# Synthesis of ranking functions using extremal counterexamples

**Laure Gonnord** David MONNIAUX Gabriel RADANNE

June, 17th 2015 - PLDI


# Goal : Safety

Prove that (some) loops terminate :

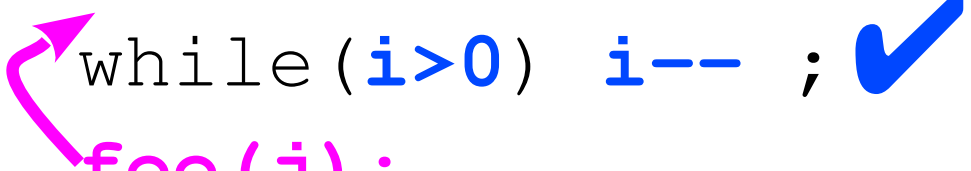
```
int main () {
    unsigned int i,j ;
    i=42 ; j=1515 ;
    while(i>0)i-- ;
    while(j>=0)j++ ;
}
```

- 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).

# Contributions

- 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.

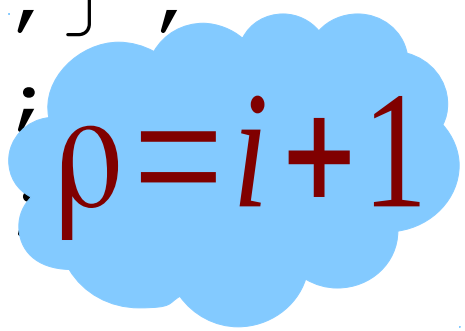
# Proving termination : ranking functions

**Non strict** Linear ranking function

- Non increasing along the transitions
- Positive
- Linear

**Strict** linear ranking function : decreasing by $\geqslant 1$.

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



# Existing techniques : drawbacks / solutions

Existing techniques : build a system of constraints and solve :

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

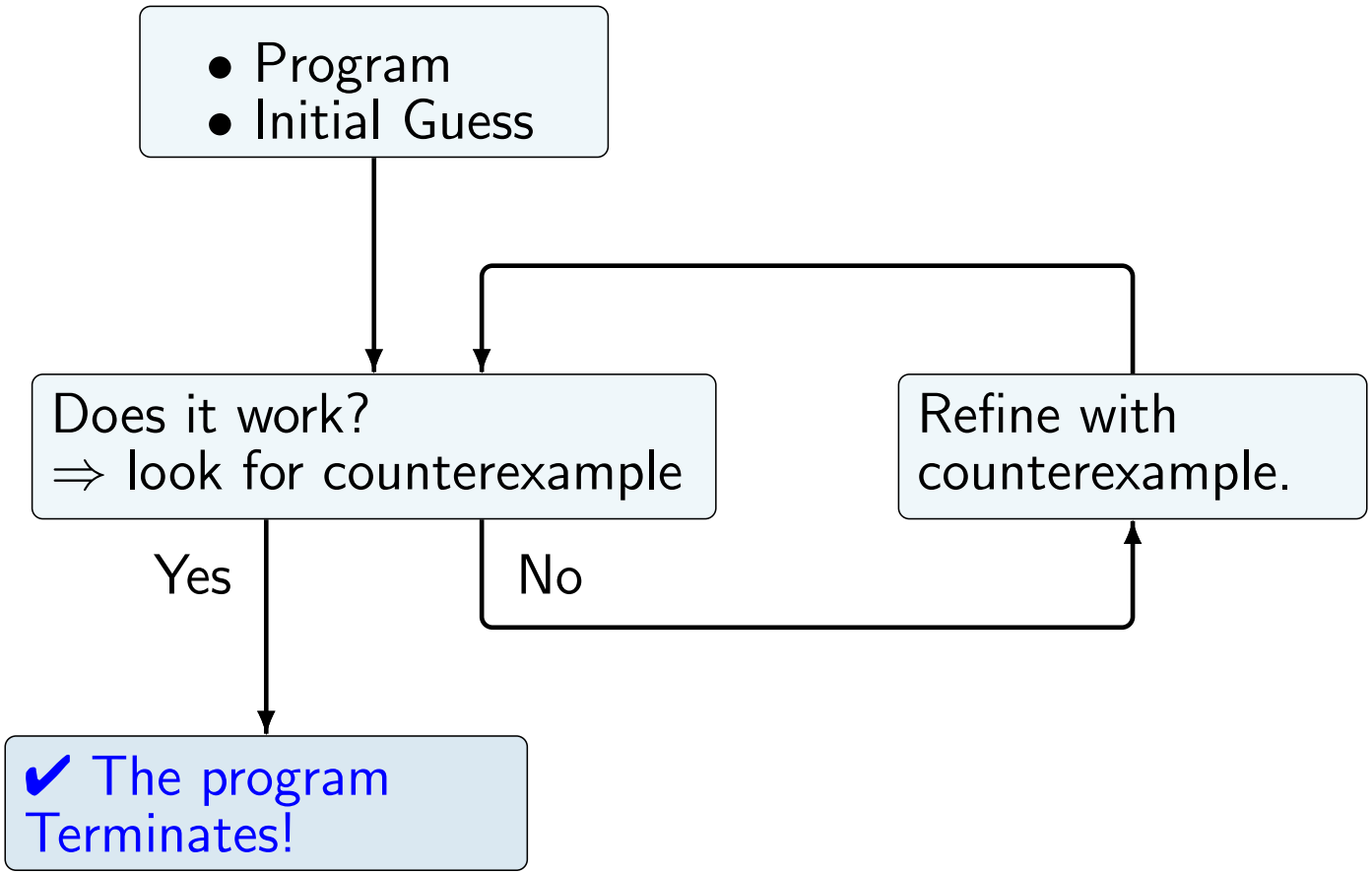
- scalability : all basic blocks $\rightsquigarrow$ big constraint systems
- precision : $\rho$ must decrease at **each** transition.

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

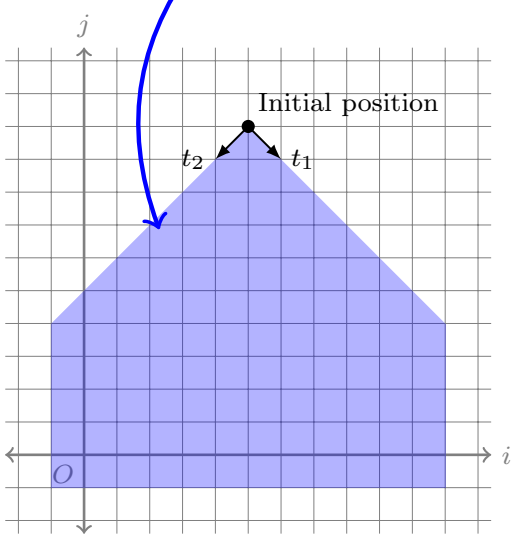


# Sub-problem

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 $\rho = \lambda \boldsymbol{x} + \ell$ an affine function :



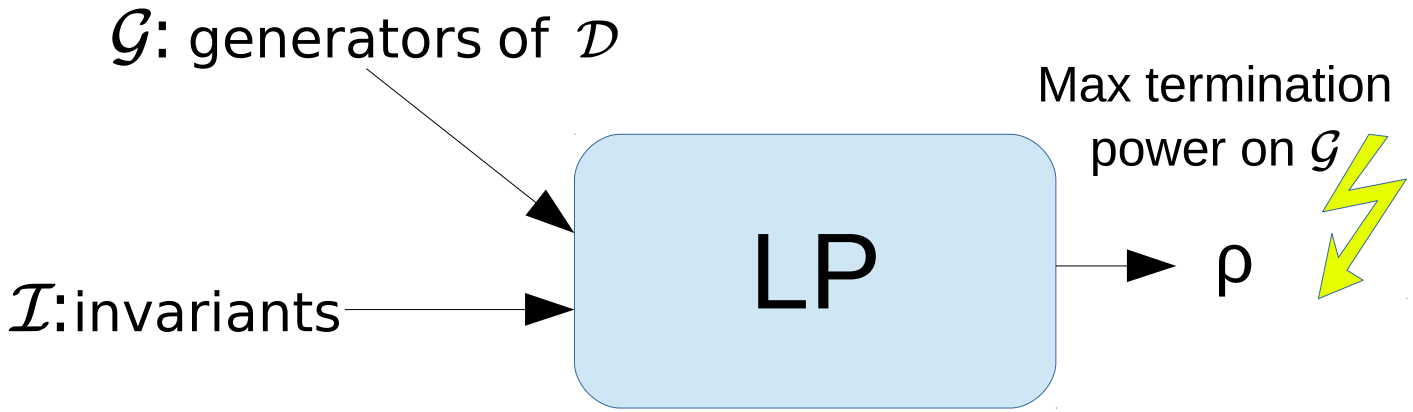

- Positive on $\mathcal{I}$.
- Decreasing on $\tau$.

