Proving Termination of Imperative Programs using Max-SMT

Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell and Albert Rubio

Universitat Politècnica de Catalunya

FMCAD, October 2013

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

Motivation

- Prove termination of imperative programs automatically.
- Find ranking functions.
- Find supporting invariants.
- How to guide the search!.

Simple example

```
void simpleT(int x, int y) {

  while (y>0) {

    while (x>0) {
      x=x-y;
      y=y+1;
    }
    y=y-1;
  }
}
```

Simple example

```
void simpleT(int x, int y) {

  while (y>0) {

    while (x>0) {
      x=x-y;
      y=y+1;
    }
    y=y-1;
  }
}
```

Terminates.

Simple example

```
void simpleT(int x, int y) {

  while (y>0) { Ranking function: y
    // Inv: y>0
    while (x>0) { Ranking function: x
      x=x-y;
      y=y+1;
    }
    y=y-1;
  }
}
```

Terminates.

Goals

Main goal: fully-automatic program termination analysis.

Goals

Main goal: fully-automatic program termination analysis.

- Consider integer linear programs.
- Use the constraint-based method [CSS2003, BMS2005].

Goals

Main goal: fully-automatic program termination analysis.

- Consider integer linear programs.
- Use the constraint-based method [CSS2003, BMS2005].
  - Use an SMT solver to solve the constraints.

Goals

Main goal: fully-automatic program termination analysis.

- Consider integer linear programs.
- Use the constraint-based method [CSS2003, BMS2005].
  - Use an SMT solver to solve the constraints.
- Use Max-SMT to guide the search
  - Invariant conditions are *hard*
  - Termination conditions are *soft*

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: T = linear integer/real arithmetic.

$$(x < 0 \vee x \leq y \vee y < z) \wedge (x \geq 0) \wedge (x > y \vee y < z)$$

$$\{x = 1, y = 0, z = 2\}$$

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: T = linear integer/real arithmetic.

$$(x < 0 \vee x \leq y \vee \underline{y < z}) \wedge (\underline{x \geq 0}) \wedge (x > y \vee \underline{y < z})$$

$$\{x = 1, y = 0, z = 2\}$$

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: T = linear integer/real arithmetic.

$$(x < 0 \vee x \leq y \vee \underline{y < z}) \wedge (\underline{x \geq 0}) \wedge (x > y \vee \underline{y < z})$$

$$\{x = 1, y = 0, z = 2\}$$

There exist very efficient solvers: yices, z3, Barcelogic, ...
Can handle large formulas with a complex boolean structure.

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: $T =$ non-linear (polynomial) integer/real arithmetic.

$$(x^2 + y^2 > 2 \vee x \cdot z \leq y \vee y \cdot z < z^2) \wedge (x > y \vee 0 < z)$$

$$\{x = 0, y = 1, z = 1\}$$

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: T = non-linear (polynomial) integer/real arithmetic.

$$(x^2 + y^2 > 2 \vee \underline{x \cdot z \leq y} \vee y \cdot z < z^2) \wedge (x > y \vee \underline{0 < z})$$

$$\{x = 0, y = 1, z = 1\}$$

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: T = non-linear (polynomial) integer/real arithmetic.

$$(x^2 + y^2 > 2 \vee \underline{x \cdot z \leq y} \vee y \cdot z < z^2) \wedge (x > y \vee \underline{0 < z})$$

$$\{x = 0, y = 1, z = 1\}$$

Non-linear arithmetic decidability:

- *Integers:* undecidable
- *Reals:* decidable **but** unpractical due to its complexity.

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: $T =$ non-linear (polynomial) integer/real arithmetic.

$$(x^2 + y^2 > 2 \vee \underline{x \cdot z \leq y} \vee y \cdot z < z^2) \wedge (x > y \vee \underline{0 < z})$$

$$\{x = 0, y = 1, z = 1\}$$

Non-linear arithmetic decidability:

- *Integers:* undecidable
- *Reals:* decidable **but** unpractical due to its complexity.

Incomplete solvers focused on either satisfiability or unsatisfiability.

SMT solving

Input: Given a boolean formula φ over some theory T.

Question: Is there any interpretation that satisfies the formula?

