

Proving Termination with polyhedra

Laure Gonnord

with Christophe Alias, Alain Darte, and Paul Feautrier
(Compsys, ENS Lyon)

LIFL - LIP

`http://laure.gonnord.org/pro/ —
Laure.Gonnord@lifl.fr`

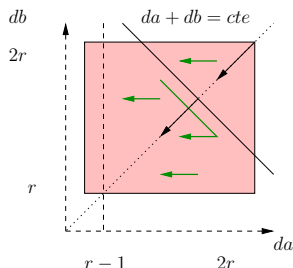


thx to C. Alias who gave the sources of his slides

Context : Transforming WHILE into FOR

Example : GCD of 2 polynomials

```
da = 2r; db = 2r;
while (da >= r) {
  cond = (da >= db || A[expr] == 0);
  if (!cond) {
    tmp = db; db = da; da = tmp - 1;
  } else da = da - 1;
}
```



Hard to optimize for a high-level synthesis tool :

- No loop unrolling possible.
- Limited software pipelining.
- ▶ Need to **bound the number of iterations**.

Contributions

Program termination with global multi-dimensional affine rankings

- Proven to be complete, fully implemented
 - **Worst-case computational complexity**, in case of success.
 - Sometimes a candidate to be an infinite loop
 - WIP : sufficient preconditions for termination.
- ▶ for **general** flowcharts programs (with a proper preprocessing !)

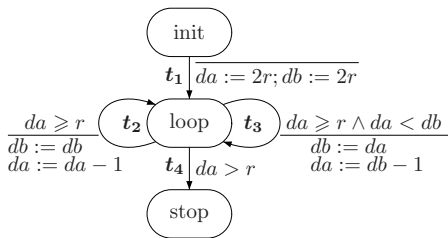
- 1 The Method
- 2 Implementation and Experimental results
- 3 Extensions and work in progress

From a Program to an affine Automaton

```

// expression expr,
// array A,
// r>0 integer.
da = 2r; db = 2r;
while (da >= r) {
  cond = ( da >= db ||
    A[expr] == 0 );
  if (!cond) {
    tmp = db;
    db = da;
    da = tmp - 1;
  }
  else da = da - 1;
}

```

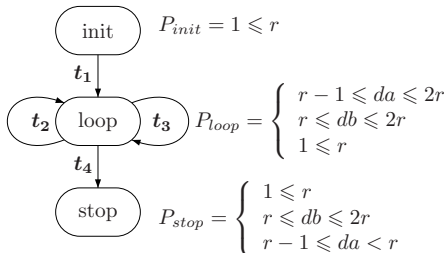


► Safe abstractions of non-affine behaviours.

From an affine Automaton to invariants

System of equations :

$$\begin{cases} P_{init} &= \{1 \leq r\} \\ P_{loop} &= P_{init} \cup t_2(P_{loop}) \\ &\quad \cup t_3(P_{loop}) \\ P_{stop} &= t_4(P_{loop}) \end{cases}$$



► Fixpoint system with affine guards and actions. Use **abstract interpretation** to get (an over approximation of) the live space of variables.

Ranking functions

A ranking function is :

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

► We restrict to \mathbb{N}^p with \leq_{lex} , and **multidimensional** affine rankings :

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

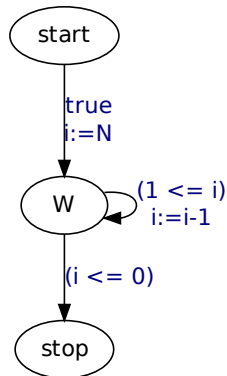
Finding a ranking function - 1

The 1D-case :

```
assume(N>0);
```

```
i=N;
```

```
while(i>0) --i;
```



Searching for :

$$\rho(start, \vec{x}) = \alpha_{start}^1 \cdot \mathbf{i} + \alpha_{start}^2 \cdot \mathbf{N}$$

$$+ \alpha_{start}^3 \cdot \mathbf{i}_0 + \alpha_{start}^4 \cdot \mathbf{N}_0 + \alpha_{start}^5$$

$$\rho(w, \vec{x}) = \alpha_w^1 \cdot \mathbf{i} + \dots$$

$$\rho(stop, \vec{x}) = \alpha_{stop}^1 \cdot \mathbf{i} + \dots$$

The constraints are :

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

Finding a ranking function - 2

The 1D-case (cont') : incoding into a **linear programming** problem !