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

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

# Solving the problem

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



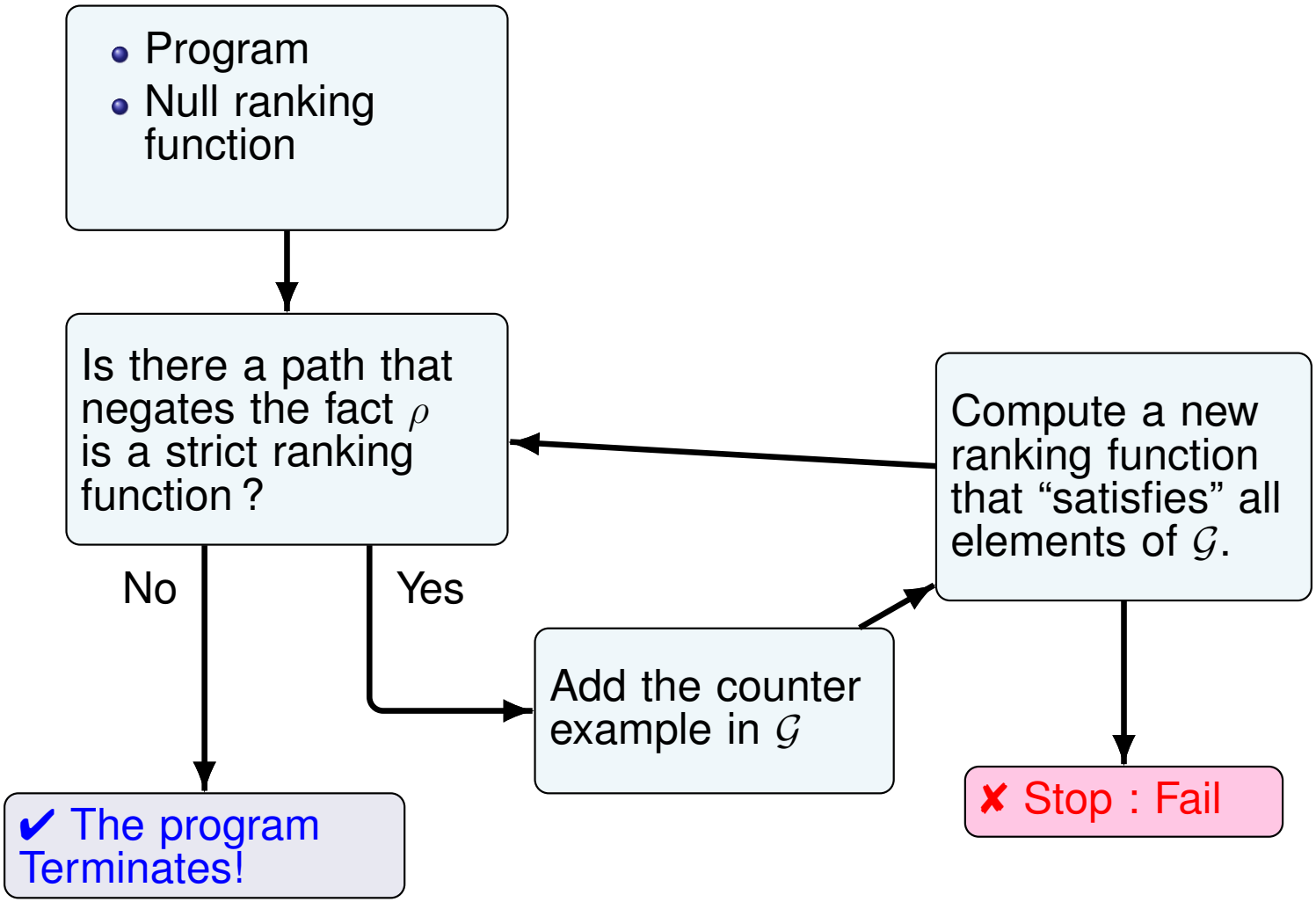

with $\mathcal{D}$ the set of reachable one-step differences :

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

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

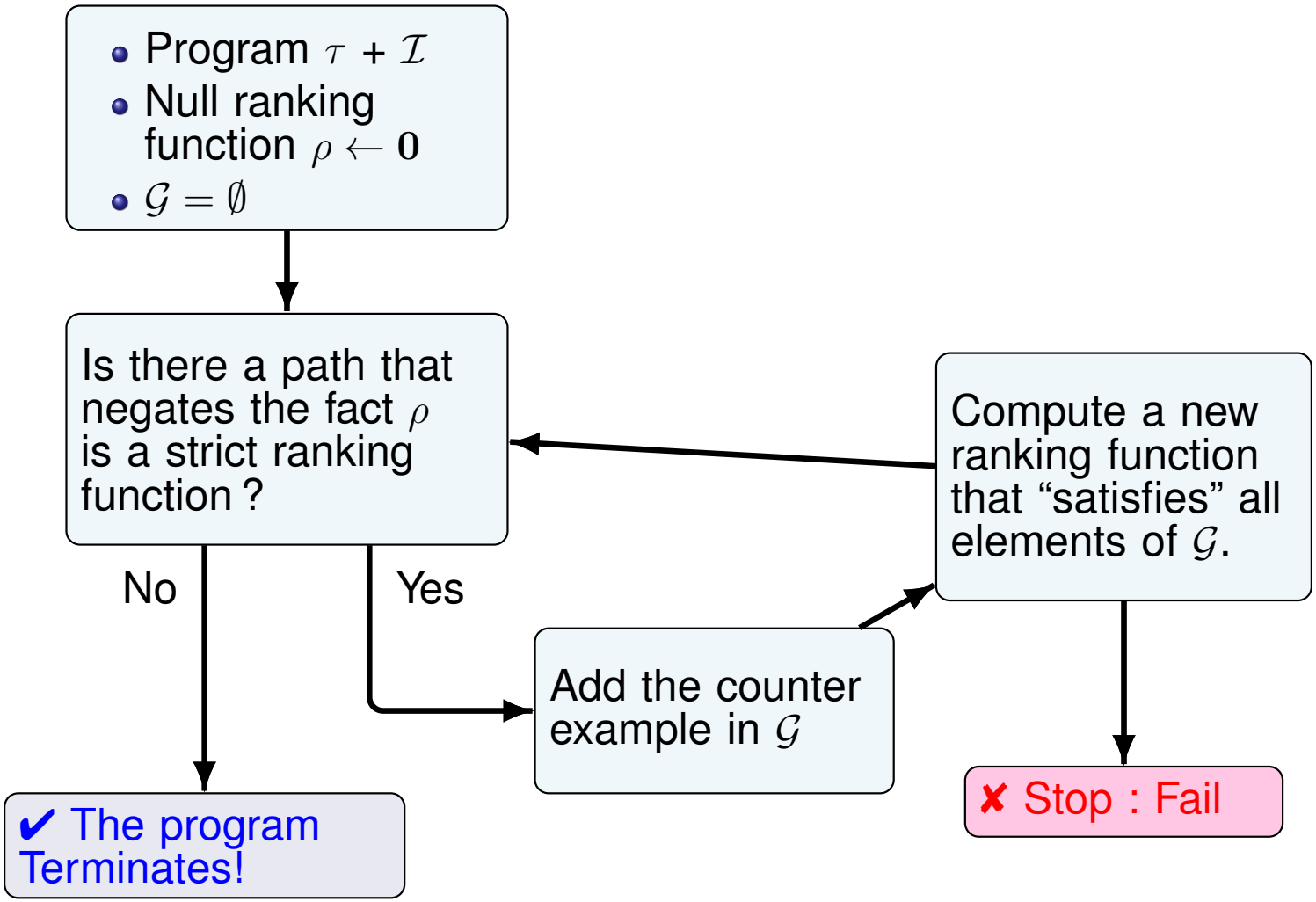
# Simple algorithm for one control point