Example: $T =$ non-linear (polynomial) integer/real arithmetic.

$$(x^2 + y^2 > 2 \vee \underline{x \cdot z \leq y} \vee y \cdot z < z^2) \wedge (x > y \vee \underline{0 < z})$$

$$\{x = 0, y = 1, z = 1\}$$

Non-linear arithmetic decidability:

- *Integers:* undecidable
- *Reals:* decidable **but** unpractical due to its complexity.

Incomplete solvers focused on either satisfiability or unsatisfiability.

Need to handle again large formulas with complex boolean structure.

Barcelogic SMT-solver works very well finding solutions

Optimization problems

(Weighted) Max-SMT problem

Input: Given an SMT formula $\varphi = C_1 \wedge \ldots \wedge C_m$ in CNF, where some of the clauses are *hard* and the others *soft* with a weight.

Output: An assignment for the hard clauses that minimizes the sum of the weights of the falsified soft clauses.

$$(x^2 + y^2 > 2 \vee x \cdot z \leq y \vee y \cdot z < z^2) \wedge (x > y \vee 0 < z \vee w(5)) \wedge \ldots$$

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

Invariants

Definition

An *invariant* of a program at a location is an assertion over the program variables that remains true whenever the location is reached.

Invariants

Definition

An *invariant* of a program at a location is an assertion over the program variables that remains true whenever the location is reached.

Definition

An invariant is said to be *inductive* at a program location if:

- *Initiation condition:* It holds the first time the location is reached.
- *Consecution condition:* It is preserved under every cycle back to the location.

Invariants

Definition

An *invariant* of a program at a location is an assertion over the program variables that remains true whenever the location is reached.

Definition

An invariant is said to be *inductive* at a program location if:

- *Initiation condition:* It holds the first time the location is reached.
- *Consecution condition:* It is preserved under every cycle back to the location.

We are focused on inductive invariants.

Constraint-based invariant generation [CSS2003]

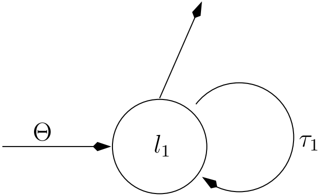
- Assume input programs consist of linear expressions
- Model the program as a *transition system*

Constraint-based invariant generation [CSS2003]

- Assume input programs consist of linear expressions
- Model the program as a *transition system*

Simple example:

```
int main()
{
    int x;
    int y=-x;
l1: while (x>=0) {
      x--;
      y--;
    }
}
```



$\rho_\Theta : x' = x, \quad y' = -x$

$\rho_{\tau_1} : x \geq 0, \quad x' = x - 1, \quad y' = y - 1$

Constraint-based invariant generation [CSS2003]

Assume we have a transition system with linear expressions.

Constraint-based invariant generation [CSS2003]

Assume we have a transition system with linear expressions.

Keys:

Constraint-based invariant generation [CSS2003]

Assume we have a transition system with linear expressions.

Keys:

- Use a template for candidate invariants.

$$c_1 x_1 + \ldots + c_n x_n + d \leq 0$$

Constraint-based invariant generation [CSS2003]

Assume we have a transition system with linear expressions.

Keys:

- Use a template for candidate invariants.

$$c_1 x_1 + \ldots + c_n x_n + d \leq 0$$

- Check initiation and consecution conditions obtaining an $\exists\forall$ problem.

Constraint-based invariant generation [CSS2003]

Assume we have a transition system with linear expressions.

Keys:

- Use a template for candidate invariants.

$$c_1 x_1 + \ldots + c_n x_n + d \leq 0$$

- Check initiation and consecution conditions obtaining an $\exists\forall$ problem.
- Transform it using Farkas' Lemma into an $\exists$ problem over non-linear arithmetic.

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

Motivation:

- Prove termination of imperative programs automatically.
- Find ranking functions.
- Find supporting invariants.
- How to guide the search!.

Ranking functions and Invariants

Basic method: find a single *ranking function* $f : \text{States} \to \mathbb{Z}$, with $f(S) \geq 0$ and $f(S) > f(S')$ after every iteration.

Ranking functions and Invariants

Basic method: find a single *ranking function* f : States $\rightarrow \mathbb{Z}$, with $f(S) \geq 0$ and $f(S) > f(S')$ after every iteration.
It does not work in practice in many cases.
What is (at least) necessary?