- Constraints $\rho(pc, \vec{x}) \geq 0$ on P_{pc} :
 - Invariant for $W = \{N_0 > 0, N = N_0, 0 \leq i \leq N\}$
 - Farkas lemma

$$\begin{aligned} \rho(W, \vec{x}) &= \lambda_W^1 \cdot (N_0 - 1) + \lambda_W^2 \cdot (N_0 - N) \\ &+ \lambda_W^3 \cdot (N - N_0) + \lambda_W^4 \cdot i + \lambda_W^5 \cdot (N - i) \end{aligned}$$

+affine form for $\rho(W, \vec{x})$:

$$\rho(W, \vec{x}) = \alpha_W^1 \cdot i + \alpha_W^2 \cdot N + \alpha_W^3 \cdot i_0 + \alpha_W^4 \cdot N_0 + \alpha_W^5$$

► Identifying **i** : $\alpha_W^1 = \lambda_W^4 - \lambda_W^5, \dots$

- Constraints for decreasing transitions : similar

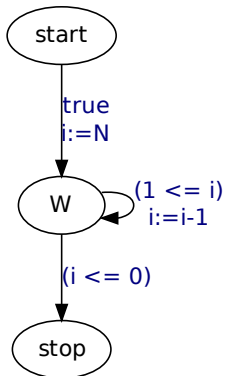
Finding a ranking function - 3

The 1D-case :

```
assume(N>0);
```

```
i=N;
```

```
while(i>0) --i;
```



We find :

```
state start:
```

```
2+N__o
```

```
state W:
```

```
1+i
```

```
state stop:
```

```
0
```

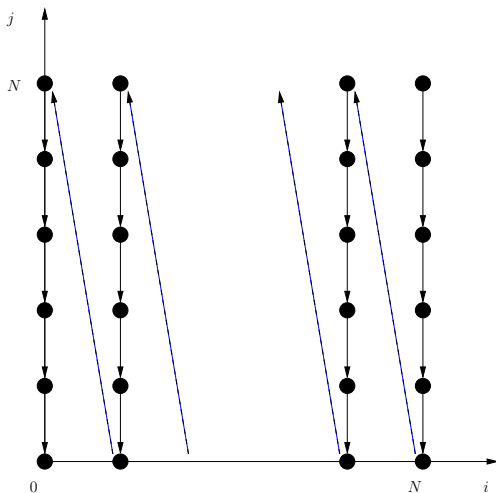
Finding a ranking function - 4

The nD-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**.

Example - 1

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

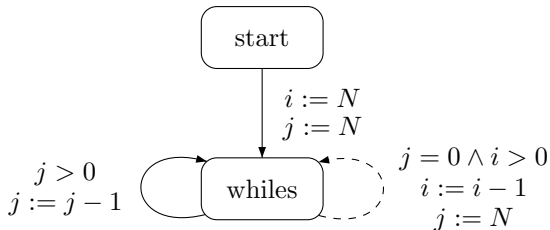


Example - 2

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

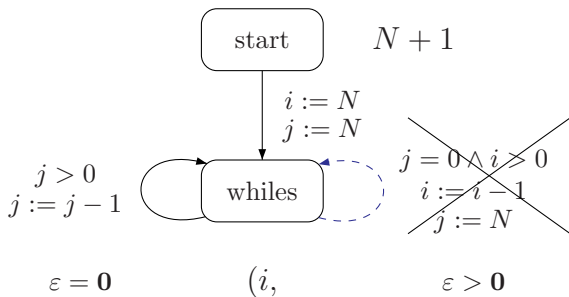
Invariant for `whiles` :