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



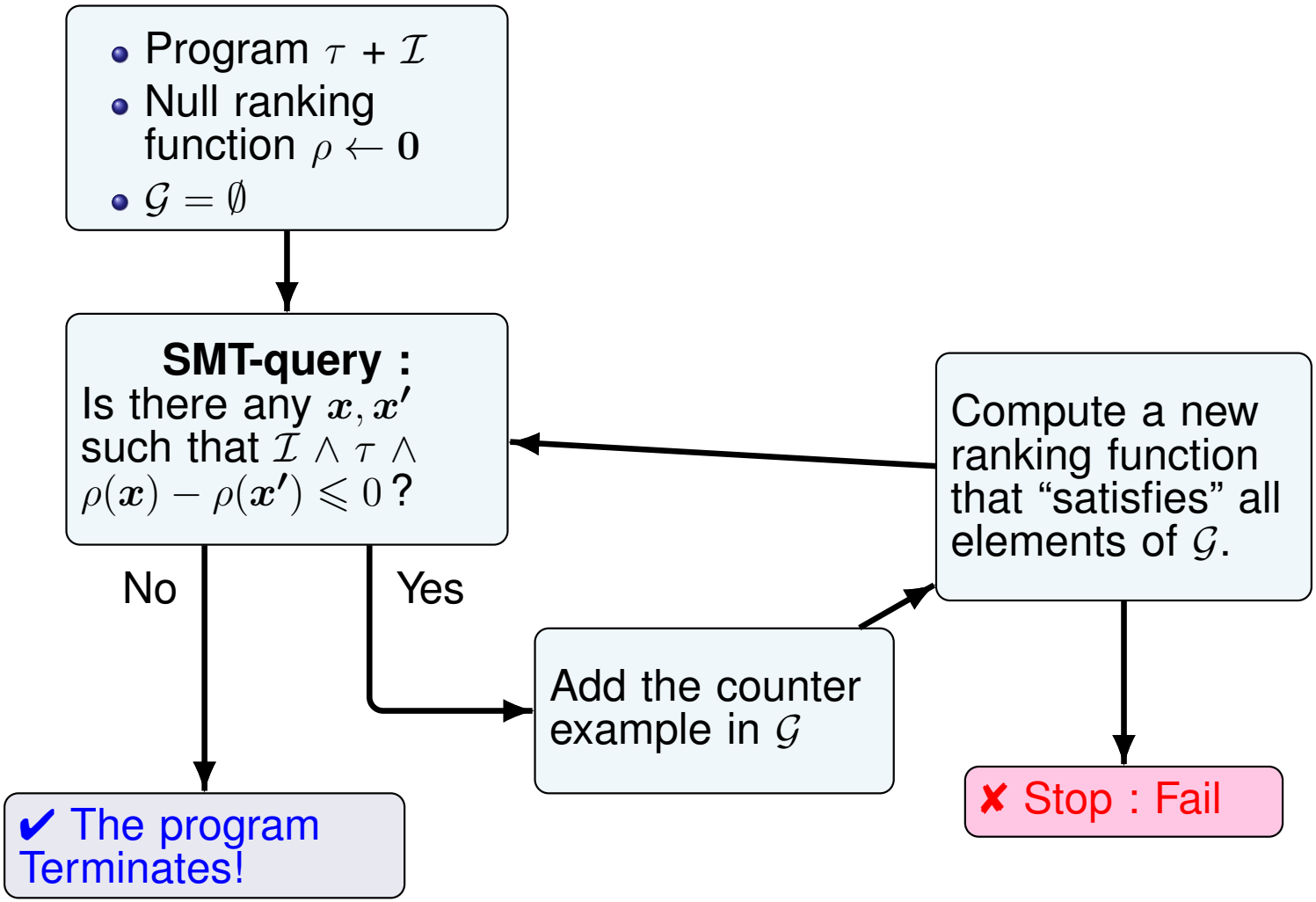
# Simple algorithm for one control point

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



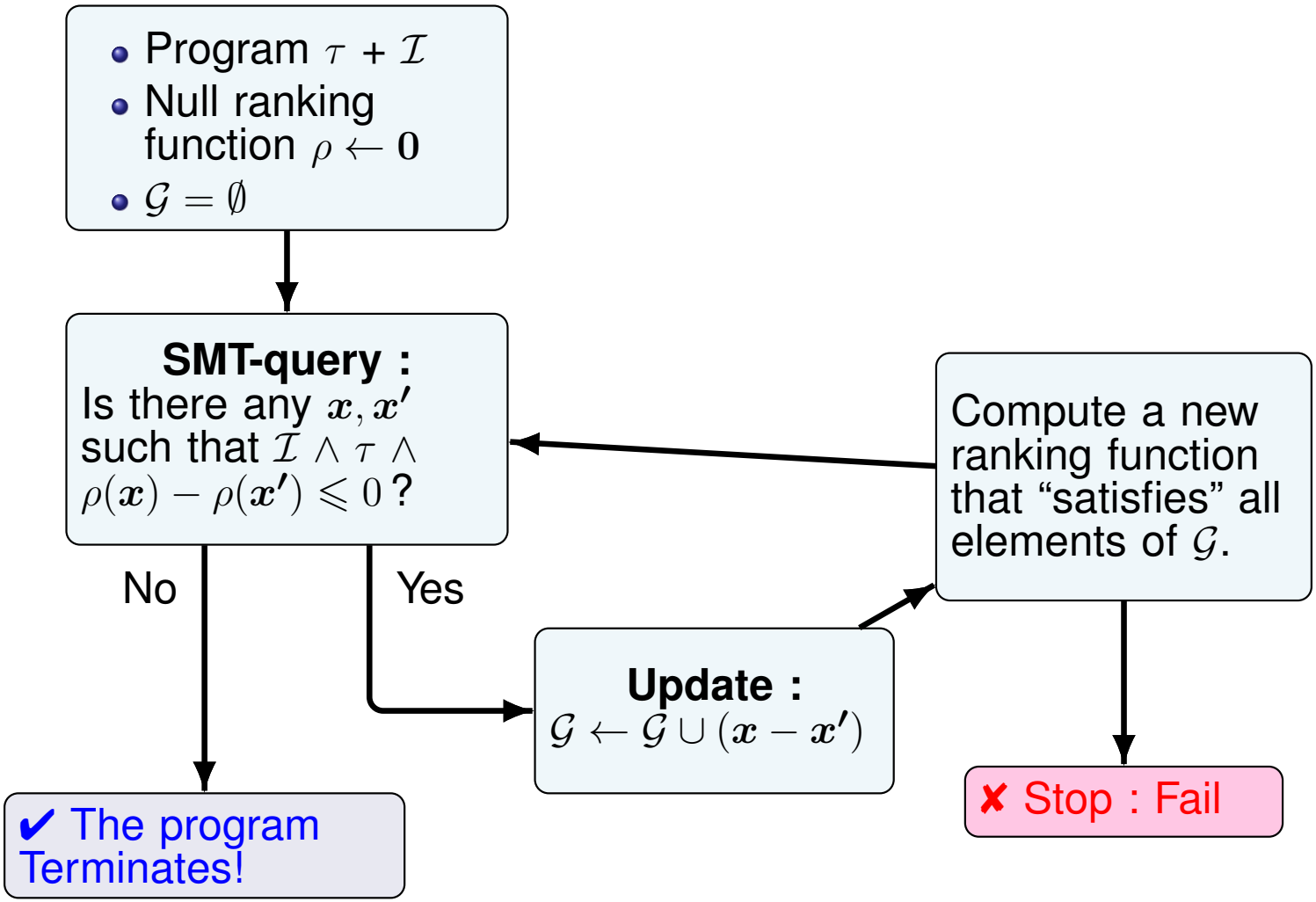
# Simple algorithm for one control point

