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).
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.
Termination proofs, what for?

- for fun!
- prove total correctness
- reactive systems need termination of inner loops
- (compute worst-case bounds)
Contributions

**Program termination** with global multi-dimensional affine rankings:

- Proven to be complete, fully implemented.
- Work on scalability: program-level optimisations, algorithmic optimisation.

▶ for **general** flowcharts programs (with a proper preprocessing!)
Method

1. Preliminaries
2. Kernel algorithm
3. Implementation and Experimental results
4. Scalability issues

What is a ranking function?
**Summary**

- From a program: generate an affine automaton with invariants.
- Generate a linear programming (LP) instance and solve it.
- If it is not sufficient, retry in a lesser automaton.
Our model for programs

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

- \(\mathcal{K}\) : control points
- \(n\) rational variables \(x\)
- \(k_{init} \in \mathcal{K}\) 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

**Software Eng.**

From program to an accurate model. Semantic transformation.

**Problem 1**: how to compute such (numerical) automata from general programs?

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

Home-made tool \(C2\text{FSM}\) or Rose-based tool \(ST\text{OP}\) TODO: implement inside the LLVM framework.
**Invariants**

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

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

\[ k_1 : N \geq 0 \land i \geq 0 \land i \leq N \]
From an affine Automaton to invariants

**Problem 2** : how to compute numerical invariants?

- Accurate numerical invariants
- Polyhedra
- Within a reasonable amount of time.

We use abstract interpretation.

**Tools** : ASPIC or the promising Pagai (J. Henry)

**Software Eng.**

Use compact (internal) representations. Carefully deal with interprocedural analysis.
Ranking functions

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

```
assume(N>0);
i:=N
(1 <= i)
i:=i-1
i <= 0)
```
Pb and Restrictions

**Pb** How to compute a ranking function for a given automata? We restrict to $\mathbb{N}^p$ with $\leq_{lex}$, and **multidimensional** affine rankings:

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

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

▶ Demo!
1. Method
2. Kernel algorithm
3. Implementation and Experimental results
4. Scalability issues
Finding a ranking function - 1

The 1D-case:

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

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

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

$$\rho(W, \vec{x}) = \alpha_{W,1}.i + \ldots$$

$$\rho(stop, \vec{x}) = \alpha_{stop,1}.i + \ldots$$

The constraints are:

- For each control point: $\rho(pc, \vec{x}) \geq 0$ on $P_{pc}$
- For each transition $(\vec{x'} - \vec{x}) \in t \Rightarrow \rho(dest, \vec{x'}) - \rho(src, \vec{x'}) > 0$
Finding a 1D ranking function 2

The 1D-case: incoding into a **linear programming** problem

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

Constraint of the form “linear positive form on a convex polyhedra” \( \blacktriangleright \) Farkas Lemma.

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”:

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

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

For decreasing transitions:

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

becomes

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

▶ “some” more constraints.

**Objective function**: Maximize \(\sum_t \epsilon_t\)
Finding a 1D ranking function - 4

The 1D-case:
assume(N>0);
i=N;
while(i>0) --i;

We find:
state start:
2+N
state W:
1+i
state stop:
0
Finding a ranking function - nD

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

- $i = 0$; $T = T$, 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**.
Example - 1

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

Laure Gonnord (LIP) Proving Termination of programs Dec 2013 - Rennes
Example - 2

```c
//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:

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

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

Invariant for \texttt{while}s :

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

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

Invariant for whiles:

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

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.

This algorithm (+ completeness results) and preliminary experimental results have been published in Static Analysis Symposium (Alias et al, 2010).
1. Method

2. Kernel algorithm

3. Implementation and Experimental results

4. Scalability issues
Our toolsuite

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

Published in [TAPAS2010] and [CSTV 2012].

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 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.
1 Method

2 Kernel algorithm

3 Implementation and Experimental results

4 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.
Scalability issues

Divide and conquer 1/2

**Software Eng.**

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.

Divide and conquer 2/2

if (up == 0) {
    i = 1;
    while (i <= n) {
        a[i] = a[i+n];
        i = i + 1;
    }
}

(a) Before slicing

if (up == 0) {
    i = 1;
    while (i <= n) {
        i = (i + 1);
    }
}

(b) After slicing
Work on the initial algorithm 1/2

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

Lesson - Limits on Soft. Eng. :-)  
Sometimes, working on the initial algorithm itself is the only solution.

- 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.

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></td>
<td>#LP</td>
<td>Avg. #lines/#cols</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
A survey on termination techniques : [Ben Amram 2012].

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

Back to the initial problem:
- Generate for loops (source-to-source transformation).
- Link with program scheduling?

Key idea
Finding a ranking function is nearly the same as finding a (sequential) schedule.