Proving termination through conditional termination

Cristina Borralleras, Marc Brockschmidt, Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell and Albert Rubio

Universitat de Vic
Universitat Politècnica de Catalunya - Barcelona Tech
Microsoft Research, Cambridge

supported by MINECO/FEDER project TIN2015-69175-C4-3-R

Obergurgl, Austria
September 2016

Overview of the talk

Overview of the talk

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

- **Strategy:** Take advantage of powerful arithmetic constraint solvers.

SMT solvers

Constraint-based Program Analysis techniques

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

- **Strategy:** Take advantage of powerful arithmetic constraint solvers.

Max-SMT solvers

Constraint-based Program Analysis techniques

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

- **Strategy:** Take advantage of powerful arithmetic constraint solvers.

Max-SMT solvers

Constraint-based Program Analysis techniques

Goal: Prove termination of imperative programs automatically

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

- **Strategy:** Take advantage of powerful arithmetic constraint solvers.

Max-SMT solvers

Constraint-based Program Analysis techniques

Goal: Prove termination of imperative programs automatically

- Find ranking functions.
- Find supporting invariants.
- How to guide the search!.

Motivation

- **Main Goal:** Build static analysis tools for programmers.
  - Fully automatic.
  - Efficient.
  - Scalable.

- **Strategy:** Take advantage of powerful arithmetic constraint solvers.

Max-SMT solvers

Constraint-based Program Analysis techniques

Goal: Prove termination of imperative programs automatically

- Find ranking functions.
- Find supporting (conditional) invariants.
- How to guide the search!.

Overview of the talk

SMT solving

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model 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

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model 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

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model 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 (Hilbert's 10th problem).
- Reals: decidable (Tarski) **but** algorithms have prohibitive complexity.

SMT solving

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model 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 (Hilbert's 10th problem).
- Reals: decidable (Tarski) **but** algorithms have prohibitive complexity.

Incomplete solvers focus on either satisfiability or unsatisfiability.

SMT solving

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model that satisfies the formula?

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

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

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

Non-linear arithmetic decidability:

- Integers: undecidable (Hilbert's 10th problem).
- Reals: decidable (Tarski) **but** algorithms have prohibitive complexity.

Incomplete solvers focus on either satisfiability or unsatisfiability.

SMT solving

We make extensive use of SMT solvers inside our program analysis tools.

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

Question: Is there any model that satisfies the formula?

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

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

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

- Need to handle large formulas with non-linear arithmetic and complex boolean structure.

- Barcelogic has shown to be the best SMT-solver proving satisfiability of this kind of problems.

Optimization problems

(Weighted) Max-SMT problem

Input: Given an SMT formula $\varphi = C_1 \land \ldots \land C_m$ 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.

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

Overview of the talk

Invariant generation

Definition

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

Invariant generation

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.

Invariant generation

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 focus on inductive invariants.

Constraint-based invariant generation

We inspire ourselves with the constraint-based method [CSS'03].

Assume input programs consist of linear expressions.

Constraint-based invariant generation

We inspire ourselves with the constraint-based method [CSS'03].

Assume input programs consist of 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

We inspire ourselves with the constraint-based method [CSS'03].

Assume input programs consist of linear expressions.

Keys:

- Use a template for candidate invariants.

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

- Impose initiation and consecution conditions obtaining an $\exists\forall$ problem over non-linear arithmetic.

Constraint-based invariant generation

We inspire ourselves with the constraint-based method [CSS'03].

Assume input programs consist of linear expressions.

Keys:

- Use a template for candidate invariants.

