Compositional Program Analysis using Max-SMT

Albert Rubio

Cristina Borralleras, Marc Brockschmidt, Daniel Larraz, Albert Oliveras, José Miguel Rivero and Enric Rodríguez-Carbonell

Universitat Politècnica de Catalunya - Barcelona Tech

UCM Seminar
March 2018
Overview of the talk

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Compositional safety verification
5. VeryMax Tool
6. Conclusions and current work
Overview of the talk

1 Introduction

2 SMT/Max-SMT solving

3 Invariant generation

4 Compositional safety verification

5 VeryMax Tool

6 Conclusions and current work
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:** Verify safety and liveness properties of programs
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: Verify safety and liveness properties of programs

Challenge: discover (loop) invariants.

How can we guide the search?
Overview of the talk

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Compositional safety verification
5. VeryMax Tool
6. Conclusions and current work
We make extensive use of SMT solvers inside our program analysis tools.

**SAT and SMT solvers gain efficiency by:**

- addressing only (expressive enough) **decidable fragments** of a certain logic
- incorporate **domain-specific** reasoning, e.g:
  - arithmetic reasoning
  - equality
  - data structures (arrays, lists, stacks, ...)

**SAT**: use **propositional logic** as the formalization language
  - high degree of efficiency
    - expressive (all NP-complete) but involved encodings

**SMT**: **propositional logic** + **domain-specific** reasoning
  - improves the expressivity
    - certain (but acceptable) loss of efficiency
Need and Applications of SMT

- Some problems are more naturally expressed in other logics than propositional logic, e.g:
  - Software verification needs reasoning about equality, arithmetic, data structures, ...

- SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory

- Example (Equality with Uninterpreted Functions – EUF):
  $g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d$

- Wide range of applications:
  - Predicate abstraction
  - Model checking
  - Scheduling
  - Test generation
  - ...

Albert Rubio (UPC)
Theories of Interest - Arithmetic

- Very useful for obvious reasons

- Restricted fragments support more efficient methods:
  - **Bounds**: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - **Difference logic**: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - **UTVPI**: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - **Linear arithmetic**, e.g.: $2x - 3y + 4z \leq 5$
  - **Non-linear arithmetic**, e.g.: $2xy + 4xz^2 - 5y \leq 10$
  - Variables are either reals or integers
SMT problems

**Input:** Given a *boolean* formula $\varphi$ over some *theory* $T$.

**Question:** Is there any *interpretation* (*solution*) that satisfies the formula?

**Example:** $T = \text{linear integer/real arithmetic.}$

\[
(x < 0 \lor x \leq y \lor y < z) \land (x \geq 0) \land (x > y \lor y < z)
\]

\[
\{x = 1, y = 0, z = 2\}
\]
SMT problems

**Input:** Given a boolean formula \( \varphi \) over some theory \( T \).

**Question:** Is there any *interpretation (solution)* that satisfies the formula?

**Example:** \( T = \) linear integer/real arithmetic.

\[
(x < 0 \lor x \leq y \lor y < z) \land (x \geq 0) \land (x > y \lor y < z)
\]

\[
\{x = 1, y = 0, z = 2\}
\]
Input: Given a boolean formula $\phi$ over some theory $T$.

Question: Is there any interpretation (solution) that satisfies the formula?

Example: $T = \text{linear integer/real arithmetic}$. 

$$\begin{align*}
(x < 0 \lor x \leq y \lor y < z) \land (x \geq 0) \land (x > y \lor y < z)
\end{align*}$$

