Overview of the Talk

- Non-linear constraint solving
  - Review of [JAR’12]
  - Alternative Max-SMT approach

- Constraint-based termination analysis
  - Review of program termination and constraint-based program analysis
  - Using Max-SMT for termination analysis
  - Implementation and experiments

- Conclusions & future work
Non-linear Constraint Solving

- **Problem:** Given a quantifier-free formula $F$ containing polynomial inequality atoms, is $F$ satisfiable?

- **Applications:** system analysis and verification, ... Here, focus will be on *termination of imperative programs*

- In $\mathbb{Z}$: *undecidable* (Hilbert’s 10th problem)

- In $\mathbb{R}$: decidable, even with quantifiers (Tarski)
  But algorithms have *prohibitive complexity*

- **Goal:** Can we have a procedure that works “well” in practice?
Review of [JAR’12]

- Our method is aimed at proving satisfiability in the integers (as opposed to finding non-integer solutions, or proving unsatisfiability)

- **Basic idea:** use bounds on integer variables to linearize the formula

- **Refinement:** analyze unsatisfiable cores to enlarge bounds (and sometimes even prove unsatisfiability)
Translating into Linear Arithmetic

- For any formula there is an equisatisfiable one of the form

\[ F \land (\bigwedge_i y_i = M_i) \]

where \( F \) is linear and each \( M_i \) is non-linear

- Example

\[ u^4 v^2 + 2u^2 vw + w^2 \leq 4 \land 1 \leq u, v, w \leq 2 \]

\[ x_{u^4 v^2} + 2x_{u^2 vw} + x_{w^2} \leq 4 \land 1 \leq u, v, w \leq 2 \land x_{u^4 v^2} = u^4 v^2 \land x_{u^2 vw} = u^2 vw \land x_{w^2} = w^2 \]
Translating into Linear Arithmetic

- **Idea:** linearize non-linear monomials with case analysis on some of the variables with finite domain

- Assume variables are in $\mathbb{Z}$

- $F \land x_{u^4v^2} = u^4v^2 \land x_{u^2vw} = u^2vw \land x_{w^2} = w^2$

  where $F$ is $x_{u^4v^2} + 2x_{u^2vw} + x_{w^2} \leq 4 \land 1 \leq u, v, w \leq 2$

- Since $1 \leq w \leq 2$, add $x_{u^2v} = u^2v$ and
  
  $w = 1 \rightarrow x_{u^2vw} = x_{u^2v}$

  $w = 2 \rightarrow x_{u^2vw} = 2x_{u^2v}$
Translating into Linear Arithmetic

Applying the same idea recursively, the following linear formula is obtained:

\[ x_{u^4v^2} + 2x_{u^2vw} + x_{w^2} \leq 4 \]
\[ \land 1 \leq u, v, w \leq 2 \]
\[ \land w = 1 \rightarrow x_{u^2vw} = x_{u^2v} \]
\[ \land w = 2 \rightarrow x_{u^2vw} = 2x_{u^2v} \]
\[ \land u = 1 \rightarrow x_{u^2v} = v \]
\[ \land u = 2 \rightarrow x_{u^2v} = 4v \]
\[ \land w = 1 \rightarrow x_{w^2} = 1 \]
\[ \land w = 2 \rightarrow x_{w^2} = 4 \]
\[ \land v = 1 \rightarrow x_{u^4v^2} = x_{u^4} \]
\[ \land v = 2 \rightarrow x_{u^4v^2} = 4x_{u^4} \]
\[ \land u = 1 \rightarrow x_{u^4} = 1 \]
\[ \land u = 2 \rightarrow x_{u^4} = 16 \]

A model can be computed:

\[ u = 1 \]
\[ v = 1 \]
\[ w = 1 \]
\[ x_{u^4v^2} = 1 \]
\[ x_{u^4} = 1 \]
\[ x_{u^2vw} = 1 \]
\[ x_{u^2v} = 1 \]
\[ x_{w^2} = 1 \]
Unsatisfiable Core Analysis

- If linearization achieves a linear formula then we have a **sound** and **complete** decision procedure.