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

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

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  c1a + c2s + c3t + d ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  c1a + c2s + c3t + d ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

$\exists\, c_1,\ c_2,\ c_3,\ d \quad \forall\, a,\ s,\ t$

$\underbrace{true \Longrightarrow c_1 \cdot 0 + c_2 \cdot 1 + c_3 \cdot 1 + d \le 0}\ \wedge$ Initiation condition

$$\underbrace{s \le N \wedge c_1 \cdot a + c_2 \cdot s + c_3 \cdot t + d \le 0 \Longrightarrow c_1 \cdot (a+1) + c_2 \cdot (s+t+2) + c_3 \cdot (t+2) + d \le 0}_{\text{consecution condition}}$$

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  c1a + c2s + c3t + d ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

$\exists\ c_1,\ c_2,\ c_3,\ d\quad \forall\ a,\ s,\ t$

$\underbrace{c_2 + c_3 + d \leq 0}\ \wedge$ Initiation condition

$$\underbrace{s \leq N \wedge c_1 \cdot a + c_2 \cdot s + c_3 \cdot t + d \leq 0 \Longrightarrow c_1 \cdot a + c_2 \cdot s + (c_2 + c_3) \cdot t + c_1 + 2c_2 + 2c_3 + d \leq 0}_{\text{consecution condition}}$$

▸ Return

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  c1 a + c2 s + c3 t + d ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

Apply Farkas' Lemma to remove $\forall\ a,\ s,\ t$

Use Barcelogic to solve the non-linear SMT problem!

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  c1 a + c2 s + c3 t + d ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

$$\{c_1 = -2, c_2 = 0, c_3 = 1, d = -1\}$$

Scalar invariant generation: Example

Square root of a natural number N:

```
int isqrt(int N) {
  int a = 0, s = 1, t = 1;
  // Inv:  −2a + 0s + 1t − 1 ≤ 0
  while (s ≤ N) {
    a = a + 1;
    s = s + t + 2;
    t = t + 2;
  }
  return a;
}
```

$$\{c_1 = -2, c_2 = 0, c_3 = 1, d = -1\}$$

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds.

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds.
- but Initiation condition may not hold.

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds. Hard

- but Initiation condition may not hold.

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds. Hard
- but Initiation condition may not hold. Soft

Key: We prefer invariants but we can live with conditional invariants

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds. Hard
- but Initiation condition may not hold. Soft

Key: We prefer invariants but we can live with conditional invariants

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

Conditional invariant generation

Definition

A formula is a conditional (inductive) invariant at a program location if:

- Consecution condition holds. Hard
- but Initiation condition may not hold. Soft

Key: We prefer invariants but we can live with conditional invariants

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

Encode the problem using Max-SMT,

We use Barcelogic to solve it.

Overview of the talk

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.

Ranking functions and Invariants

Basic method: find a single ranking function $f : \text{States} \rightarrow \mathbb{Z}$, 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 $\to \mathbb{Z}$, 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 (conditional) invariants
- Consider a (lexicographic) combination of ranking functions

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.

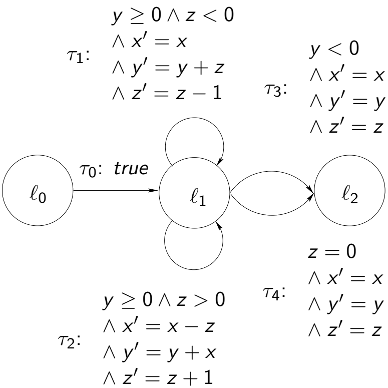
It does not work in practice in many cases

What is (at least) necessary?

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

Running example

```
int main() {
  int x, y, z;
  x = nondet();
  y = nondet();
  z = nondet();
  while (y ≥ 0 && z ≠ 0) {
    if (z < 0) { y = y + z;
                 z = z − 1;
    } else { x = x − z;
             y = y + x;
             z = z + 1;
    }
  }
}
```



Ranking functions and Conditional 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)

Use a linear template for the ranking function as well.

