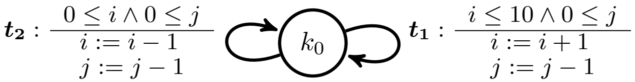


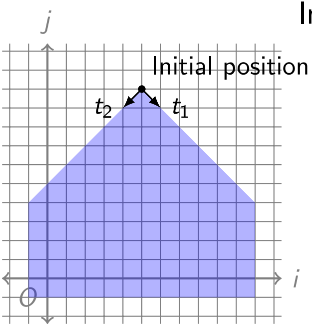


Initial $\rho \leftarrow \mathbf{0}$:

- **SMT**
  $(i, j) = (-1, 0), (i', j') = (0, -1)$ is a counterex for ρ (from t_1)
  $\rightsquigarrow \mathcal{G} \leftarrow \{(-1, 1)\}$.
- **LP** new candidate: $\rho = 11 - i$.
- **SMT**
  $(i, j) = (11, 0), i', j' = (10, -1)$ is a counterex for ρ (from t_2)
  $\rightsquigarrow \mathcal{G} \leftarrow \mathcal{G} \cup \{(1, 1)\}$.

On the example



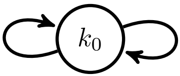


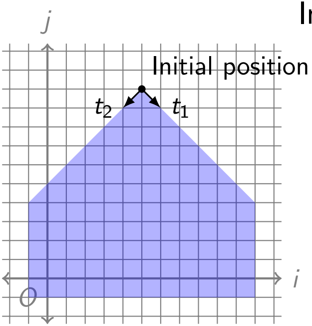
Initial $\rho \leftarrow \mathbf{0}$:

- **SMT**
  $(i, j) = (-1, 0), (i', j') = (0, -1)$ is a counterex for ρ (from t_1)
  $\rightsquigarrow \mathcal{G} \leftarrow \{(-1, 1)\}$.
- **LP** new candidate: $\rho = 11 - i$.
- **SMT**
  $(i, j) = (11, 0), i', j' = (10, -1)$ is a counterex for ρ (from t_2)
  $\rightsquigarrow \mathcal{G} \leftarrow \mathcal{G} \cup \{(1, 1)\}$.
- **LP** new candidate: $\rho = j + 1$.

On the example

$$t_2 : \frac{0 \leq i \wedge 0 \leq j}{\begin{array}{c} i := i - 1 \\ j := j - 1 \end{array}} \qquad t_1 : \frac{i \leq 10 \wedge 0 \leq j}{\begin{array}{c} i := i + 1 \\ j := j - 1 \end{array}}$$





Initial $\rho \leftarrow \mathbf{0}$:

- **SMT** $(i, j) = (-1, 0), (i', j') = (0, -1)$ is a counterex for ρ (from t_1) $\rightsquigarrow \mathcal{G} \leftarrow \{(-1, 1)\}$.
- **LP** new candidate: $\rho = 11 - i$.
- **SMT** $(i, j) = (11, 0), i', j' = (10, -1)$ is a counterex for ρ (from t_2) $\rightsquigarrow \mathcal{G} \leftarrow \mathcal{G} \cup \{(1, 1)\}$.
- **LP** new candidate: $\rho = j + 1$.
- **SMT** UNSAT ! $\rho = j + 1$ **is strict**

A major issue!

This algorithm **doesn't terminate** in general:

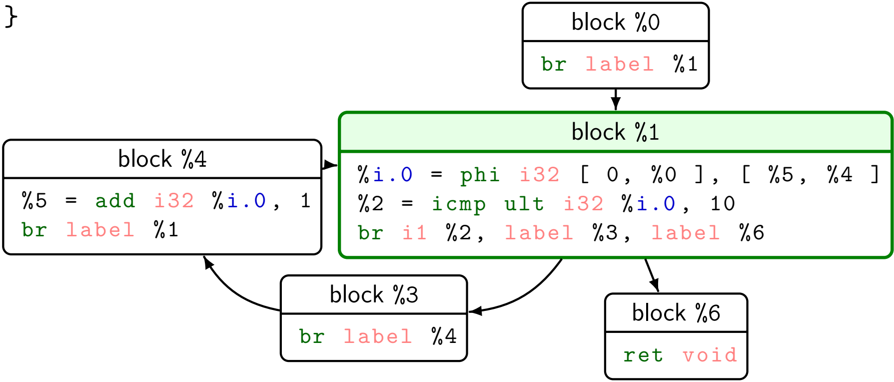
- The set of counter examples can be infinite.
- If there is no strict ranking function.