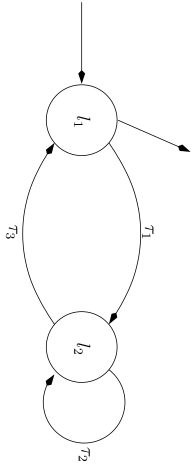
Ranking functions and Invariants

Basic method: find a single *ranking function* f : States $\rightarrow \mathbb{Z}$, with $f(S) \geq 0$ and $f(S) > f(S')$ after every iteration.
It does not work in practice in many cases.
What is (at least) necessary?

- Find supporting Invariants
- Consider a (lexicographic) combination of ranking functions

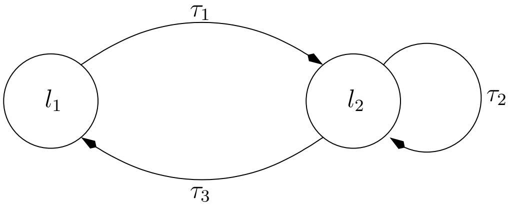
Ranking functions and Invariants: Example

```
int main()
{
    int x=indet(),y=indet(),z=indet();
l1: while (y>=1) {
      x--;
l2:   while (y<z) {
        x++; z--;
      }
      y=x+y;
    }
}
```



Ranking functions and Invariants: Example

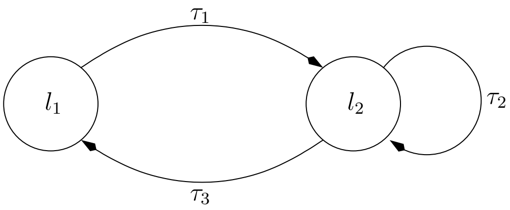
Transition system:



$$\begin{array}{lllll}
\rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}$$

Ranking functions and Invariants: Example

Transition system:

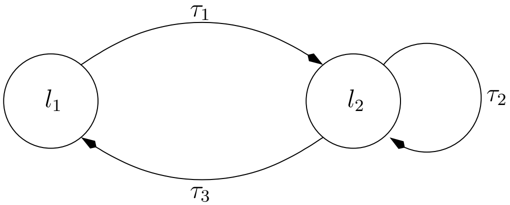


$$\begin{array}{llllll} \rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\ \rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\ \rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z \end{array}$$

$f(x, y, z) = z$ is a ranking function for τ_2

Ranking functions and Invariants: Example

Transition system:

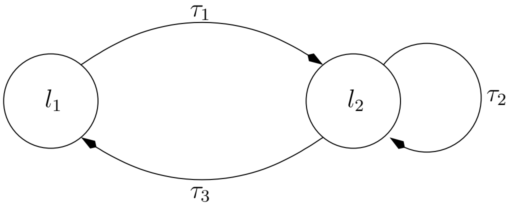


$$
\begin{array}{llllll}
\rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

It is necessary a supporting invariant $y \geq 1$ at ℓ_2.

Ranking functions and Invariants: Example

Transition system:

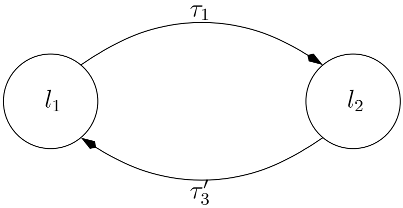


$$
\begin{array}{lllll}
\rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

We can discard all executions that pass through τ_2.

Ranking functions and Invariants: Example

Transition system:



$$\begin{array}{llllll}
\rho_{\tau_1}: & & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau'_3}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}$$

We can discard all executions that pass through τ_2.

Ranking functions and Invariants

In order to discard a transition τ_i we need to find a ranking function f over the integers such that:

1. $\tau_i \Longrightarrow f(x_1, \ldots, x_n) \geq 0$ (bounded)
2. $\tau_i \Longrightarrow f(x_1, \ldots, x_n) > f(x'_1, \ldots, x'_n)$ (strict-decreasing)
3. $\tau_j \Longrightarrow f(x_1, \ldots, x_n) \geq f(x'_1, \ldots, x'_n)$ for all j (non-increasing)

Ranking functions and Invariants: Combined problem

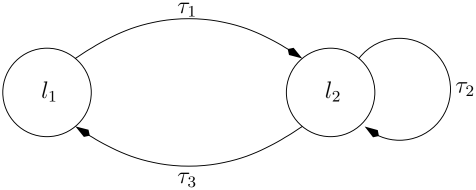
In order to prove properties of the ranking function we may need to generate invariants.