Ranking functions and Invariants: Combined problem

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

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

Back

Ranking functions and Invariants: Combined problem

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

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

Back

1. $\mathcal{I} \wedge \tau_i \Longrightarrow f(x_1, \ldots, x_n) \geq 0$ (bounded)
2. $\mathcal{I} \wedge \tau_i \Longrightarrow f(x_1, \ldots, x_n) > f(x'_1, \ldots, x'_n)$ (strict-decreasing)
3. $\mathcal{I} \wedge \tau_j \Longrightarrow f(x_1, \ldots, x_n) \geq f(x'_1, \ldots, x'_n)$ 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 conditional invariants and ranking functions should be combined in the same optimization problem. ▸ Back

1. $\mathcal{I} \wedge \tau_i \Longrightarrow f(x_1, \ldots, x_n) \geq 0$ (bounded)
2. $\mathcal{I} \wedge \tau_i \Longrightarrow f(x_1, \ldots, x_n) > f(x_1', \ldots, x_n')$ (strict-decreasing)
3. $\mathcal{I} \wedge \tau_j \Longrightarrow f(x_1, \ldots, x_n) \geq f(x_1', \ldots, x_n')$ for all j (non-increasing)

Considering conditional invariants give more chances to the solver

Ranking functions and Invariants: Combined problem

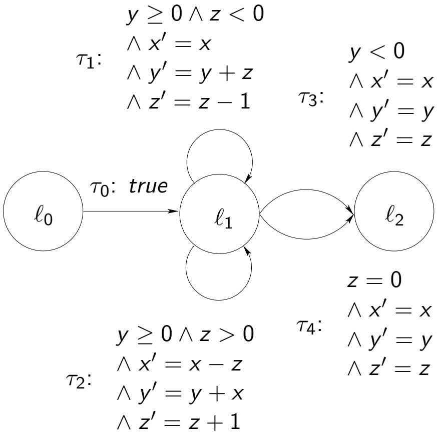
In order to prove properties of the ranking function we may need to generate invariants.

Generation of both conditional invariants and ranking functions should be combined in the same optimization problem. ▸ Back

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

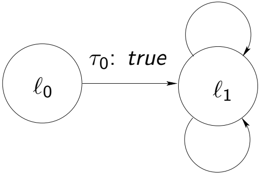
Considering conditional invariants give more chances to the solver

But we get a conditional termination proof


τ_1: $y \geq 0 \wedge z < 0$ $\wedge\ x' = x$ $\wedge\ y' = y + z$ $\wedge\ z' = z - 1$
τ_3: $y < 0$ $\wedge\ x' = x$ $\wedge\ y' = y$ $\wedge\ z' = z$
τ_0: *true*
ℓ_0
ℓ_1
ℓ_2
τ_4: $z = 0$ $\wedge\ x' = x$ $\wedge\ y' = y$ $\wedge\ z' = z$
τ_2: $y \geq 0 \wedge z > 0$ $\wedge\ x' = x - z$ $\wedge\ y' = y + x$ $\wedge\ z' = z + 1$

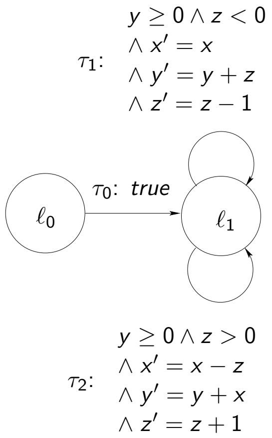
Running example

$$\tau_1: \quad \begin{array}{l} y \geq 0 \land z < 0 \\ \land\, x' = x \\ \land\, y' = y + z \\ \land\, z' = z - 1 \end{array}$$



$$\tau_2: \quad \begin{array}{l} y \geq 0 \land z > 0 \\ \land\, x' = x - z \\ \land\, y' = y + x \\ \land\, z' = z + 1 \end{array}$$

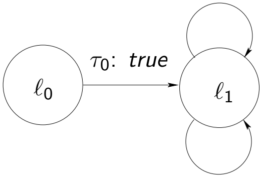
Running example



- $z < 0$ is a conditional invariant at location ℓ_1
- y is a ranking function
  1. τ_1 is bounded and strictly decreasing
  2. τ_2 is disabled