Fix: limit the search area for the counterexample $u = \mathbf{x} - \mathbf{x}'$

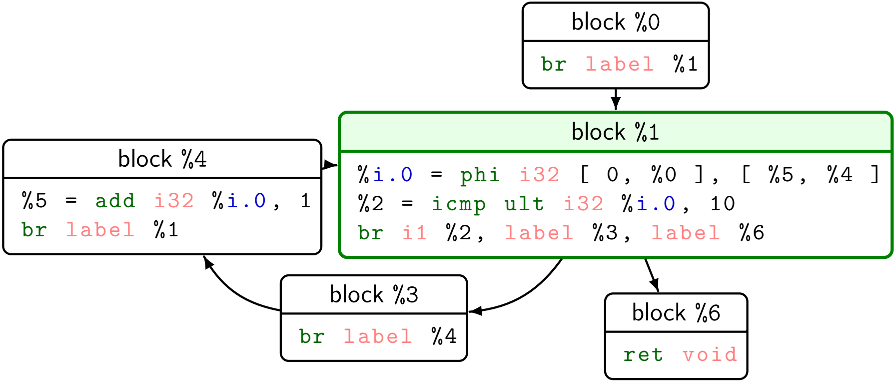
- impose counterexamples to be in the boundary of $\mathcal{D}$ (max-SMT).
- always **improve** the ranking or quit.

Control flow graph and LLVM representation

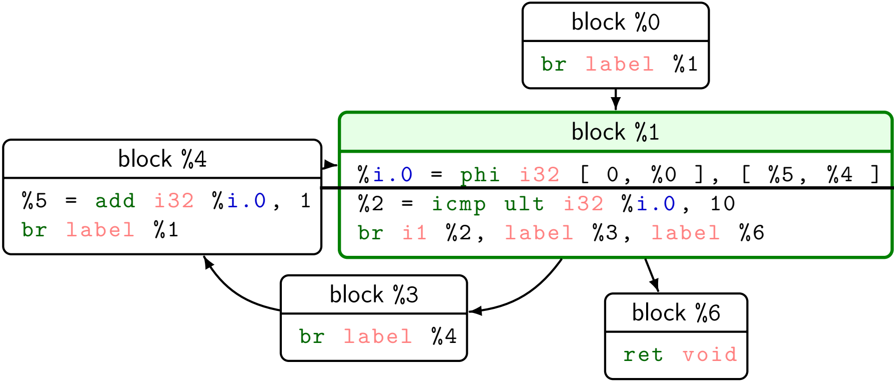
```
void simple_loop_constant() {
  for(unsigned i=0; i<10; i++) {
    // Do nothing
  }
}
```



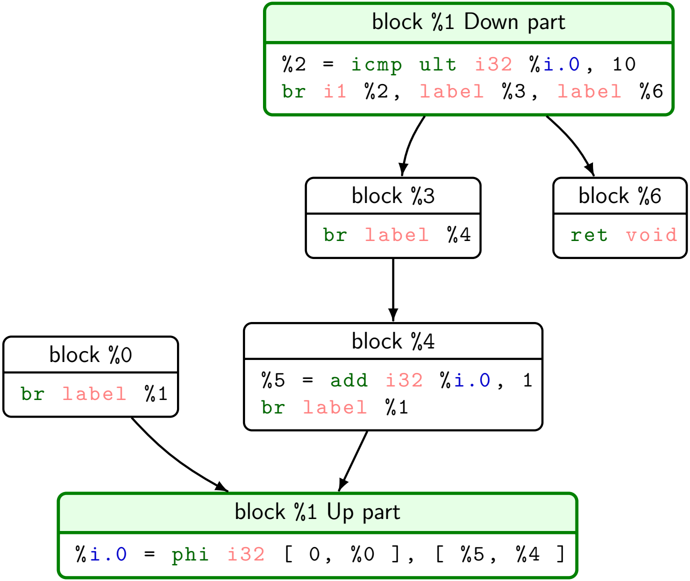
SMT encoding for control-flow-graph (LLVM2SMT)



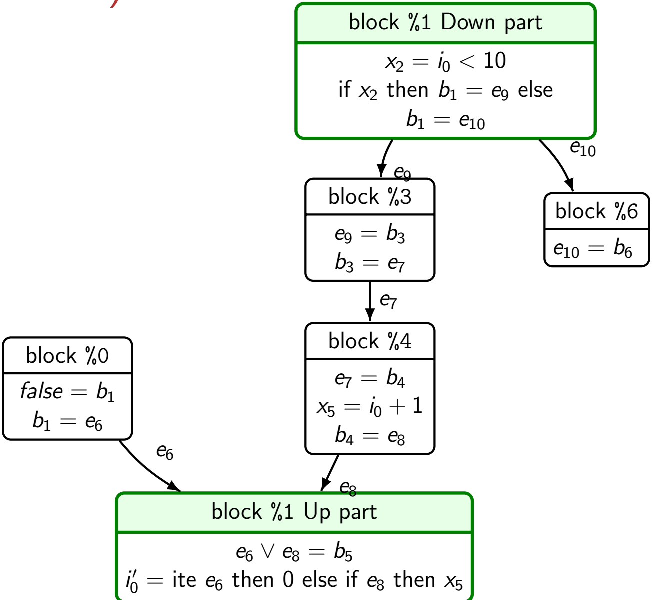
SMT encoding for control-flow-graph (LLVM2SMT)