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



Example - 2

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

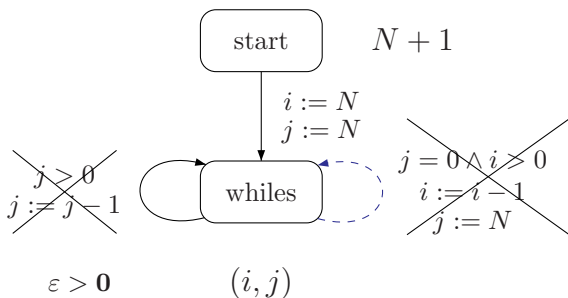


Invariant for `whiles` :

$$-1 < i \leq N, -1 < j \leq N, N > 0, N = N_0$$

Example - 2

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

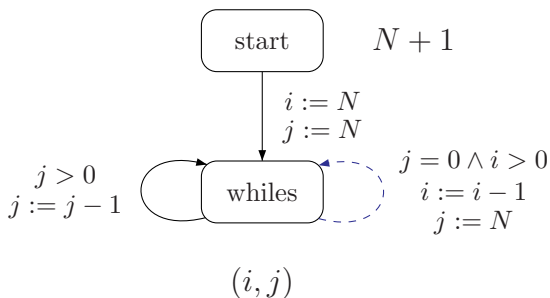


Invariant for whiles :

$$-1 < i \leq N, -1 < j \leq N, N > 0, N = N_0$$

Example - 2

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



Invariant for `whiles` :

$$-1 < i \leq N, -1 < j \leq N, N > 0, N = N_0$$

In summary

From (arbitrary) flowchart programs :

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

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**.*

- 1 The Method
- 2 Implementation and Experimental results
- 3 Extensions and work in progress

Our toolsuite

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

► The two first tools are described in :

TAPAS 2010 - Feautrier/Gonnord

Accelerated Invariant Generation for C Programs with Aspic and C2fsm

Sorting algorithms

Sorting arrays :

Name	LOCs	Time(c2fsm/analysis) ¹	dim	WCCC
selection	20	1.0/0.4	3	$\frac{N^2}{2} + \frac{3N}{2} + 1$
insertion	12	0.6/0.22	3	$\frac{N^2}{2} + \frac{3N}{2} + 1$
bubble	22	1.2/0.4	3	$N^2 + 2$
shell	23	1.0/1.1	4	$\frac{N^3}{6} - \frac{N}{6}$
heap	45	3.0/2.8	3	$4N^2 - 11N + 9$

► More examples on :

<http://www.ens-lyon.fr/LIP/COMPSYS/Tools/Ranking/>

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

Some comments on experimental results

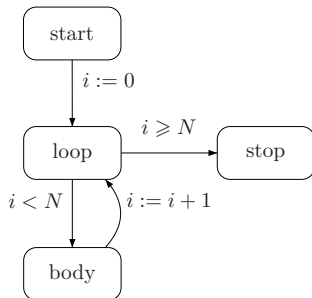
- The algorithm scales (relatively) well.
- The form of the automaton has a strong impact on the invariants.
- The precision of invariants is **crucial**.

Piecewise-affine ranking functions

```

i=0;
while(i<N)
{
  i = i + 1;
}

```



We expect $loop \mapsto (1, 2(N - i))$, $body \mapsto (1, 2(N - i) - 1)$.
 But... the **unknown sign of N** prevents to conclude.