Running example

$$\tau_1: \quad \begin{array}{l} y \geq 0 \wedge z < 0 \\ \wedge\ x' = x \\ \wedge\ y' = y + z \\ \wedge\ z' = z - 1 \end{array}$$



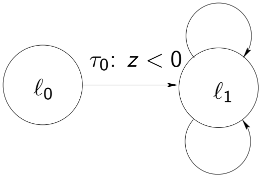
$$\tau_2: \quad \begin{array}{l} y \geq 0 \wedge z > 0 \\ \wedge\ x' = x - z \\ \wedge\ y' = y + x \\ \wedge\ z' = z + 1 \end{array}$$

We have a conditional proof:

The system terminates if the condition $z < 0$ holds at l_0 (or τ_0)

Running example

$$\tau_1: \begin{array}{l} y \geq 0 \land z < 0 \\ \land\, x' = x \\ \land\, y' = y + z \\ \land\, z' = z - 1 \end{array}$$



$$\tau_2: \begin{array}{l} y \geq 0 \land z > 0 \\ \land\, x' = x - z \\ \land\, y' = y + x \\ \land\, z' = z + 1 \end{array}$$

We have a conditional proof:

The system terminates if the condition $z < 0$ holds at ℓ_0 (or τ_0)

Running example: Narrowing

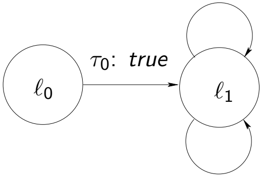
In order to complete the termination proof we have to consider the complementary problem.

Narrow the transitions removing all states that we already now that are terminating.

We can do better than just add the negation of the condition in the entry.

Running example: Narrowing

$$\tau_1: \quad \begin{array}{l} y \geq 0 \wedge z < 0 \\ \wedge\ x' = x \\ \wedge\ y' = y + z \\ \wedge\ z' = z - 1 \end{array}$$



$$\tau_2: \quad \begin{array}{l} y \geq 0 \wedge z > 0 \\ \wedge\ x' = x - z \\ \wedge\ y' = y + x \\ \wedge\ z' = z + 1 \end{array}$$

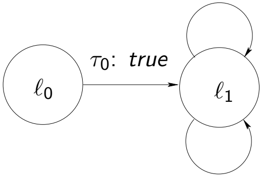
We know more!:

whenever $z < 0$ holds at ℓ_1 the system terminates

Skip

Running example: Narrowing

$$\tau_1: \quad \begin{array}{l} y \geq 0 \wedge z < 0 \\ \wedge\ x' = x \\ \wedge\ y' = y + z \\ \wedge\ z' = z - 1 \end{array}$$



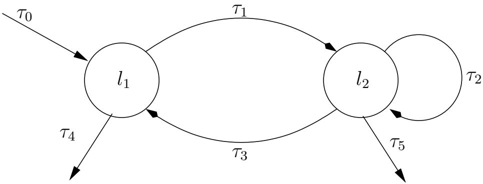
$$\tau_2: \quad \begin{array}{l} y \geq 0 \wedge z > 0 \\ \wedge\ x' = x - z \\ \wedge\ y' = y + x \\ \wedge\ z' = z + 1 \end{array}$$

Narrow the transition system according to this:

whenever $z < 0$ holds at ℓ_1 the system terminates

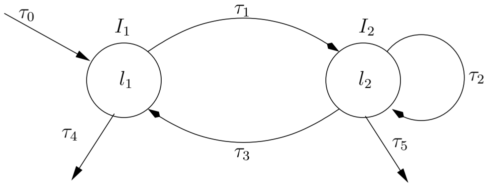
Narrowing

Assume we have the following transition system:



Narrowing

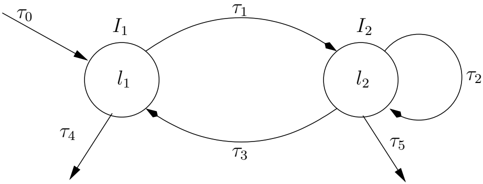
After sending the problem to our Max-SMT solver we get:



- Conditional invariant I_1 at location l_1.
- Conditional invariant I_2 at location l_2.