Generation of both invariants and ranking functions should be combined in the same satisfaction problem.

Both are found at the same time [BMS2005].

Ranking functions and Invariants: Example

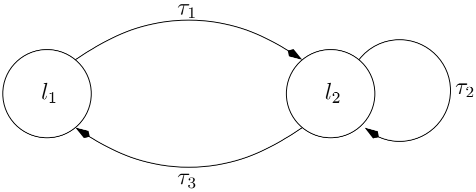
Transition system:



$$
\begin{array}{llllll}
\rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

Ranking functions and Invariants: Example

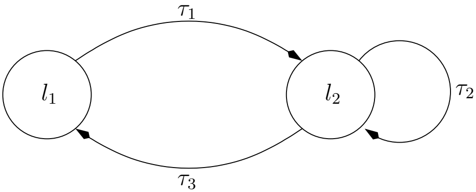
Transition system:



$$
\begin{array}{lllllll}
\rho_{\tau_1}: & l_1, & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & l_2, & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & l_2, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

Ranking functions and Invariants: Example

Transition system:

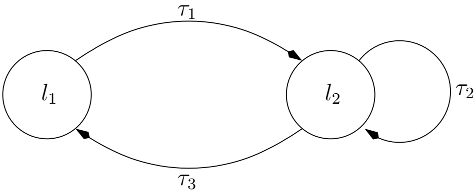


$$
\begin{array}{llllll}
\rho_{\tau_1'}: & 0 \leq 0, & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y \geq 1, & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

and ranking function $f(x, y, z) = z$, fulfiling all properties for τ_2

Ranking functions and Invariants: Example

Transition system:

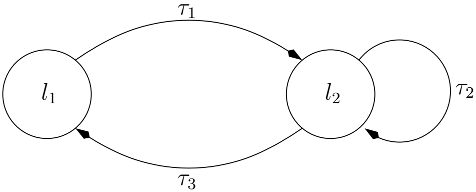


$$
\begin{array}{llllll}
\rho_{\tau_1}: & & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y \geq 1, & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

and ranking function $f(x, y, z) = z$, fulfiling all properties for τ_2

Ranking functions and Invariants: Example

Transition system:

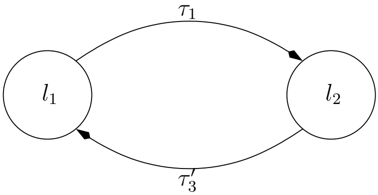


$$
\begin{array}{lllllll}
\rho_{\tau_1}: & & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & \color{blue}{y \geq 1}, & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & \color{blue}{y \geq 1}, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

and ranking function $\color{blue}{f(x, y, z) = z}$, fulfiling all properties for τ_2
we can remove τ_2

Ranking functions and Invariants: Example

Transition system:



$$\rho_{\tau_1}: \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$$
$$\rho_{\tau_3'}: \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z$$

and ranking function $f(x, y, z) = z$, fulfiling all properties for τ_2

we can remove τ_2

Ranking functions and Invariants: Combined problem

In order to prove properties of the ranking function we may need to generate invariants.

Generation of both invariants and ranking functions should be combined in the same satisfaction problem.

Both are found at the same time [BMS2005].

Ranking functions and Invariants: Combined problem

In order to prove properties of the ranking function we may need to generate invariants.

Generation of both invariants and ranking functions should be combined in the same satisfaction problem.

Both are found at the same time [BMS2005].

In order to be correct we need to have two transition systems:

- the original system (extended with all found invariants) for invariant generation.
- the *termination transition system* which includes the transitions not yet proved to be terminating.

Similar to the *cooperation graph* in [BCF2013].

Our approach: Example

The approach in [BMS2005] is nice but in practice some problems arise:

- May need several invariants before finding a ranking function.

  We should be able to generate invariants even if there is no ranking function (how to guide the search?).

- Might be no ranking function fulfiling all properties

  We have to generate *quasi-ranking functions*.

  Similar concept as in e.g. Amir Ben-Amram's work.

  May not fulfil some of the properties.
  For instance, boundedness or decreasingness or even both.