- If we don’t have enough variables with finite domain... ...we can add bounds at cost of **losing completeness**. We cannot trust UNSAT answers!

- But we can analyze **why** the CNF is UNSAT: an **unsatisfiable core** (= unsatisfiable subset of clauses) can be obtained from the trace of the DPLL execution [Zhang & Malik’03]

- If core contains no extra bound: truly UNSAT
  If core contains extra bound: guide to enlarge domains
Unsatisfiable Core Analysis

• $u^4 v^2 + 2u^2 vw + w^2 \leq 3$ cannot be linearized

• Consider $u^4 v^2 + 2u^2 vw + w^2 \leq 3 \land 1 \leq u, v, w \leq 2$

• The linearization is unsatisfiable:

$$x_{u^4 v^2} + 2x_{u^2 vw} + x_{w^2} \leq 3$$
$$\land 1 \leq x_{u^4 v^2} \land x_{u^4 v^2} \leq 64$$
$$\land 1 \leq x_{u^2 vw} \land x_{u^2 vw} \leq 16$$
$$\land 1 \leq x_{w^2} \land x_{w^2} \leq 4$$
$$\land 1 \leq u \land u \leq 2$$
$$\land 1 \leq v \land v \leq 2$$
$$\land 1 \leq w \land w \leq 2$$

• Should decrease lower bounds for $u, v, w$
An Alternative Max-SMT Approach

- **Max-STM(T):** Given a set of weighted clauses, find a $T$-consistent assignment that minimizes cost ($=$ sum of weights) of falsified clauses.

- Assume we are given a non-linear formula and have computed a linearization (possibly with extra bounds).

  Then we transform the linear formula into a weighted one as follows:
  - Clauses $C$ of extra bounds are given finite weights $\omega_C$ (soft clauses).
  - Rest of clauses are given weight $\infty$ (hard clauses).

- So we have a **Max-SMT(LIA)** problem, instead of an **SMT(LIA)** one.

- If found model with null cost, we have a solution.
- Else falsified soft clauses show bounds to relax.
An Alternative Max-SMT Approach

• There exist simple Branch & Bound algorithms for Max-SMT
  [Nieuwenhuis & Oliveras, SAT’06], [Cimatti et al., TACAS’10]

• Advantages over the analysis of unsatisfiable cores
  • Max-SMT approach is easier to implement and maintain
  • Leads naturally to an extension to Max-SMT(NIA):
    Given a set of weighted clauses in NIA, linearize as usual but
    • Original clauses keep their weight
    • Clauses of case splits are given weight \( \infty \)
    • Clauses of extra bounds are given weights \( \omega > W \),
      where \( W \) is the sum of the weights of the original soft clauses

So models that violate original clauses are preferred over those violating
case splits (that ensure a true model for NA can be reconstructed)
An Alternative Max-SMT Approach

- Example revisited
- \(u^4v^2 + 2u^2vw + w^2 \leq 3\) cannot be linearized
- Consider \(u^4v^2 + 2u^2vw + w^2 \leq 3 \land 1 \leq u, v, w \leq 2\), with extra bounds having weight 1
- Linearization does not have 0-cost solution: optimal solutions have weight 1, e.g. falsifying \(1 \leq w\)
- Should decrease lower bound of \(w\)
Current set of targeted programs:

- **Imperative** programs: iterative and recursive (ignoring return values)
- **Integer variables** and **linear** expressions
  (other constructions considered unknowns)
Example

```c
int gcd ( int a, int b ) {
    int tmp;
    while ( a >= 0 && b > 0 ) {
        tmp = b;
        if (a == b) b = 0;
        else {
            int z = a;
            while ( z > b ) z -= b;
            b = z;
        }
        a = tmp;
    }
    return a;
}
```
Example

As a transition system:
As a transition system:

\[
\begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_1 : & \quad b \geq 1, \quad a \geq 0, \quad a = b, \quad a' = b, \quad b' = 0, \quad tmp' = b, \quad z' = z \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_5 : & \quad b \geq z, \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*}
\]
Proving Termination

- Idea: prove that no transition can be executed infinitely many times.