Narrowing

After sending the problem to our Max-SMT solver we get:

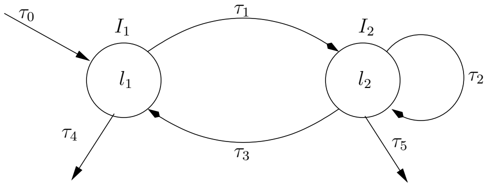


- Conditional invariant I_1 at location l_1.
- Conditional invariant I_2 at location l_2.

- If I_1 holds in location l_1 then I_2 holds in location l_2.
- I_2 is preserved in l_2.
- If I_2 holds in location l_2 then I_1 holds in location l_1.
- If I_2 holds in l_2 and I_2 holds in l_2 then it terminates.

Narrowing

After sending the problem to our Max-SMT solver we get:



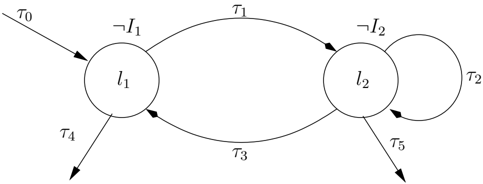
- Conditional invariant I_1 at location l_1.
- Conditional invariant I_2 at location l_2.

Therefore

- If I_1 holds in location l_1 we are done.
- If I_2 holds in location l_2 we are done.

Narrowing

After narrowing

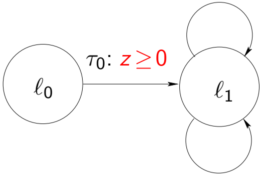


Remains to be proved

Therefore

- The entry τ_0 is narrowed with $\neg I'_1$
- Transition τ_1 is narrowed with $\neg I_1$ and $\neg I'_2$
- Transition τ_2 is narrowed with $\neg I_2$ and $\neg I'_2$
- Transition τ_3 is narrowed with $\neg I_2$ and $\neg I'_1$

$$\tau_1:\quad \begin{array}{l} z \geq 0 \\ y \geq 0 \wedge z < 0 \\ \wedge\, x' = x \\ \wedge\, y' = y + z \\ \wedge\, z' = z - 1 \end{array}$$



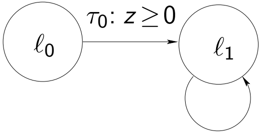
$$\tau_2:\quad \begin{array}{l} z \geq 0 \\ y \geq 0 \wedge z > 0 \\ \wedge\, x' = x - z \\ \wedge\, y' = y + x \\ \wedge\, z' = z + 1 \end{array}$$

Narrow the transition system according to this:

whenever $z < 0$ holds at ℓ_1 the system terminates

Running example. Narrowing

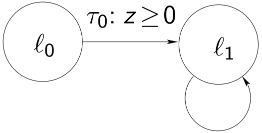
After simplifying the transition system we get:



$$\tau_2: \quad \begin{array}{l} y \geq 0 \wedge z > 0 \\ \wedge\, x' = x - z \\ \wedge\, y' = y + x \\ \wedge\, z' = z + 1 \end{array}$$

Running example. Narrowing

After simplifying the transition system we get:

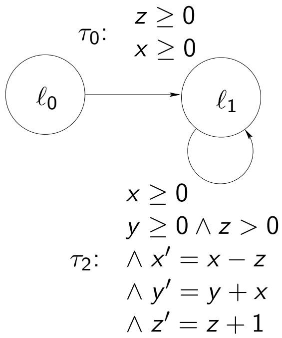


$$\tau_2: \quad \begin{array}{l} y \geq 0 \wedge z > 0 \\ \wedge\, x' = x - z \\ \wedge\, y' = y + x \\ \wedge\, z' = z + 1 \end{array}$$