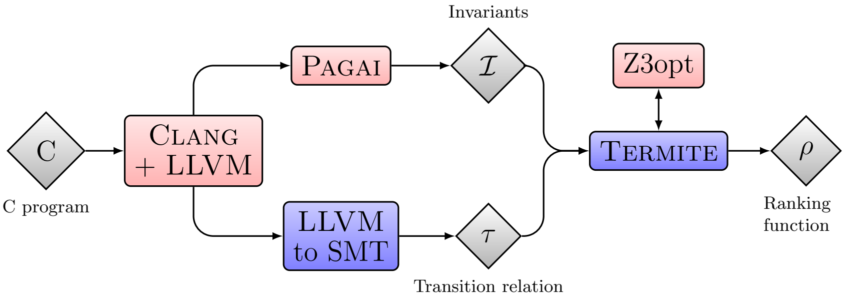
SMT encoding for control-flow-graph (LLVM2SMT)



SMT encoding for control-flow-graph (LLVM2SMT)



Software architecture



http://termite-analyser.github.io/

Experimental setup

- **Benchmarks**: TermComp + others
- **Machine:** Intel(R) Xeon(R) @ 2.00GHz 20MB Cache.
- **Other tools:** (Rank), Aprove, Bchi Ultimate, Loopus.

- Issue : various front-ends / invariant generators

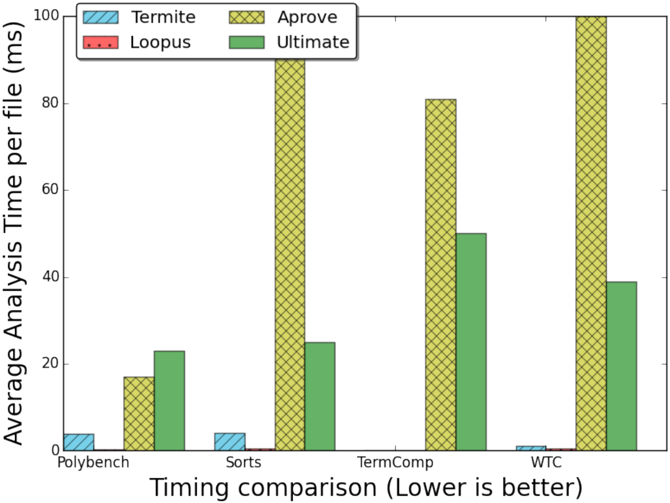
Comparison : Linear Programming instances sizes

On WTC benchmark (average per file):

Tool	#lines (constraints)	#columns (variables)
Rank	584	229
Termite	5	2

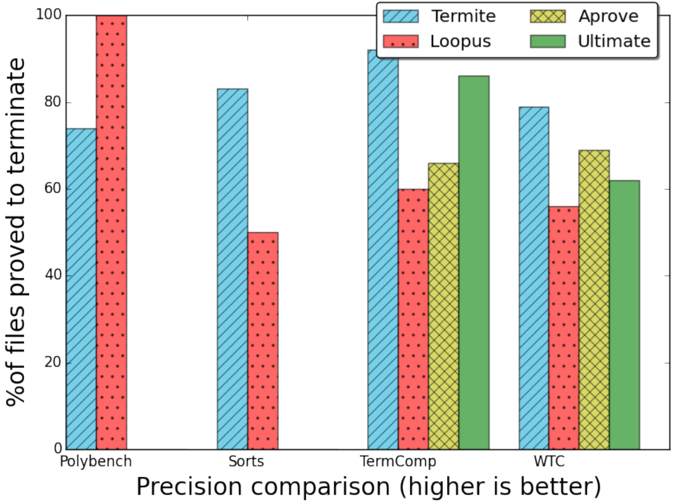
Rank is the termination tool from [Alias et al, SAS 2010]

Timing Comparison



Timings exclude the front-end for TERMITE and LOOPUS.

Precision Comparison



In the paper

- The complete method: multidimensional algorithm, multi control points.
- Correctness, Complexity.
- Experimental evaluation.

Summary

- A complete method to synthetise multidimensional ranking functions
- Based on large block encoding + counter-example based linear programming instance generation.
- Experiments show great results!

- `http://termite-analyser.github.io/`

Future Work

- Use the technique to also compute $\mathcal{I}$.
- Conditional termination.
- Quantifier elimination.