- In order to discard a transition $\tau_i$ we need either:
  - an unfeasibility argument, or
  - a ranking function $f$ over $\mathbb{Z}$ 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)
Auxiliary Assertions: Invariants

- We may need **invariant** assertions to build our termination argument

- We consider **inductive invariants**:
  - **Initiation condition**  
    (it holds the first time the location is reached)
  - **Consecution condition**  
    (it is preserved under every cycle back to the location)
Constraint-based Program Analysis

Introduced in [Colon, Sankaranarayanan & Sipma, CAV’03]

Keys:

- Fix a template for candidate invariants
- Impose initiation and consecution conditions obtaining \( \exists \forall \) problem
- Transform with Farkas' Lemma into \( \exists \) problem over non-linear arith.
- Constraints can be solved with SMT(NA) solver, e.g. Barcelogic.

Larraz, Oliveras, Rodríguez-Carbonell, Rubio, UPC, 2013

Non-linear Arithmetic Solving for Termination Analysis
Constraint-based Program Analysis

Introduced in [Colon, Sankaranarayanan & Sipma, CAV’03]

Keys:

- Fix a template for candidate invariants

\[ c_1 x_1 + \ldots + c_n x_n + d \leq 0 \]

where \( c_1, \ldots, c_n, d \) are unknowns
Constraint-based Program Analysis

Introduced in [Colon, Sankaranarayanan & Sipma, CAV’03]

Keys:

- Fix a template for candidate invariants

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

where \( c_1, \ldots, c_n, d \) are unknowns

- Impose initiation and consecution conditions obtaining \( \exists \forall \) problem
Constraint-based Program Analysis

Introduced in [Colon, Sankaranarayanan & Sipma, CAV’03]

Keys:

- Fix a template for candidate invariants
  \[ c_1x_1 + \ldots + c_nx_n + d \leq 0 \]
  where \( c_1, \ldots, c_n, d \) are unknowns

- Impose initiation and consecution conditions obtaining \( \exists \forall \) problem

- Transform with Farkas’ Lemma into \( \exists \) problem over non-linear arith.
Constraint-based Program Analysis

Introduced in [Colon, Sankaranarayanan & Sipma, CAV’03]

Keys:

- Fix a template for candidate invariants

\[ c_1 x_1 + \ldots + c_n x_n + d \leq 0 \]

where \( c_1, \ldots, c_n, d \) are unknowns

- Impose initiation and consecution conditions obtaining \( \exists \forall \) problem

- Transform with Farkas’ Lemma into \( \exists \) problem over non-linear arith.

- Constraints can be solved with SMT(NA) solver, e.g. Barcelogic.
Constraint-based Program Analysis

Following the ideas in [Bradley, Manna & Sipma, CAV’05]:
constraint-based invariant gen. (IG) + linear ranking function gen. (RG)

Assume a single location:

- **Templates**
  - For the invariant: \( I = c_1x_1 + \ldots + c_nx_n + d \leq 0 \)
  - For the ranking function: \( R = r_0 + r_1x_1 + \ldots + r_nx_n \)

- **Constraints**
  - Initiation condition on \( I \)
  - Consecution condition on \( I \)
  - \( R \) is non-increasing for all transitions
  - Some transition \( \tau_i \) can be discarded
    - \( I \iff \) unfeasibility of \( \tau_i \), or
    - \( I \iff \) strict decreasingness and boundedness of \( \tau_i \)
Although this looks like the way to work, it is not that good in practice:

- Sometimes several invariants needed to generate ranking function
  Then the problem is unsatisfiable (no solution for ranking function)
Although this looks like the way to work, it is not that good in practice:

- Sometimes several invariants needed to generate ranking function
  Then the problem is unsatisfiable (no solution for ranking function)

We need to express that even if our aim is to find a ranking function, if we find just an invariant we’ve made some progress
Although this looks like the way to work, it is not that good in practice:

- Sometimes several invariants needed to generate ranking function

  Then the problem is unsatisfiable (no solution for ranking function)

We need to express that even if our aim is to find a ranking function, if we find just an invariant we’ve made some progress

We can do it with Max-SMT
We can assign weights to the termination conditions:

1. $I \land \tau_i \implies R \geq 0$
2. $I \land \tau_i \implies R > R'$
3. $I \land \tau_j \implies R \geq R'$ for all $j$
Using Max-SMT to combine IG and RG

We can assign weights to the termination conditions:

1. \( I \land \tau_i \implies R \geq 0 \)
2. \( I \land \tau_i \implies R > R' \)
3. \( I \land \tau_j \implies R \geq R' \) for all \( j \)

1. \( (p_1, w_1) \) where \( p_1 \) represents the bound condition (1)
2. \( (p_2, w_2) \) where \( p_2 \) represents the strict-decreasing condition (2)
3. \( (p_3, w_3) \) where \( p_3 \) represents the non-increasing condition (3)
Using Max-SMT to combine IG and RG

We can assign weights to the termination conditions:

1. \( I \land \tau_i \implies R \geq 0 \)
2. \( I \land \tau_i \implies R > R' \)
3. \( I \land \tau_j \implies R \geq R' \) for all \( j \)

\((p_1, w_1)\) where \( p_1 \) represents the bound condition (1)
\((p_2, w_2)\) where \( p_2 \) represents the strict-decreasing condition (2)
\((p_3, w_3)\) where \( p_3 \) represents the non-increasing condition (3)

Once the problem is encoded in Max-SMT(NA):

- The Max-SMT solver looks for the best solution getting a ranking function if possible
- Otherwise, the weights can guide the search to get invariants and quasi-ranking functions that satisfy as many conditions as possible
Example

\[ \begin{align*}
\tau_0 & : \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \\
\tau_1 & : \quad b \geq 1, \quad a \geq 0, \quad a = b, \quad a' = b, \quad b' = 0, \quad \text{tmp}' = b, \quad z' = z \\
\tau_2 & : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_3 & : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_4 & : \quad b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \\
\tau_5 & : \quad b \geq z, \quad a' = \text{tmp}, \quad b' = z, \quad \text{tmp}' = \text{tmp}, \quad z' = z
\end{align*} \]
Example

Larraz, Oliveras, Rodríguez-Carbonell, Rubio, UPC, 2013
Non-linear Arithmetic Solving for Termination Analysis

\[\begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_1 : & \quad b \geq 1, \quad a \geq 0, \quad a = b, \quad a' = b, \quad b' = 0, \quad tmp' = b, \quad z' = z \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_5 : & \quad b \geq z, \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*}\]

Solver finds invariant \( b \geq 1 \) at \( l_8 \) and ranking function \( b \) for \( \tau_1 \)
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \]

\[ \tau_2 : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \]

\[ \tau_3 : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \]

\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \]

\[ \tau_5 : \quad b \geq z, \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z \]

Solver finds invariant \( b \geq 1 \) at \( l_8 \) and ranking function \( b \) for \( \tau_1 \)
Example

\[ \begin{align*}
\tau_0 & : & a' = ?, & b' = ?, & \text{tmp}' = ?, & z' = ? \\
\tau_2 & : & b \geq 1, & a \geq 0, & a < b, & a' = a, & b' = b, & \text{tmp}' = b, & z' = a \\
\tau_3 & : & b \geq 1, & a \geq 0, & a > b, & a' = a, & b' = b, & \text{tmp}' = b, & z' = a \\
\tau_4 & : & b < z, & a' = a, & b' = b, & \text{tmp}' = \text{tmp}, & z' = z - b \\
\tau_5 & : & b \geq z, & a' = \text{tmp}, & b' = z, & \text{tmp}' = \text{tmp}, & z' = z \\
\end{align*} \]

Nothing else can be done, but ...
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \]
\[ \tau_2 : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \]
\[ \tau_3 : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \]
\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \]
\[ \tau_5 : \quad b \geq z, \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z \]
We can split $\tau_5$ in three subcases and
We can split \( \tau_5 \) in three subcases and remove 5.2 by strict decreasingness.
Example

We can split $\tau_5$ in three subcases and remove 5.1 by unfeasibility
Example

\[ \begin{align*} 
\tau_0 & : \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \\
\tau_2 & : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_3 & : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_4 & : \quad b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \\
\tau_{5.3} & : \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = \text{tmp}, \quad b' = z, \quad \text{tmp}' = \text{tmp}, \quad z' = z 
\end{align*} \]
Example

\[ \begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \\
\tau_{5.3} : & \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = \text{tmp}, \quad b' = z, \quad \text{tmp}' = \text{tmp}, \quad z' = z
\end{align*} \]
Now, we cannot find a ranking function but get the invariant $a \geq z$ at $l_8$. 

\begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_{5.3} : & \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*}
Example

\[ l_3 \xrightarrow{\tau_0} l_8 \xrightarrow{\tau_2} l_3 \xrightarrow{\tau_3} l_8 \xrightarrow{\tau_4} l_3 \]

\begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_{5.3} : & \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*}

Now, we cannot find a ranking function but get the invariant \( a \geq z \) at \( l_8 \). Next, again, we only generate the invariant \( tmp = b \) at \( l_8 \).
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \]
\[ \tau_2 : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \]
\[ \tau_3 : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \]
\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \]
\[ \tau_{5.3} : \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = \text{tmp}, \quad b' = z, \quad \text{tmp}' = \text{tmp}, \quad z' = z \]

With the invariant \( a \geq 0 \) at \( l_8 \) we have that function \( a + b \) fulfills for \( \tau_{5.3} \):

- \( p_1 \) (bounded) and \( p_3 \) (non-increasing) but not \( p_2 \) (strict-decreasing)
With the invariant $a \geq 0$ at $l_8$ we have that function $a + b$ fulfills for $\tau_{5.3}$:

- $p_1$ (bounded) and $p_3$ (non-increasing) but not $p_2$ (strict-decreasing)

The Max-SMT solver generates $a + b$
Example

\[ \begin{align*}
\tau_0 : \quad & a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_2 : \quad & b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : \quad & b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : \quad & b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_{5.3} : \quad & b \geq z, \quad b \geq 0, \quad b = b', \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*} \]
Example

With ranking function $a + b$ we can split $\tau_{5.3}$ into

$$\tau_{5.4} : \tau_{5.3} \land a + b > a' + b'$$

$$\tau_{5.5} : \tau_{5.3} \land a + b = a' + b'$$
Example

With ranking function \( a + b \) we can split \( \tau_{5.3} \) into

\[
\tau_{5.4} : \tau_{5.3} \land a + b > a' + b' \\
\tau_{5.5} : \tau_{5.3} \land a + b = a' + b'
\]

Then \( \tau_{5.4} \) can be removed and \( \tau_{5.5} \) simplified: \( \tau_{5.5} : \tau_{5.3} \land a = a' \)
Example

\[ \begin{align*}
\tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
\tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
\tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
\tau_{5.3} : & \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z
\end{align*} \]
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \]

\[ \tau_2 : \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \]

\[ \tau_3 : \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad \text{tmp}' = b, \quad z' = a \]

\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \]

\[ \tau_{5.5} : \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = \text{tmp}, \quad b' = z, \quad \text{tmp}' = \text{tmp}, \quad z' = z \]

\[ a' = a \]

Using the information of the transitions we can infer that \( a = b \) after \( \tau_{5.5} \).
Example

\[
\begin{align*}
    \tau_0 : & \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \\
    \tau_2 : & \quad b \geq 1, \quad a \geq 0, \quad a < b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
    \tau_3 : & \quad b \geq 1, \quad a \geq 0, \quad a > b, \quad a' = a, \quad b' = b, \quad tmp' = b, \quad z' = a \\
    \tau_4 : & \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \\
    \tau_{5.5} : & \quad b \geq z, \quad b \geq 0, \quad b = b', \quad a' = tmp, \quad b' = z, \quad tmp' = tmp, \quad z' = z \quad a' = a
\end{align*}
\]

Using the information of the transitions we can infer that \( a = b \) after \( \tau_{5.5} \). Then the connections between \( \tau_{5.5} \) and \( \tau_2 \) or \( \tau_3 \) are unfeasible.
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \]

\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \]