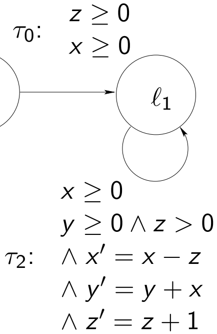
Conditionally terminates:

- $x < 0$ is a conditional invariant at location ℓ_1
- y is a ranking function
    1. τ_2 is bounded and strictly decreasing

Narrowing again with the complement of $x < 0$ we get:



Narrowing again with the complement of $x < 0$ we get:



Which terminates with x as a ranking function

Advantages of conditional termination

Conditional termination provides a natural way of

proving termination by cases

Advantages of conditional termination

Conditional termination provides a natural way of

proving termination by cases

The Max-SMT solver tries to get a direct proof

Advantages of conditional termination

Conditional termination provides a natural way of

proving termination by cases

The Max-SMT solver tries to get a direct proof

but if this is not possible (in a given time limit)

it can provide a conditional proof (soft constraints) which give us a progress.

Advantages of conditional termination

Conditional termination provides a natural way of

proving termination by cases

The Max-SMT solver tries to get a direct proof

but if this is not possible (in a given time limit)

it can provide a conditional proof (soft constraints) which give us a progress.

An additional advantage (key in some case):

If we cannot prove termination of the narrowed transition system

we can use it to try to prove non-termination

as the non-terminating execution (if any) should be there!

Overview of the talk

Scalable Termination Analysis

Aim: prove termination in large programs (several consecutive loops).

New approach:

1. Obtain a conditional termination proof.
2. Check (compositionally) the condition as a Safety property.

Simple example:

```
assume(x > y && y ≥ 0);
while (y > 0) {
  x = x - 1;
  y = y - 1;
}
while (y < 0) {
  y = y + x;
}
```

Scalable Termination Analysis

Aim: prove termination in large programs (several consecutive loops).

New approach:

1. Obtain a conditional termination proof.
2. Check (compositionally) the condition as a Safety property.

Simple example:

```
assume(x > y && y ≥ 0);
while (y > 0) {
  x = x - 1;
  y = y - 1;
}
assert(x > 0); Rank: -y
while (y < 0) {
  y = y + x;
}
```

Compositional analysis through Conditional Termination

Aim: verify termination in large programs (several consecutive loops).

Key ideas:

- Generate conditional proofs:
  - Find conditional invariants implying termination
- Check the condition as a Safety property of previous loops.

Compositional analysis through Conditional Termination

Aim: verify termination in large programs (several consecutive loops).

Key ideas:

- Generate conditional proofs:
  - Find conditional invariants implying termination
- Check the condition as a Safety property of previous loops.
- In case of failure of the Safety checker
  Narrow the loop and try again!

Compositional analysis through Conditional Termination

Aim: verify termination in large programs (several consecutive loops).

Key ideas:

- Generate conditional proofs:
  - Find conditional invariants implying termination
- Check the condition as a Safety property of previous loops.
- In case of failure of the Safety checker
  Narrow the loop and try again!

We can handle every loop (or SCC in general) independently

Experiments

Our techniques have been implemented in VeryMax(already presented)

These techniques can be highly parallelized (sharing few information).

Compared to last year competitors in TermComp on (335) Integer C programs

Tool	Terminating
AProVE	208(5)
HipTNT+	210(5)
UltimateBuchiAutomizer	207
VeryMax	213

Overview of the talk

Conclusions

Two main conclusions:

- Using SMT and Max-SMT, automatic generation of conditional invariants and ranking function becomes feasible.
- In constraint-based program analysis it is often better to consider that we have optimization problems rather than satisfiability problems!

Under development:

- Combine conditional termination and non-termination analysis.
- Use conditional termination to provide witness of termination. For instance, it has applications to check reachability.

Future developments?:

- Generate linear upper bounds

Thank you!