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



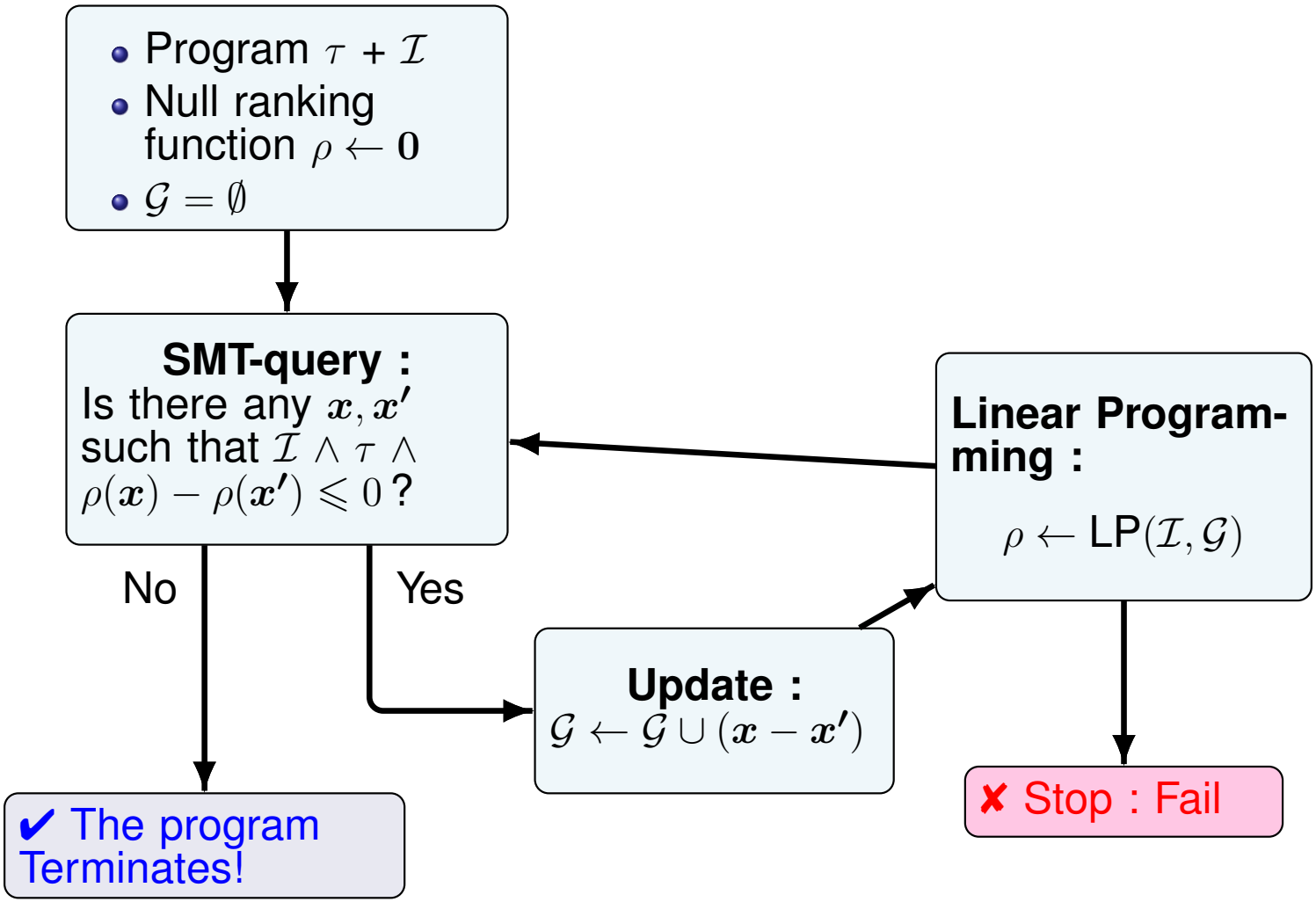
# Simple algorithm for one control point

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



# Simple algorithm for one control point

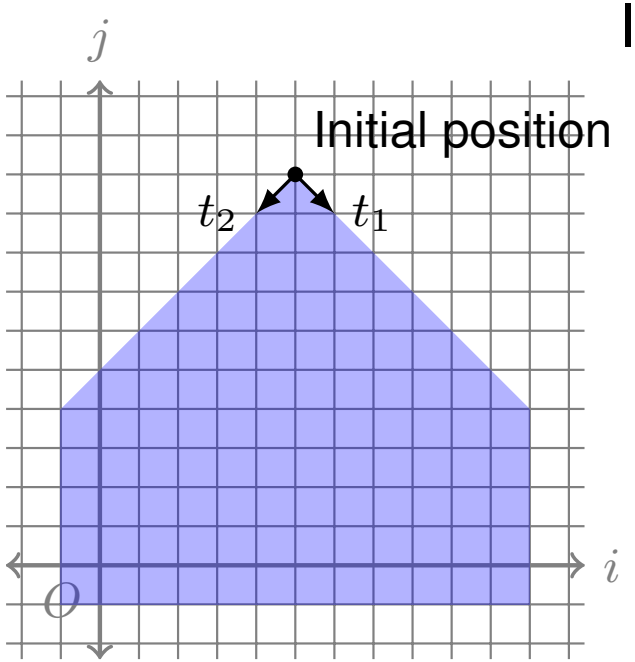
Idea : we construct $\mathcal{G}$ lazily.



# On the example

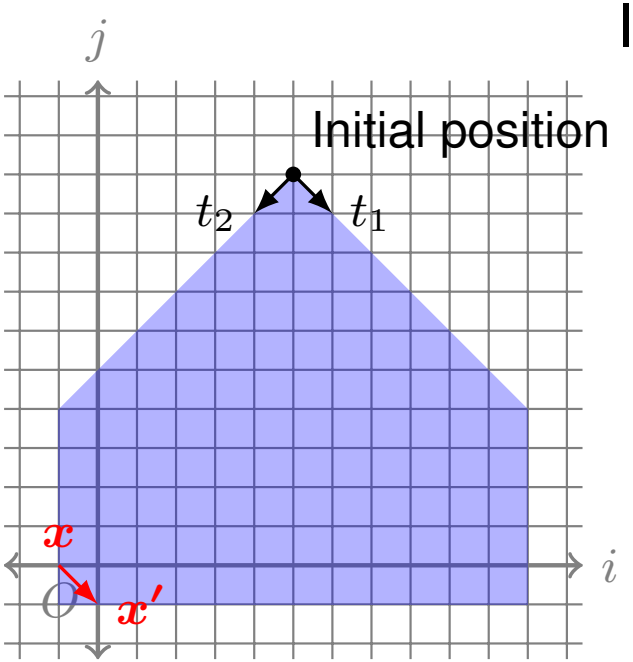
$$\boldsymbol{t_2}: \frac{0 \leq i \wedge 0 \leq j}{\begin{array}{c} i := i-1 \\ j := j-1 \end{array}} \qquad k_0 \qquad \boldsymbol{t_1}: \frac{i \leq 10 \wedge 0 \leq j}{\begin{array}{c} i := i+1 \\ j := j-1 \end{array}}$$