Our approach: optimization vs satisfaction

Our solution:

Consider that this is an optimization problem
rather than a satisfaction problem

We want to get a ranking function but if it is not possible
we want to get as much properties as possible.

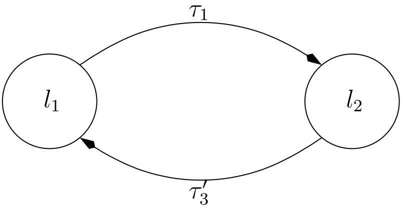
Use different weights to express which properties we prefer

Encode the problem using Max-SMT,

We use again Barcelogic to solve it.

Our approach: Example

Transition system:

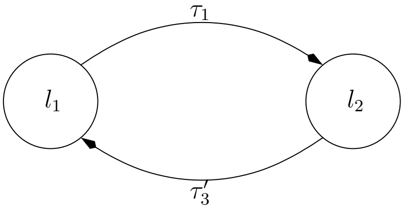


$$
\begin{array}{llllll}
\rho_{\tau_1}: & & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_3'}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

There is no ranking function that fulfils all conditions.

Our approach: Example

Transition system:



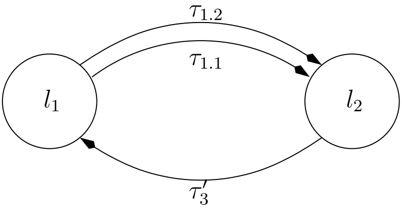
$$\begin{array}{llllll}
\rho_{\tau_1}: & & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_3'}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}$$

$f(x, y, z) = x$ is *non-increasing* and *strict decreasing* for τ_1.

However, it is not **bounded** (*soft*).

Our approach: Example

Transition system:



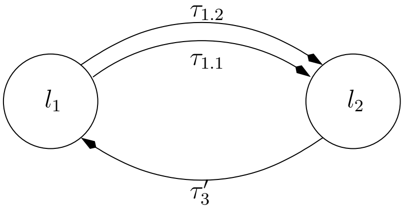
$$
\begin{array}{lllllll}
\rho_{\tau_{1.1}}: & x \geq 0 & y \geq 1, & x' = x - 1, & y' = y, & & z' = z \\
\rho_{\tau_{1.2}}: & x < 0 & y \geq 1, & x' = x - 1, & y' = y, & & z' = z \\
\rho_{\tau_3'}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & & z' = z
\end{array}
$$

$f(x, y, z) = x$ is *non-increasing* and *strict decreasing* for τ_1.

However, it is not **bounded** (*soft*).

Our approach: Example

Transition system:



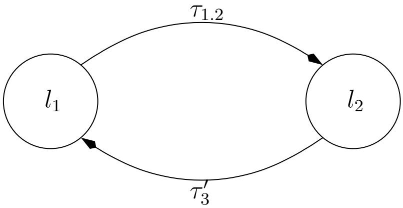
$$
\begin{array}{lllllll}
\rho_{\tau_{1.1}}: & x \geq 0 & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_{1.2}}: & x < 0 & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_3'}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

Now $f(x, y, z) = x$ is a ranking function for $\tau_{1.1}$

We can remove it!

Our approach: Example

Transition system:



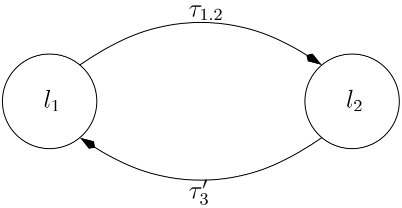
$$
\begin{array}{llllll}
\rho_{\tau_{1.2}}: & x < 0 & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_3'}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}
$$

Now $f(x, y, z) = x$ is a ranking function for $\tau_{1.1}$

We can remove it!

Our approach: Example

Transition system:

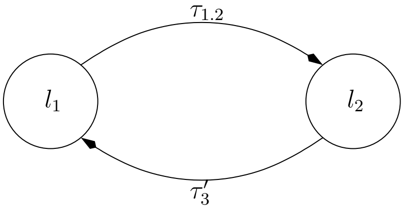


$$\begin{array}{llllll} \rho_{\tau_{1.2}}: & x < 0 & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\ \rho_{\tau'_3}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z \end{array}$$

Finally, $f(x, y, z) = y$ is used to discard τ'_3.