► A **piece-wise affine ranking** is required :

$$\rho(loop, i, N) = \begin{cases} N \geq 0 : (1, 2(N - i)) \\ N < 0 : (1) \end{cases}$$

$$\rho(body, i, N) = (1, 2(N - i) - 1)$$

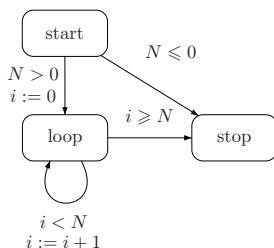
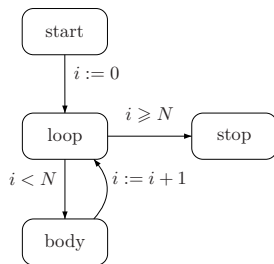
- 1 The Method
- 2 Implementation and Experimental results
- 3 Extensions and work in progress

Modifying the graph : cutpoints

Definition

A set of cut points is a subset of control points such that their removal causes the graph to become acyclic

- ▶ Compute the rankings functions on the cut points after **path compression**



$(N - i)$ is found for loop

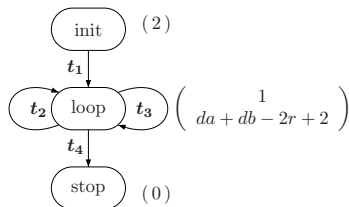
Computing a “WCET”

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

$$\begin{aligned} WCCC &\leq \#\rho(\text{init}, P_{\text{init}}) \\ &\quad + \#\rho(\text{loop}, P_{\text{loop}}) \\ &\quad + \#\rho(\text{end}, P_{\text{end}}) \\ &= 2 + \#\{(1, i) \mid 1 \leq i \leq 2r + 2\} \\ &= 2r + 4 \end{aligned}$$



Reference

The algorithm, the extensions and the experimental results are published in

[SAS 2010 - Alias/Darte/Feautrier/Gonnord](#)

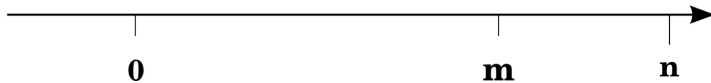
Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs

Work In Progress : finding sufficient conditions - 1

```

int catmouse(){
  int x,n,m;
  x=0;
  while(x<=n) {
    if (x<=m) ++x;
    else --x;
  }
}

```

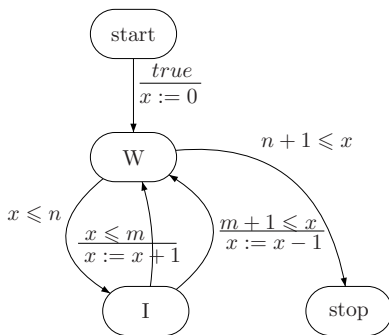


WIP : finding sufficient conditions - 2

```

int catmouse(){
  int x,n,m;
  x=0;
  W: while(x<=n)
    {
  I:  if (x<=m) ++x;
      else --x;
    }
}

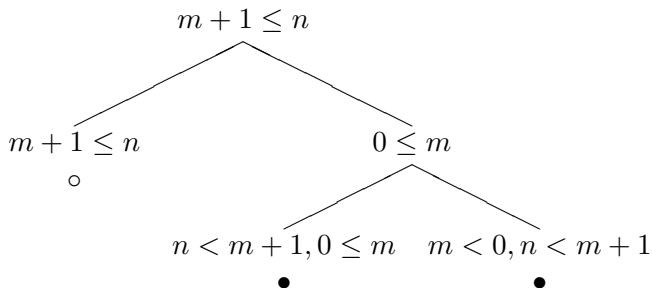
```



- ▶ Without any additional assumption, our method fails !

WIP : finding sufficient conditions - 3

- 1 Compute the automaton
- 2 Try aspic+rank ! If it fails, compute **firing conditions**
- 3 Find conditions on parameters to prove emptiness (**parametric linear programming**). Then retry !



Conclusion

An algorithm to prove termination :

- on arbitrary programs
- using the link between scheduling and ranking functions
- using **polyhedra** in the large : linear relation analysis, (parametric) linear programming, computing schedules, ...
- that gives upper approximations of the worst case complexity.

Future work :

- More experiments on bigger codes : a modular approach is necessary
- Validate/Publish the sufficient conditions
- Investigate the use of disjunctive invariants