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



# On the example

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

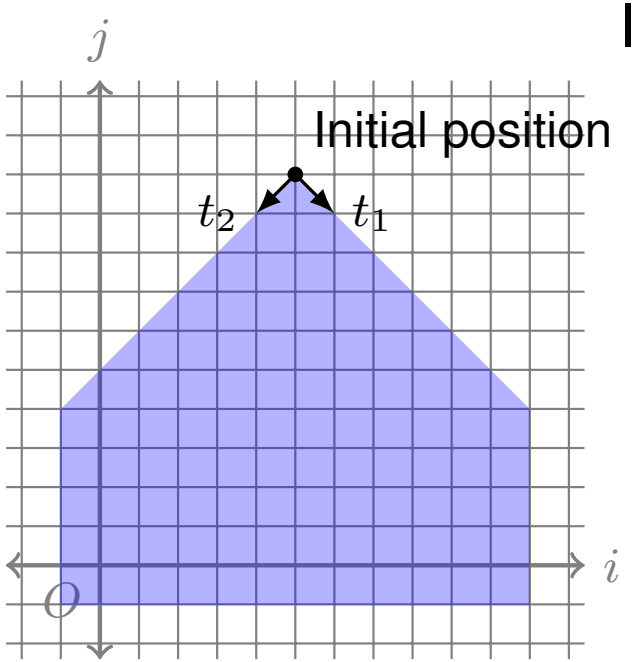



Initial $\rho \leftarrow \mathbf{0}$ :

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

# On the example

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

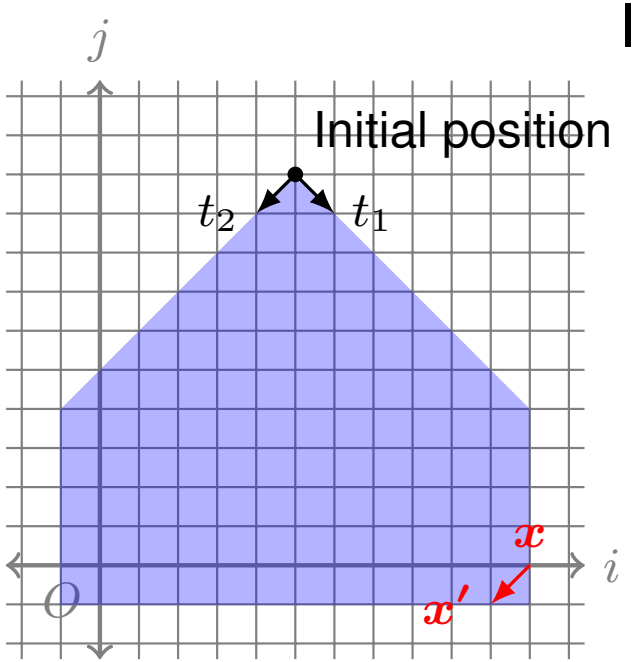



Initial $\rho \leftarrow \mathbf{0}$ :

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

# On the example

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

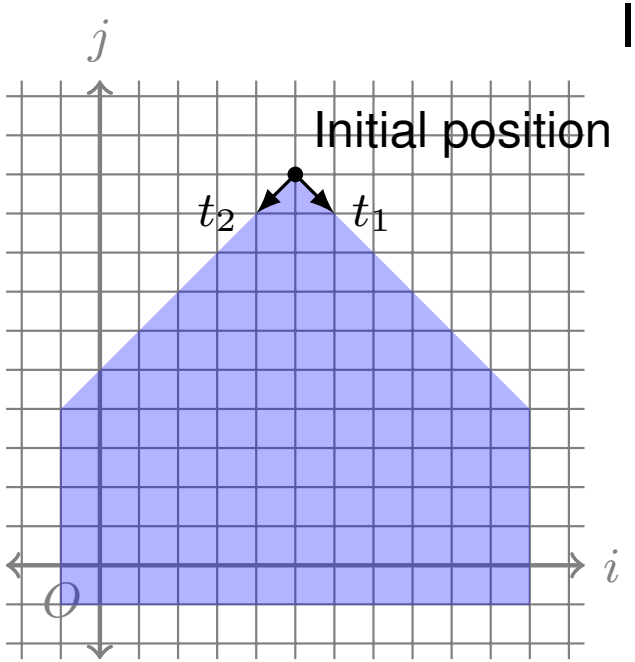



Initial $\rho \leftarrow \mathbf{0}$ :

- **SMT** $(i,j) = (-1,0), (i',j') = (0,-1)$ is a counterex for $\rho$ (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 $\rho$ (from $t_2$) $\rightsquigarrow \mathcal{G} \leftarrow \mathcal{G} \cup \{(1,1)\}$.

# On the example

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

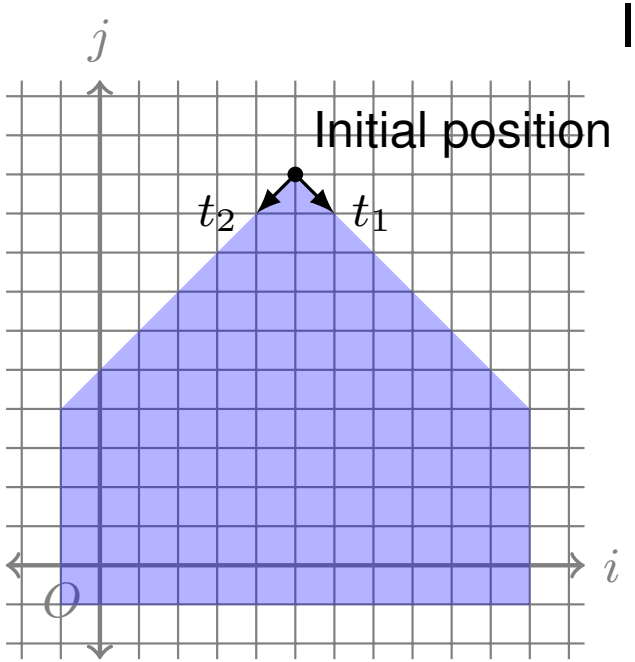



Initial $\rho \leftarrow \mathbf{0}$ :

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

# On the example

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




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

- **SMT** $(i,j) = (-1,0), (i',j') = (0,-1)$ is a counterex for $\rho$ (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 $\rho$ (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 !

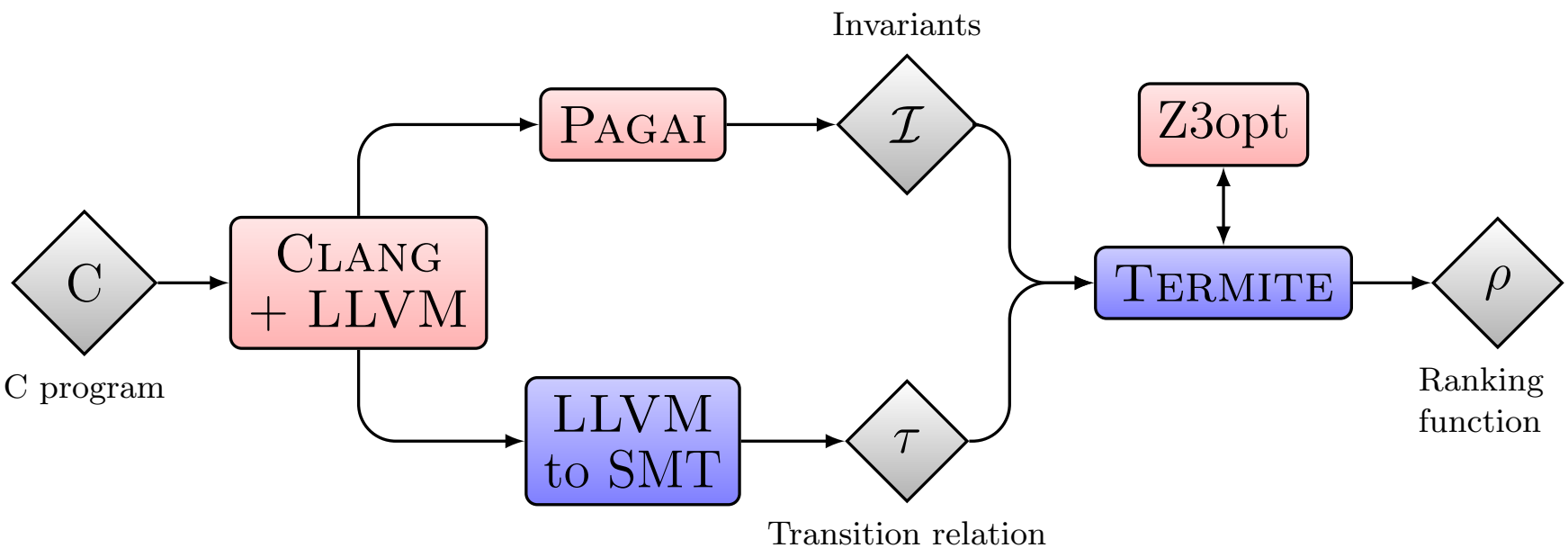
This algorithm **doesn't terminate** in general :