Using the information of the transitions we can infer that \( a = b \) after \( \tau_{5.5} \). Then the connections between \( \tau_{5.5} \) and \( \tau_2 \) or \( \tau_3 \) are unfeasible.
Example

\[ e_1 = \tau_0, \quad e_2 = \tau_4 \]

\[ a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \]

\[ b < z, \quad a' = a, \quad b' = b, \quad \text{tmp}' = \text{tmp}, \quad z' = z - b \]
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad tmp' = ?, \quad z' = ? \]
\[ \tau_4 : \quad b < z, \quad a' = a, \quad b' = b, \quad tmp' = tmp, \quad z' = z - b \]

Solver generates ranking function \( z - b \) for \( \tau_4 \)
Example

\[ \tau_0 : \quad a' = ?, \quad b' = ?, \quad \text{tmp}' = ?, \quad z' = ? \]

We are DONE!
Advantages of the method:

- Using Max-SMT we can characterize different ways of progress depending on whether $p_1$, $p_2$ or $p_3$ are fulfilled.

- Using different weights we can encode which conditions are more important than others.
Implementation and experiments

- We have implemented these techniques
- The prototype reads C code
- Possible answers:
  - YES
  - NO (few cases)
  - Unknown
Implementation and experiments

- Experiments:
  - Benchmarks used in the Termination Competition for Java programs. 111 instances of iterative programs and 41 instances of recursive programs where termination follows from scalar information.

- Results are very promising:
  - Our first implementation is already competitive compared with tools for Java programs that have been developed since many years ago.

Results from the TermComp full-run December 2011:

<table>
<thead>
<tr>
<th></th>
<th>Iterative</th>
<th></th>
<th>Recursive</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>NO</td>
<td>MAYBE</td>
<td>YES</td>
</tr>
<tr>
<td>AProVE</td>
<td>77</td>
<td>0</td>
<td>36</td>
<td>32</td>
</tr>
<tr>
<td>Costa</td>
<td>64</td>
<td>0</td>
<td>49</td>
<td>28</td>
</tr>
<tr>
<td>Julia</td>
<td>72</td>
<td>21</td>
<td>20</td>
<td>35</td>
</tr>
<tr>
<td>Max-SMT</td>
<td>76</td>
<td>22</td>
<td>18</td>
<td>32</td>
</tr>
</tbody>
</table>
Implementation and experiments

- Experiments:
  - Programs made by students (can be ugly code). Obtained from an on-line learning environment (Jutge.org). 7924 instances coming from 12 different programming problems.

- Results are very promising:
  - These programs can be considered challenging. Most often they are not the most elegant solution but a working one with many more conditional statements than necessary.

<table>
<thead>
<tr>
<th></th>
<th>YES</th>
<th>NO</th>
<th>MAYBE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max-SMT</strong></td>
<td>6139</td>
<td>15</td>
<td>1770</td>
</tr>
</tbody>
</table>
Implementation and experiments

- Experiments:
  - Benchmarks taken from [Cook et al., CAV’13] coming from Windows device drivers, the Apache web server, the PostgreSQL server, integer approximations of numerical programs from a book on numerical recipes, integer approximations of benchmarks from LLBMC, ... 260 instances known to be terminating.

- Results are very promising:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperating-T2</td>
<td>YES</td>
</tr>
<tr>
<td>Terminator</td>
<td>245</td>
</tr>
<tr>
<td>T2</td>
<td>177</td>
</tr>
<tr>
<td>ARMC</td>
<td>189</td>
</tr>
<tr>
<td>AproVE</td>
<td>138</td>
</tr>
<tr>
<td>AproVE+Interproc</td>
<td>197</td>
</tr>
<tr>
<td>KITTeL</td>
<td>185</td>
</tr>
<tr>
<td>Max-SMT</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>197</td>
</tr>
</tbody>
</table>
Conclusions

- Approach to SMT(NA) that directly extends to Max-SMT(NA)
- Approach to termination analysis relying on Max-SMT
- Our prototype is already a competitive tool
Future work

There is a very long list...

- Improve invariant generation techniques. (e.g., by combining with abstract interpretation)
- Improve termination of recursive functions.
- Termination in presence of other data types (arrays, etc.)
- Improve the NA solver combining Barcelogic solver with other methods that are much better proving unsatisfiability (like [Jovanovic and De Moura, IJCAR’12])
Thank you!