Our approach: Example

Transition system:

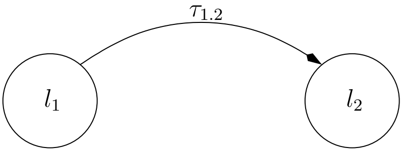


$$\begin{array}{llllll} \rho_{\tau_{1.2}}: & x < 0 & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\ \rho_{\tau'_3}: & y \geq 1, & y \geq z, & x' = x, & y' = x + y, & z' = z \end{array}$$

Finally, $f(x, y, z) = y$ is used to discard τ'_3.
But we need $x < 0$ in l_2, which is a *Termination Implication*

Our approach: Example

Transition system:



$$\rho_{\tau_{1.2}}: \quad x < 0 \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$$

Finally, $f(x, y, z) = y$ is used to discard τ_3'.
But we need $x < 0$ in l_2, which is a *Termination Implication*
We are DONE!

Contributions

- A novel optimization-based method for proving termination.
- New inferred properties: Termination Implications.
- No fixed number of supporting invariants *a priori*.
- Goal-oriented invariant generation.
- Progress in the absence of ranking functions (quasi-ranking functions).
- All these techniques have been implemented in CppInv

Experimental evaluation:

Two sources of benchmarks:

- coming from T2 (Microsoft Cambridge). Thanks!
- code made by undergraduate students taken from a programming learning environment Jutge.org

Experimental evaluation:

Two sources of benchmarks:

- coming from T2 (Microsoft Cambridge). Thanks!
- code made by undergraduate students taken from a programming learning environment Jutge.org In contrast to the standard academic examples the code is:
  - involved and ugly
  - unnecessary conditional statements
  - includes repeated code

Experimental evaluation:

	#ins.	CppInv	T2
Set1	449	238	245
Set2	472	276	279

Table: Results with benchmarks from T2

	#ins.	CppInv	T2
P11655	367	324	328
P12603	149	143	140
P12828	783	707	710
P16415	98	81	81
P24674	177	171	168
P33412	603	478	371

	#ins.	CppInv	T2
P40685	362	324	329
P45965	854	780	793
P70756	280	243	235
P81966	3642	2663	926
P82660	196	174	177
P84219	413	325	243

Table: Results with benchmarks from Jutge.org.

Outline

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Termination analysis
5. Further work

Further work

- Apply our techniques to program synthesis
- Prove non-termination.
- Combine termination and non-termination proofs.
- Improve the non-linear arithmetic solver and the interaction with the invariant generation and termination engine.
- Consider other program properties

Thank you!

Example of students' code

```
int first_occurrence(int x, int A[N]) {
  assume(N > 0);

  int e = 0, d = N - 1, m, pos;
  bool found = false, exit  = false;
  while (e <= d and not exit) {
    m = (e+d)/2;
    if (x > A[m]) {
      if (not found) e = m+1;
      else exit = true;
    } else if (x < A[m]) {
        if (not found) d = m-1;
        else exit = true;
      } else {
          found = true; pos = m; d = m-1;
        }
  }

  if (found) {
    while (x  == A[pos-1]) --pos;
    return pos; }
  return -1;
}
```

```
int first_occurrence(int x, int A[N]) {
  assume(N > 0);

  int l=0, u=N;

  while (l < u) {
    int m = (l+u)/2;
    if (A[m]<x) l=m+1;
    else u=m;
  }

 if (l>=N || A[l]!=x) l=-1;
 return l;
}
```

Farkas' Lemma

Farkas' Lemma:

$$(\forall \overline{x}) \begin{bmatrix} a_{11}x_1 + \cdots + & a_{1n}x_n + & b_1 \leq 0 \\ \vdots & \vdots & \vdots \leq 0 \\ a_{m1}x_1 + \cdots + & a_{mn}x_n + & b_m \leq 0 \end{bmatrix} \Rightarrow \varphi : c_1x_1 + \ldots + c_nx_n + d \leq 0$$

$$\Leftrightarrow$$

$$\exists \lambda_0, \lambda_1, \ldots, \lambda_m \geq 0,$$

$$c_1 = \sum_{i=1}^{m} \lambda_i a_{i1}, \; \ldots \;, c_n = \sum_{i=1}^{m} \lambda_i a_{in}, \; d = (\sum_{i=1}^{m} \lambda_i b_i) - \lambda_0$$