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

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

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

# 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, Büchi Ultimate, Loopus.

- Issue : various front-ends / invariant generators

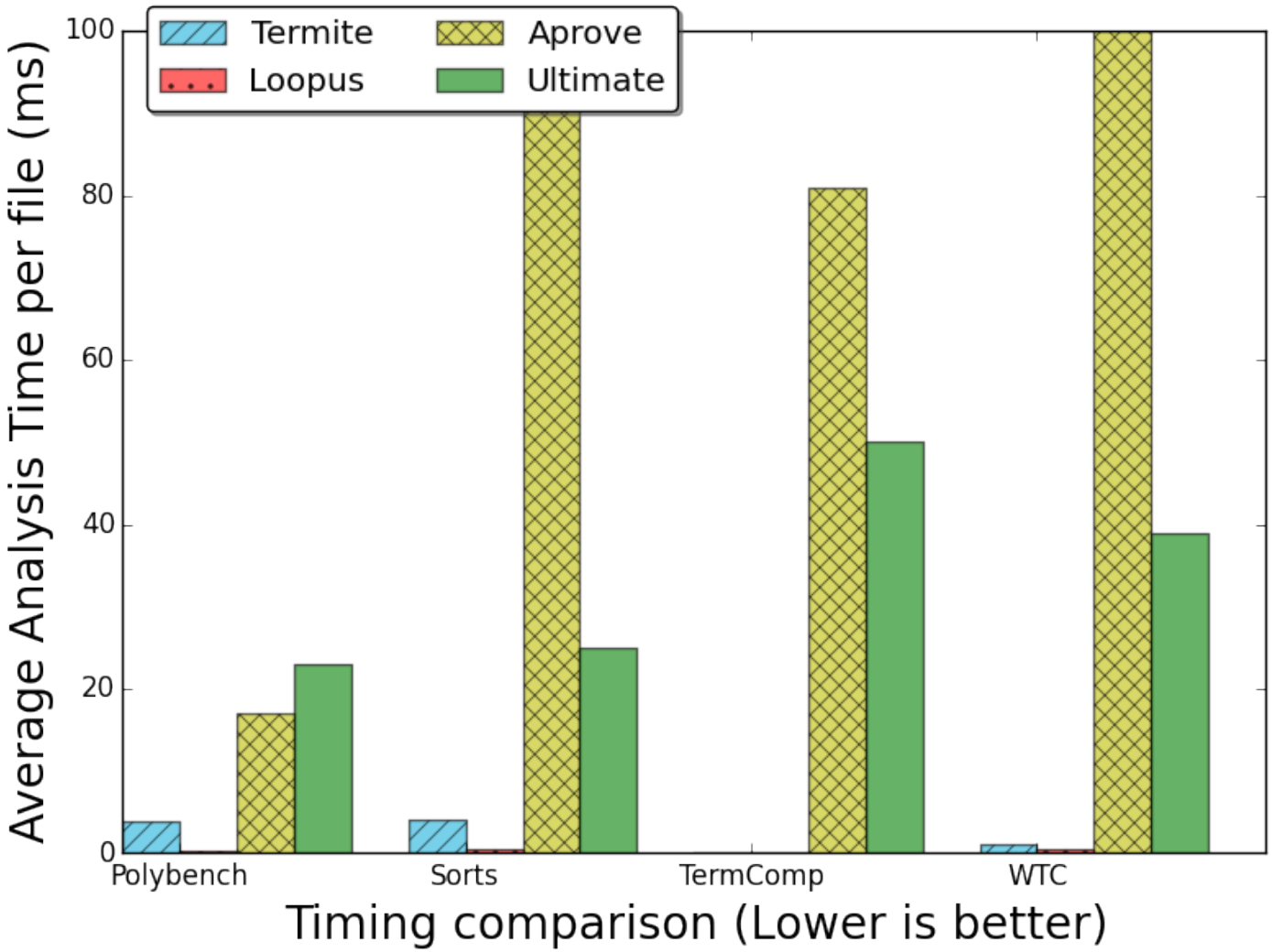
# Comparison : Linear Programming instances sizes

On WTC benchmark (average per file) :

| Tool | #lines (constraints) | #columns (variables) |
|---|---|---|
| RANK | 584 | 229 |
| TERMITE | 5 | 2 |

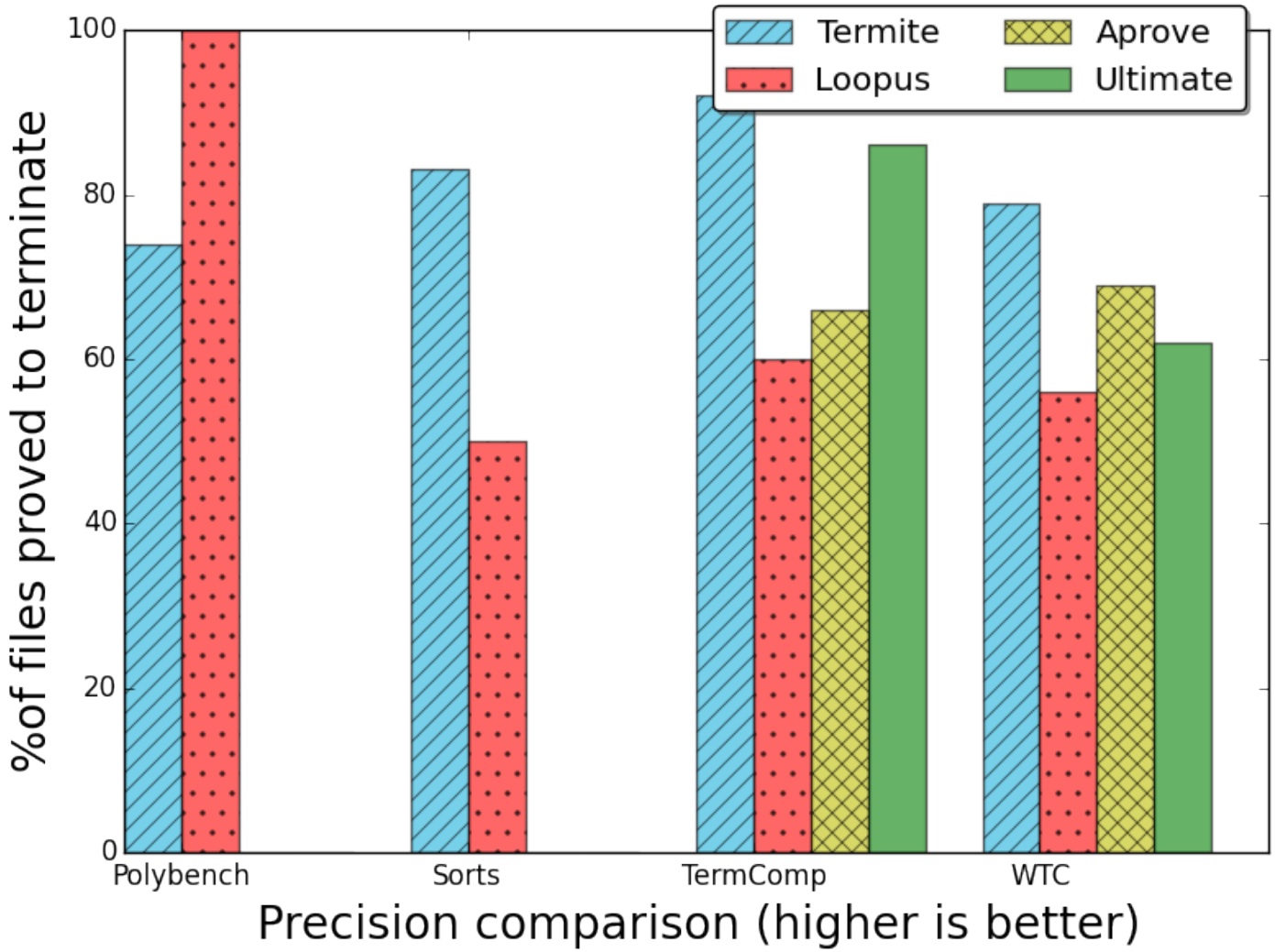
RANK is the termination tool from [Alias et al, SAS 2010]