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

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

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

**Input:** Given a *boolean* formula \( \varphi \) over some *theory* \( T \).

**Question:** Is there any solution that satisfies the formula?

**Example:** \( T = \text{non-linear (polynomial) integer/real arithmetic} \).

\[
(x^2 + y^2 > 2 \lor x \cdot z \leq y \lor y \cdot z < z^2) \land (x > y \lor 0 < z)
\]

\[
\{x = 0, \ y = 1, \ z = 1\}
\]
Non-linear SMT solving

**Input:** Given a boolean formula $\varphi$ over some theory $T$.

**Question:** Is there any solution that satisfies the formula?

**Example:** $T = \text{non-linear (polynomial) integer/real arithmetic.}$

\[
(x^2 + y^2 > 2 \lor x \cdot z \leq y \lor y \cdot z < z^2) \land (x > y \lor 0 < z)
\]

\[
\{x = 0, \ y = 1, \ z = 1\}
\]
Non-linear SMT solving

**Input:** Given a *boolean* formula $\varphi$ over some *theory* $T$.

**Question:** Is there any solution that satisfies the formula?

**Example:** $T = \text{non-linear (polynomial) integer/real arithmetic.}$

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

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

**Non-linear arithmetic decidability:**

- *Integers:* undecidable (Hilbert’s 10th problem).
- *Reals:* decidable (Tarski) but algorithms have prohibitive complexity.
Non-linear SMT solving

**Input:** Given a boolean formula $\varphi$ over some theory $T$.

**Question:** Is there any solution that satisfies the formula?

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

\[(x^2 + y^2 > 2 \lor x \cdot z \leq y \lor y \cdot z < z^2) \land (x > y \lor 0 < z)\]

\[\{x = 0, \ y = 1, \ z = 1\}\]

**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.
Non-linear SMT solving

Input: Given a boolean formula $\varphi$ over some theory $T$.

Question: Is there any solution that satisfies the formula?

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

\[
(x^2 + y^2 > 2 \lor x \cdot z \leq y \lor y \cdot z < z^2) \land (x > y \lor 0 < z)
\]

\[
\{x = 0, y = 1, z = 1\}
\]

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.
Solving non-linear SMT formulas

- 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.

- Barcelogic can handle Max-SMT formulas (over non-linear arithmetic) as well.
Overview of the talk

1. Introduction

2. SMT/Max-SMT solving

3. Invariant generation

4. Compositional safety verification

5. VeryMax Tool

6. Conclusions and current work
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.
We inspire ourselves with the constraint-based method [CSS’03]. Assume input programs consist of linear expressions.
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.

\[ c_1x_1 + \ldots + c_nx_n + d \leq 0 \]

- 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:

```c
int isqrt(int N) {  //integer square root
    int a = 0, s = 1, t = 1;
    // Inv:  \( c_1 a + c_2 s + c_3 t + d \leq 0 \)
    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 $N$:

```c
int isqrt(int N) { //integer square root
    int a = 0, s = 1, t = 1;
    // Inv: $c_1 a + c_2 s + c_3 t + d \leq 0$
    while (s $\leq$ N) {
        a = a + 1;
        s = s + t + 2;
        t = t + 2;
    }
    return a;
}
```

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

$$true \implies c_1 \cdot 0 + c_2 \cdot 1 + c_3 \cdot 1 + d \leq 0 \land \text{Initiation condition}$$

$$s \leq N \land c_1 \cdot a + c_2 \cdot s + c_3 \cdot t + d \leq 0 \implies c_1 \cdot (a+1) + c_2 \cdot (s+t+2) + c_3 \cdot (t+2) + d \leq 0$$

consecution condition
Scalar invariant generation: Example

Square root of a natural number N:

```c
int isqrt(int N) { //integer square root
    int a = 0, s = 1, t = 1;
    // Inv:  \( c_1a + c_2s + c_3t + d \leq 0 \)
    while (s <= N) {
        a = a + 1;
        s = s + t + 2;
        t = t + 2;
    }
    return a;
}
```

\[ \exists c_1, c_2, c_3, d \quad \forall a, s, t \]
\[ c_2 + c_3 + d \leq 0 \quad \land \quad \text{Initiation condition} \]
\[ s \leq N \land c_1 \cdot a + c_2 \cdot s + c_3 \cdot t + d \leq 0 \implies c_1 \cdot a + c_2 \cdot s + (c_2 + c_3) \cdot t + c_1 + 2c_2 + 2c_3 + d \leq 0 \quad \text{consecution condition} \]
Scalar invariant generation: Example

Square root of a natural number N:

```c
int isqrt(int N) { //integer square root
    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;
}
```

Apply Farkas’ Lemma to remove ∀ a, s, t

Use Barcelogic to solve the non-linear SMT problem!
Scalar invariant generation: Example

Square root of a natural number N:

```c
int isqrt(int N) { //integer square root
    int a = 0, s = 1, t = 1;
    // Inv: $c_1a + c_2s + c_3t + d \leq 0$
    while (s \leq 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!

$$\{c_1 = -2, c_2 = 0, c_3 = 1, d = -1\}$$
Scalar invariant generation: Example

Square root of a natural number N:

```c
int isqrt(int N) { //integer square root
    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;
}
```

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

Use Barcelogic to solve the non-linear SMT problem!

$$\{c_1 = -2, c_2 = 0, c_3 = 1, d = -1\}$$
Scalar invariant generation: Example

Square root of a natural number N:

```c
int isqrt(int N) {
    int a = 0, s = 1, t = 1;
    // Inv: t ≤ 2a + 1
    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!

$$\{c_1 = -2, c_2 = 0, c_3 = 1, d = -1\}$$
We have used this approach for:

- Array invariant generation. [VMCAI2013]
- Termination analysis using Max-SMT. [FMCAD2013] (inspired by [BMS’05])
  
  Key notion: quasi-ranking functions

- Non-Termination analysis using Max-SMT. [CAV2014]
  
  Key notion: quasi-invariant/conditional invariants
Overview of the talk

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Compositional safety verification
5. VeryMax Tool
6. Conclusions and current work
Safety verification

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

Our approach: **Goal oriented.** Starts from the postcondition. **Automatically generate intermediate assertions!!**

Simple example:

```java
while (j>0) {
    j--;
    i++;
}

while (i>0) {
    x=x+5;
    i--;
}
assert(x≥0);
```
Safety verification

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

Our approach: **Goal oriented.** Starts from the postcondition. **Automatically generate intermediate assertions!!**

Simple example:

```java
while (j>0) {
    j--;
    i++;
}
assert(x + 5*i >=0);
while (i>0) {
    x=x+5;
    i--;
}
assert(x>=0);
```
Safety verification

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

Our approach: **Goal oriented.** Starts from the postcondition. **Automatically generate intermediate assertions!!**

Simple example:

```c
assert(j>=0 and x + 5*(i+j) >=0);
while (j>0) {
    j--;
    i++;
}
assert(x + 5*i >=0);
while (i>0) {
    x=x+5;
    i--;
}
assert(x>=0);
```
Definition
A formula is a conditional (inductive) invariant at a program location if:
- Consecution condition holds.
Definition
A formula is a conditional (inductive) invariant at a program location if:
- Consecution condition holds.
- but Initiation condition may not hold.
Definition
A formula is a **conditional (inductive) invariant** at a program location if:

- Consecution condition holds. **Hard**

- but Initiation condition may not hold.
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
Altogether we have:

- **Initiation condition (soft)**
- **Consecution condition (hard)**
Altogether we have:

- **Initiation condition (soft)**
- **Consecution condition (hard)**
- Plus **implication of the Postcondition (hard)**
Altogether we have:

- **Initiation condition (soft)**
- **Consecution condition (hard)**
- Plus **implication of the Postcondition (hard)**

Solve the problem with a Max-SMT solver (*we use Barcelogic*)
Altogether we have:

- **Initiation condition (soft)**
- **Consecution condition (hard)**
- **Plus implication of the Postcondition (hard)**

Solve the problem with a Max-SMT solver (we use Barcelogic)

If initiation condition holds we are done
 Altogether we have:

- **Initiation condition (soft)**
- **Consecution condition (hard)**
- **Plus implication of the Postcondition (hard)**

Solve the problem with a Max-SMT solver *(we use Barcelogic)*

If initiation does not hold we have a **new** Postcondition for previous code
Altogether we have:

- Initiation condition (soft)
- Consecution condition (hard)
- Plus implication of the Postcondition (hard)

Solve the problem with a Max-SMT solver (we use Barcelogic)

If initiation does not hold we have a new Postcondition for previous code call recursively to the safety checker
In case of failure of the recursive call to the safety checker

- Add the negation of the conditional invariant in the corresponding locations: *Narrow the loop*

- Try to prove the Postcondition again (with more information).
Narrowing loops: Recovering from failures

Simple example:

```c
int x=-50;
int y=nondet();

while (x<0) {
    x = x + y;
    y = y + 1;
}

assert(y>0);
```
Narrowing loops: Recovering from failures

Simple example:

```c
int x=-50;
int y=nondet();

assert(y>0);
while (x<0) {
    x = x + y;
    y = y + 1;
}

assert(y>0);
```
Safety verification

Narrowing loops: **Recovering from failures**

Simple example:

```c
int x=-50;
int y=nondet();

assert(y>0); // Conditional invariant
while (x<0) {
    x = x + y;
    y = y + 1;
}

assert(y>0);
```

Albert Rubio (UPC)  
Program Analysis using Max-SMT  
UCM Seminar 2018
Safety verification

Narrowing loops: Recovering from failures

Simple example:

```c
int x = -50;
int y = nondet();
assume(!(y > 0));

while (x < 0) {
    assume(!(y > 0));
    x = x + y;
    y = y + 1;
}
assert(y > 0);
```
Narrowing loops: Recovering from failures

Simple example:

```
int x=-50;
int y=nondet();
assume(y<=0);

while (x<0) {
    assume(y<=0);
    x = x + y;
    y = y + 1;
}
assert(y>0);
```
Narrowing loops: Recovering from failures

Simple example:

```c
int x=-50;
int y=nondet();
assume(y<=0);
assert(x<0); // Invariant
while (x<0) {
    assume(y<=0);
    x = x + y;
    y = y + 1;
}
assert(y>0); // Unreachable!
```
Experiments

Our techniques have been implemented in a tool called VeryMax

- 217 programs taken from *Numerical Recipes in C++*
- up to 284 lines of code
- 6452 safety problems
- 6106 can be proved
- highly parallelizable
- FMCAD 2015
Given a program (loop), obtain a (pre-)condition ensuring its termination.
Conditional Termination

Given a program (loop), obtain a (pre-)condition ensuring its termination.

Our approach:

- Find ranking functions using (linear) templates
- Find supporting conditional invariants (like before)
Conditional Termination

Given a program (loop), obtain a (pre-)condition ensuring its termination.

Our approach:

- Find ranking functions using (linear) templates
- Find supporting conditional invariants (like before)

Encode the problem with Max-SMT
Conditional Termination

Given a program (loop), obtain a (pre-)condition ensuring its termination.

Our approach:

- Find ranking functions using (linear) templates
- Find supporting conditional invariants (like before)

Encode the problem with Max-SMT

Using conditional termination we can

- prove termination by cases.
- combine termination and non-termination analysis (in parallel).
- prove termination of consecutive loops in a compositional way.
```c
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;
        }
    }
}
```

### Labeled Transition System

<table>
<thead>
<tr>
<th>Labeled Transition</th>
<th>Transition Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\ell_0) (\rightarrow) (\ell_1)</td>
<td>(y \geq 0 \land z &gt; 0) &lt;br&gt; (\land x' = x - z) &lt;br&gt; (\land y' = y + x) &lt;br&gt; (\land z' = z + 1)</td>
</tr>
<tr>
<td>(\ell_2) (\rightarrow) (\ell_1)</td>
<td>(y &lt; 0) &lt;br&gt; (\land x' = x) &lt;br&gt; (\land y' = y) &lt;br&gt; (\land z' = z)</td>
</tr>
<tr>
<td>(\ell_1) (\rightarrow) (\ell_0)</td>