# 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 !

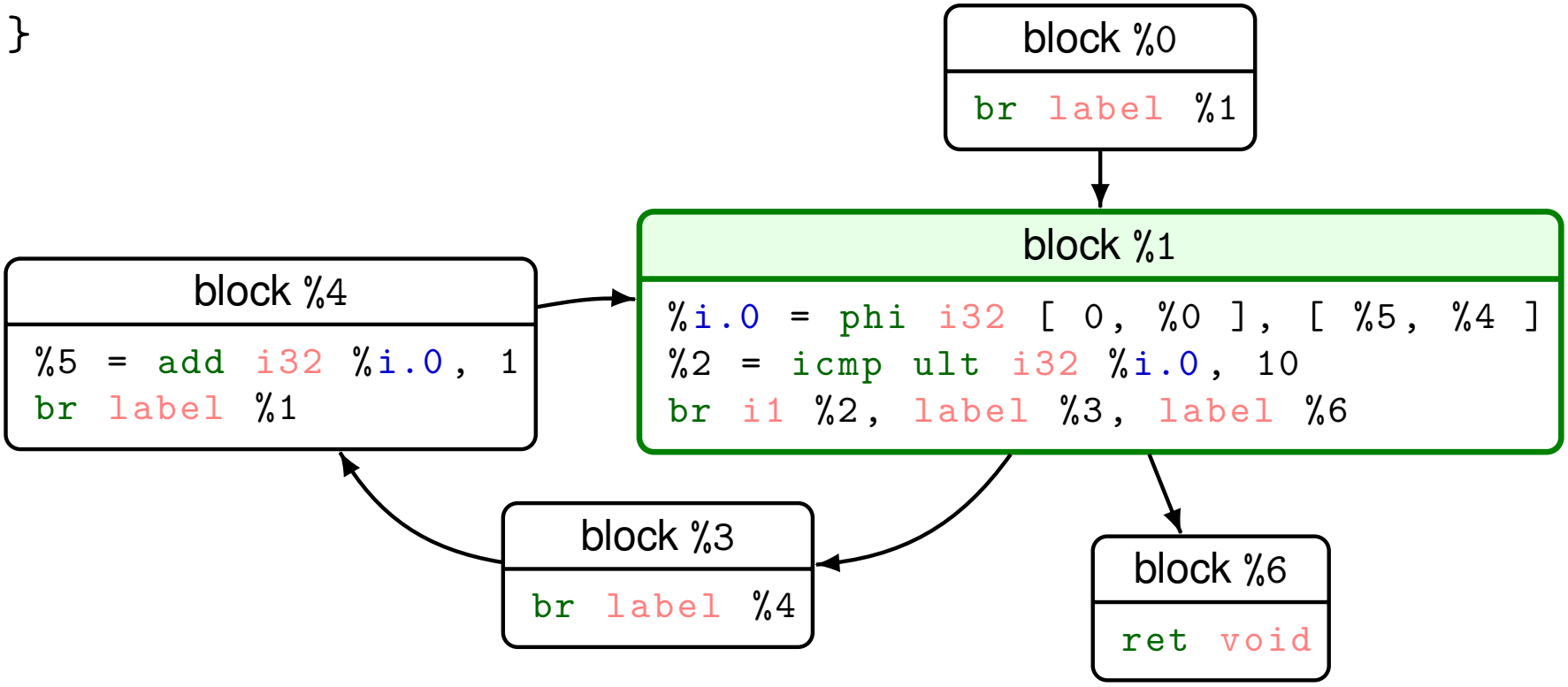
▶ `http://termite-analyser.github.io/`

# Future Work

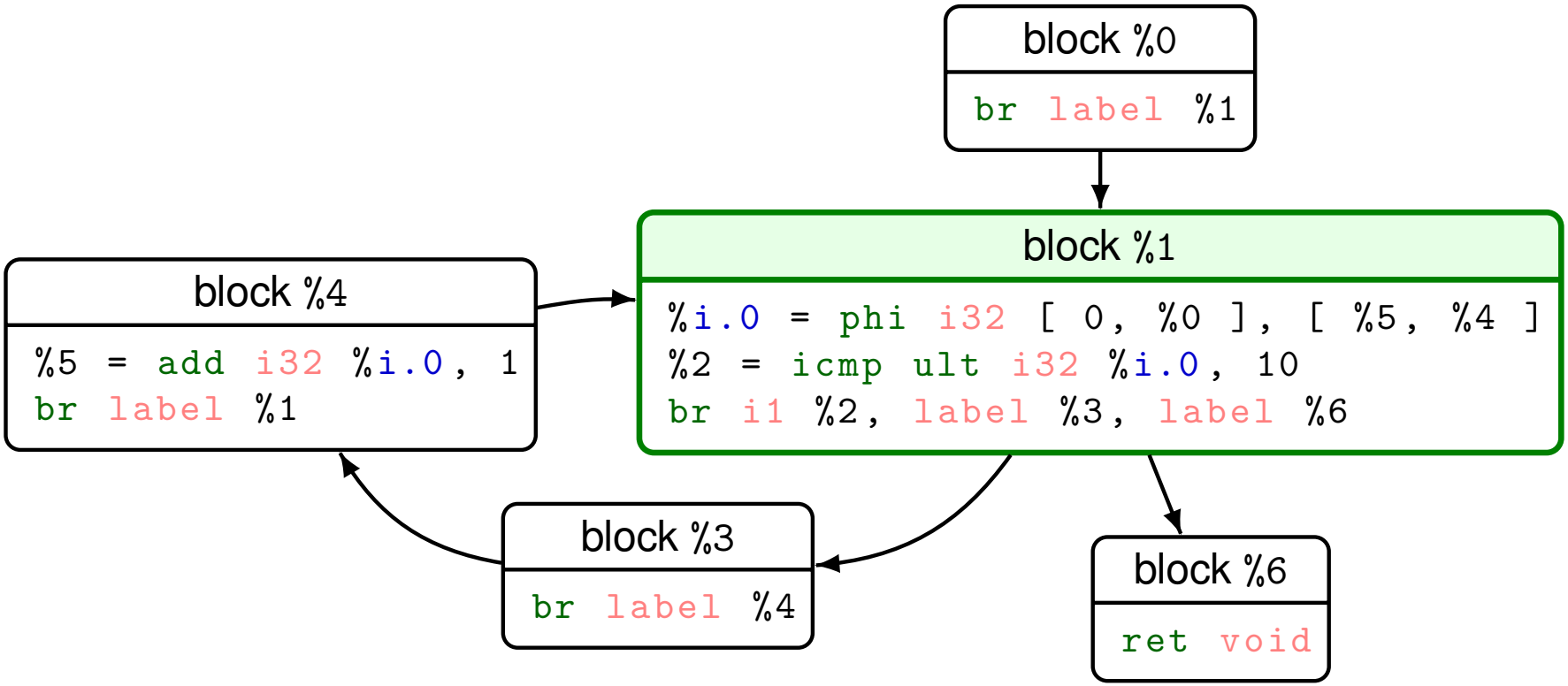
- Use the technique to also compute $\mathcal{I}$.
- Conditional termination.
- Quantifier elimination.

# Control flow graph and LLVM representation

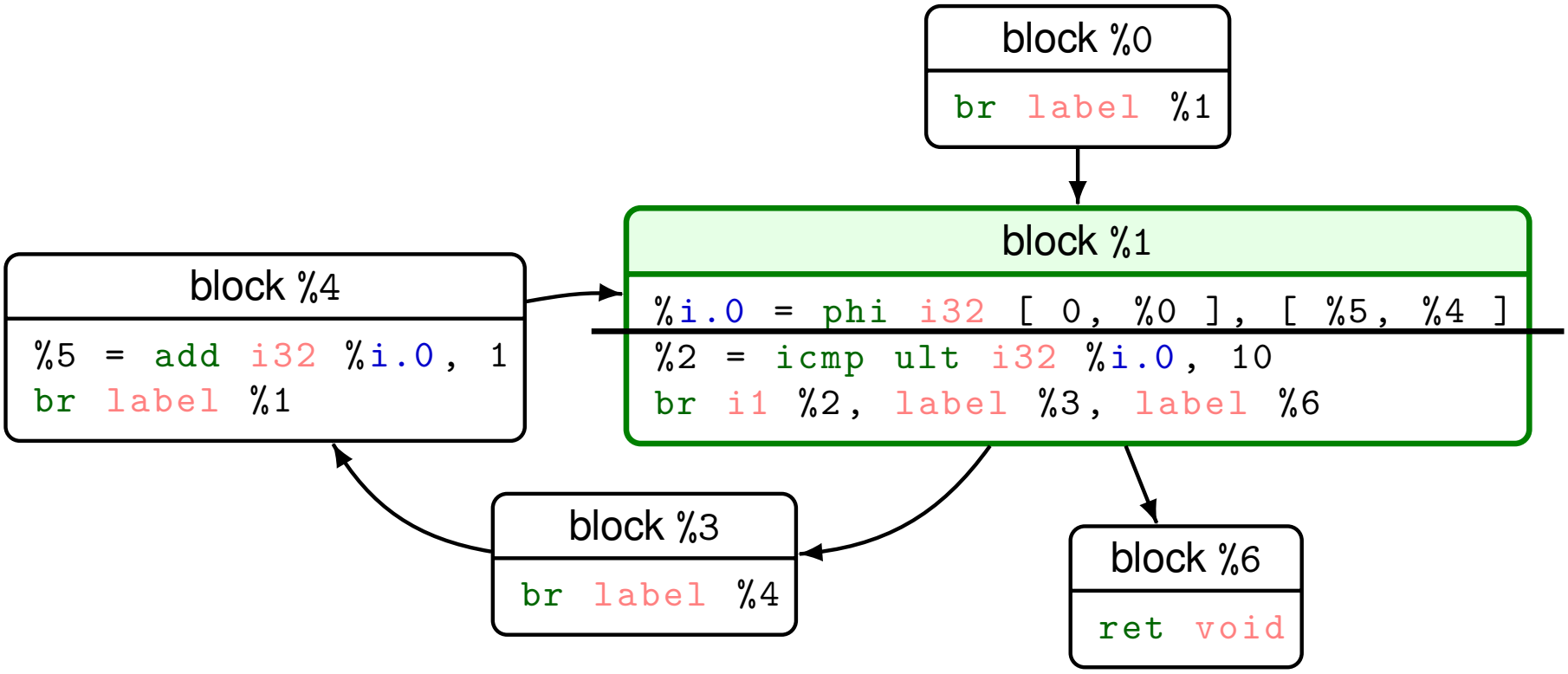
```
void simple_loop_constant() {
  for(unsigned i=0; i<10; i++) {
    // Do nothing
  }
}
```



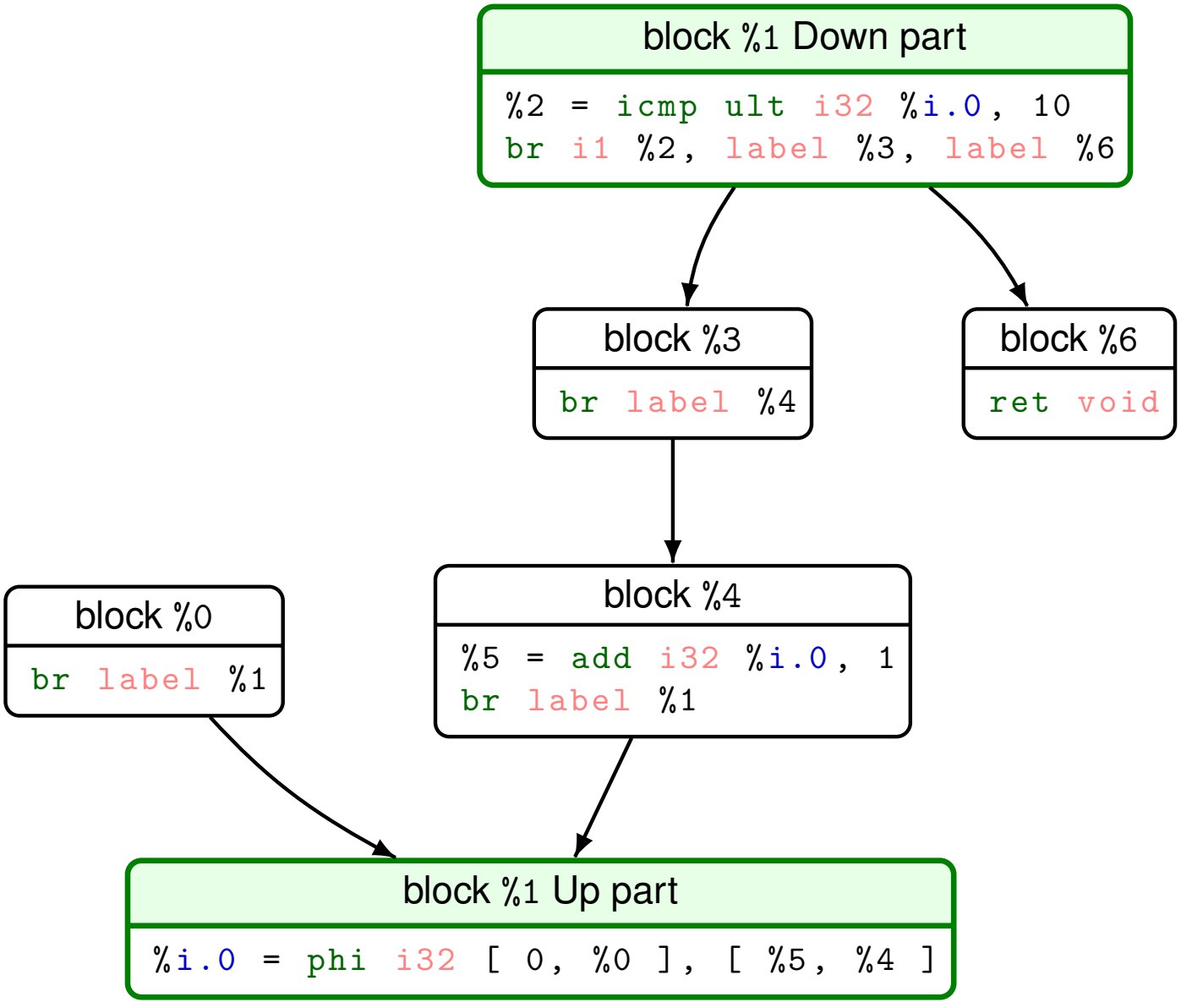
# SMT encoding for control-flow-graph (LLVM2SMT)



# SMT encoding for control-flow-graph (LLVM2SMT)



# SMT encoding for control-flow-graph (LLVM2SMT)



# SMT encoding for control-flow-graph (LLVM2SMT)