<td>(y \geq 0 \land z &lt; 0) &lt;br&gt; (\land x' = x) &lt;br&gt; (\land y' = y + z) &lt;br&gt; (\land z' = z - 1)</td>
</tr>
<tr>
<td>(\ell_1) (\rightarrow) (\ell_2)</td>
<td>(z = 0) &lt;br&gt; (\land x' = x) &lt;br&gt; (\land y' = y) &lt;br&gt; (\land z' = z)</td>
</tr>
</tbody>
</table>

Albert Rubio (UPC)  
Program Analysis using Max-SMT  
UCM Seminar 2018  
26 / 44
In order to discard a transition $\tau_i$ we need to find a ranking function $f$ over the integers such that:

1. $\tau_i \implies f(x_1, \ldots, x_n) \geq 0$ (bounded)

2. $\tau_i \implies f(x_1, \ldots, x_n) > f(x'_1, \ldots, x'_n)$ (strict-decreasing)

3. $\tau_j \implies f(x_1, \ldots, x_n) \geq f(x'_1, \ldots, x'_n)$ for all $j$ (non-increasing)

Use a linear template for the ranking function as well.
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.
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.

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

1. \( I \land \tau_i \implies f(x_1, \ldots, x_n) \geq 0 \)  
   (bounded)

2. \( I \land \tau_i \implies f(x_1, \ldots, x_n) > f(x'_1, \ldots, x'_n) \)  
   (strict-decreasing)

3. \( I \land \tau_j \implies f(x_1, \ldots, x_n) \geq f(x'_1, \ldots, x'_n) \) for all \( j \)  
   (non-increasing)

Considering conditional invariants give more chances to the solver
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.

1. $I \land \tau_i \implies f(x_1, \ldots, x_n) \geq 0$ (bounded)

2. $I \land \tau_i \implies f(x_1, \ldots, x_n) > f(x'_1, \ldots, x'_n)$ (strict-decreasing)

3. $I \land \tau_j \implies f(x_1, \ldots, x_n) \geq f(x'_1, \ldots, x'_n)$ for all $j$ (non-increasing)

Considering conditional invariants give more chances to the solver

But we get a **conditional** termination proof
Running example

\[ y \geq 0 \land z > 0 \]
\[ \land x' = x - z \]
\[ \land y' = y + x \]
\[ \land z' = z + 1 \]

\[ \tau_1: \]

\[ y < 0 \]
\[ \land x' = x \]
\[ \land y' = y \]
\[ \land z' = z \]

\[ \tau_3: \]

\[ y \geq 0 \land z < 0 \]
\[ \land x' = x \]
\[ \land y' = y + z \]
\[ \land z' = z - 1 \]

\[ \tau_2: \]

\[ \tau_0: \text{true} \]

\[ \tau_4: \]

\[ z = 0 \]
\[ \land x' = x \]
\[ \land y' = y \]
\[ \land z' = z \]
Running example

\( \tau_0: \text{true} \)

\( \ell_0 \quad \rightarrow \quad \ell_1 \)

\( \tau_1: \)

\[ y \geq 0 \land z > 0 \]
\[ \land x' = x - z \]
\[ \land y' = y + x \]
\[ \land z' = z + 1 \]

\( \tau_2: \)

\[ y \geq 0 \land z < 0 \]
\[ \land x' = x \]
\[ \land y' = y + z \]
\[ \land z' = z - 1 \]
Running example

\[ \ell_0 \rightarrow \ell_1 \]

\( \tau_0: \text{true} \)

\( \tau_1: y \geq 0 \land z > 0 \land x' = x - z \land y' = y + x \land z' = z + 1 \)

\( \tau_2: y \geq 0 \land z < 0 \land x' = x \land y' = y + z \land z' = z - 1 \)

- \( z < 0 \) is a conditional invariant at location \( \ell_1 \)
- \( y \) is a ranking function
  - 1. \( \tau_1 \) is disabled
  - 2. \( \tau_2 \) is bounded and strictly decreasing
Running example

We have a conditional proof:

The system terminates if the condition \( z < 0 \) holds at \( \ell_0 \)
We have a conditional proof:

The system terminates if the condition $z < 0$ holds at $\ell_0$ (or $\tau_0$)
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 know that are terminating.

We can do better than just add the negation of the condition in the entry.
Running example: Narrowing

\( \tau_0: \text{true} \)

\( \ell_0 \rightarrow \ell_1 \)

\( \tau_1: \)

\[\begin{align*}
y &\geq 0 \land z > 0 \\
&\land x' = x - z \\
&\land y' = y + x \\
&\land z' = z + 1
\end{align*}\]

\( \tau_2: \)

\[\begin{align*}
y &\geq 0 \land z < 0 \\
&\land x' = x \\
&\land y' = y + z \\
&\land z' = z - 1
\end{align*}\]

We know more!

whenever \( z < 0 \) holds at \( \ell_1 \) the system terminates
Running example: Narrowing

\[ y \geq 0 \land z > 0 \]
\[ \land x' = x - z \]
\[ \land y' = y + x \]
\[ \land z' = z + 1 \]

\[ \tau_1: \]

\[ y \geq 0 \land z < 0 \]
\[ \land x' = x \]
\[ \land y' = y + z \]
\[ \land z' = z - 1 \]

\[ \tau_2: \]

\[ \tau_0: \text{true} \]

Narrow the transition system according to this:

whenever \( z < 0 \) holds at \( \ell_1 \) the system terminates
Running example: Narrowing

\[ \ell_0 \rightarrow \ell_1 \]

\[ \tau_0: \ z \geq 0 \]

\[ \tau_1:\ ^\wedge x' = x - z \]
\[ \wedge y' = y + x \]
\[ \wedge z' = z + 1 \]

\[ \tau_2:\ ^\wedge x' = x \]
\[ \wedge y' = y + z \]
\[ \wedge z' = z - 1 \]

\textbf{Narrow} the transition system according to this:

whenever \( z < 0 \) holds at \( \ell_1 \) the system terminates
Running example. Narrowing

After simplifying the transition system we get:

\[ \ell_0 \xrightarrow{\tau_0} \ell_1 \]

\[ \tau_1: \]

\[ z \geq 0 \]
\[ y \geq 0 \land z > 0 \]
\[ x' = x - z \]
\[ y' = y + x \]
\[ z' = z + 1 \]
Running example. Narrowing

After simplifying the transition system we get:

\[ \begin{align*}
\tau_0 &: \quad z \geq 0 \\
\tau_1 &: \quad y \geq 0 \land z > 0 \\
& \quad \land x' = x - z \\
& \quad \land y' = y + x \\
& \quad \land z' = z + 1
\end{align*} \]

Conditionally terminates:

- \( x < 0 \) is a conditional invariant at location \( \ell_1 \)
- \( y \) is a ranking function
  - \( \tau_1 \) is bounded and strictly decreasing
Narrowing again with the complement of $x < 0$ we get:

\begin{align*}
z &\geq 0 \\
x &\geq 0 \\
y &\geq 0 \land z > 0 \\
&\land x' = x - z \\
&\land y' = y + x \\
&\land z' = z + 1
\end{align*}

\[ \tau_0: \quad z \geq 0 \]
\[ \tau_0: \quad x \geq 0 \]

\[ \tau_1: \quad z \geq 0 \]
\[ \tau_1: \quad x' = x - z \land y' = y + x \land z' = z + 1 \]

Diagram:

- $\ell_0 \rightarrow \ell_1$
- $\tau_0: z \geq 0$
- $\tau_0: x \geq 0$
- $\tau_1: z \geq 0$
- $\tau_1: x' = x - z \land y' = y + x \land z' = z + 1$
Narrowing again with the complement of \( x < 0 \) we get:

\[
\begin{align*}
\ell_0: & \quad z \geq 0 \\
& \quad x \geq 0 \\
& \quad y \geq 0 \land z > 0 \\
& \quad \tau_1: \quad \land x' = x - z \\
& \quad \land y' = y + x \\
& \quad \land z' = z + 1
\end{align*}
\]

Which terminates with \( x \) as a ranking function.
Compositional Termination Analysis

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

New approach:

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

Simple example:

```plaintext
assume(x > y && y ≥ 0);
while (y > 0) {
    x = x - 1;
    y = y - 1;
}
while (y < 0) {
    y = y + x;
}
```
Aim: prove termination of large programs (several consecutive loops).

New approach:

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

Simple example:

```plaintext
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;
}
```
Aim: verify termination in large programs (several consecutive loops).

New approach:

1. Obtain a conditional termination proof.
2. Check the condition as a Safety property.
Aim: verify termination in large programs (several consecutive loops).

New approach:

1. Obtain a conditional termination proof.
2. Check the condition as a Safety property.
3. In case of failure of the Safety checker
   Narrow the loop and try again!
Aim: verify termination in large programs (several consecutive loops).

New approach:

1. Obtain a conditional termination proof.
2. Check the condition as a Safety property.
3. In case of failure of the Safety checker
   Narrow the loop and try again!

We can handle every loop independently
Experiments

Our techniques have been implemented in VeryMax

Results:

- won the C Integer programs categories of the Termination Competition in 2016 and 2017

- Comparison with tools at the Termination category of SVComp. On the 358 benchmarks not involving recursion or pointers (273)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Term</th>
<th>NTerm</th>
<th>Fail</th>
<th>TO</th>
<th>Total (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AProVE</td>
<td>222</td>
<td>76</td>
<td>41</td>
<td>19</td>
<td>10235.44</td>
</tr>
<tr>
<td>SeaHorn</td>
<td>189</td>
<td>75</td>
<td>22</td>
<td>72</td>
<td>34760.69</td>
</tr>
<tr>
<td>UltimateBuchiA</td>
<td>224</td>
<td>103</td>
<td>25</td>
<td>6</td>
<td>7882.13</td>
</tr>
<tr>
<td>VeryMax</td>
<td>231</td>
<td>101</td>
<td>26</td>
<td>0</td>
<td>2444.29</td>
</tr>
</tbody>
</table>

See our paper at TACAS 2017 for more details.
Overview of the talk

1 Introduction
2 SMT/Max-SMT solving
3 Invariant generation
4 Compositional safety verification
5 VeryMax Tool
6 Conclusions and current work
Two phases

1. Front-end. From source programs to VeryMax Transition Systems
2. Static Analysis Tools
VeryMax static analysis tools

VERYMAX TRANSITION SYSTEM

SAFETY CHECK

REACHABILITY CHECK

TERMINATION ANALYSIS

NON TERMINATION ANALYSIS

CONDITIONAL INVARIANT + RANKING FUNCTION GENERATOR

MAX-SMT SOLVER

VeryMax

Albert Rubio (UPC)

UCM Seminar 2018
VeryMax can

1. check safety properties
2. check reachability properties
3. prove termination
4. prove non-termination
Overview of the talk

1. Introduction
2. SMT/Max-SMT solving
3. Invariant generation
4. Compositional safety verification
5. VeryMax Tool
6. Conclusions and current work
Conclusions

Two main conclusions:

- Using SMT and Max-SMT, automatic generation of needed (conditional) invariants can be made efficiently.
- Scalable program verification becomes feasible

Other potential applications of conditional analysis and Max-SMT:

- Analysis of concurrent/distributed systems.
- Program synthesis.
- Program repair (minimize changes).
- Using Max-SMT we can express preferences among possible solutions
Thank you!