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

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

    while (y>0) {

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

```c
void simpleT(int x, int y) {
    while (y > 0) {
        while (x > 0) {
            x = x - y;
            y = y + 1;
        }
        y = y - 1;
    }
}
```

Terminates.
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.
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 $\varphi$ over some theory $T$.

**Question:** Is there any interpretation 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 \( \varphi \) over some theory \( T \).

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

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

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

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

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

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

Example: $T =$ linear integer/real arithmetic.

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

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

There exist very efficient solvers: yices, z3, Barcelogic, ...

Can handle large formulas with a complex boolean structure.
SMT solving

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

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

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

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

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

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

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

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

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

\( \{x = 0, y = 1, z = 1\} \)
SMT solving

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

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

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

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

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

Non-linear arithmetic decidability:

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

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

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

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

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

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

Non-linear arithmetic decidability:

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

Incomplete solvers focused on either satisfiability or unsatisfiability.
SMT solving

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

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

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

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

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

Non-linear arithmetic decidability:

- **Integers:** undecidable
- **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 $\varphi = C_1 \land \ldots \land C_m$ 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.

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

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

```c
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:
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.
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.
Introduction

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* \( f : \text{States} \rightarrow \mathbb{Z} \), 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?
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?

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

\[
\begin{align*}
\tau_1 & : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*}
\]
Ranking functions and Invariants: Example

Transition system:

\[ \begin{align*}
\tau_1 & : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*} \]

\[ f(x, y, z) = z \] is a ranking function for \( \tau_2 \)
Ranking functions and Invariants: Example

Transition system:

Transition system:

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

$\rho_{T_1}: y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$

$\rho_{T_2}: y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1$

$\rho_{T_3}: y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z$
**Ranking functions and Invariants: Example**

**Transition system:**

\[ \begin{align*}
\tau_1 & : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*} \]

We can discard all executions that pass through \( \tau_2 \).
Ranking functions and Invariants: Example

Transition system:

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

We can discard all executions that pass through \( \tau_2 \).
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)
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:

\[
\begin{align*}
\tau_1 & : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*}
\]
Ranking functions and Invariants: Example

Transition system:

\[
\begin{align*}
\tau_1 & : l_1, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : l_2, \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : l_2, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*}
\]
Ranking functions and Invariants: Example

Transition system:

\[ \begin{align*}
\tau_1 & : 0 \leq 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_2 & : y \geq 1, \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\tau_3 & : y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*} \]

and ranking function \( f(x, y, z) = z \), fulfilling all properties for \( \tau_2 \)
Ranking functions and Invariants: Example

Transition system:

\[ l_1 \xrightarrow{\tau_1} l_2 \xrightarrow{\tau_2} l_1 \xrightarrow{\tau_3} l_2 \]

\[
\begin{align*}
\rho_{\tau_1} & : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\rho_{\tau_2} & : \quad y \geq 1, \quad y < z, \quad x' = x + 1, \quad y' = y, \quad z' = z - 1 \\
\rho_{\tau_3} & : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*}
\]

and ranking function \( f(x, y, z) = z \), fulfilling all properties for \( \tau_2 \)
Ranking functions and Invariants: Example

Transition system:

\[ \begin{align*}
\tau_1 &: 
\begin{align*}
y &\geq 1, \\
x' &= x - 1, \\
y' &= y, \\
z' &= z
\end{align*} \\
\tau_2 &: 
\begin{align*}
y &\geq 1, \\
y &< z, \\
x' &= x + 1, \\
y' &= y, \\
z' &= z - 1
\end{align*} \\
\tau_3 &: 
\begin{align*}
y &\geq 1, \\
y &\geq z, \\
x' &= x, \\
y' &= x + y, \\
z' &= z
\end{align*}
\end{align*} \]

and ranking function \( f(x, y, z) = z \), fulfilling all properties for \( \tau_2 \)
we can remove \( \tau_2 \)
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 \( f(x, y, z) = z \), fulfilling all properties for \( \tau_2 \)

we can remove \( \tau_2 \)
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 fulfilling 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:

\[ \rho_{\tau_1} : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \]

\[ \rho_{\tau'_3} : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z \]

There is no ranking function that fulfils all conditions.
Our approach: Example

Transition system:

\[ l_1 \xrightarrow{\tau_1} l_2 \xrightarrow{\tau_3} l_1 \]

\[ \rho_{\tau_1} : \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \]

\[ \rho_{\tau_3'} : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z \]

\[ f(x, y, z) = x \] is non-increasing and strict decreasing for \( \tau_1 \).

However, it is not bounded (soft).
Our approach: Example

Transition system:

\[ f(x, y, z) = x \] is non-increasing and strict decreasing for \( \tau_1 \).

However, it is not bounded (soft).
Our approach: Example

Transition system:

\[ \begin{align*}
\tau_{1.1} & : \quad x \geq 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_{1.2} & : \quad x < 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\tau_3 & : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*} \]

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

We can remove it!
Our approach: Example

Transition system:

\[ \begin{align*}
\rho_{\tau_{1.2}} & : \quad x < 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\rho_{\tau_3} & : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*} \]

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

We can remove it!
Our approach: Example

Transition system:

\[\begin{align*}
\rho_{\tau_{1.2}} & : \quad x < 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z \\
\rho_{\tau_3'} & : \quad y \geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z
\end{align*}\]

Finally, \( f(x, y, z) = y \) is used to discard \( \tau_3' \).
Our approach: Example

Transition system:

\[ \rho_{\tau_{1.2}} : \begin{align*} x &< 0 \quad y \geq 1, \\ x' & = x - 1, \quad y' = y, \quad z' = z \end{align*} \]

\[ \rho_{\tau'_{3}} : \begin{align*} y &\geq 1, \quad y \geq z, \quad x' = x, \quad y' = x + y, \quad z' = z \end{align*} \]

Finally, \( f(x, y, z) = y \) is used to discard \( \tau'_{3} \).

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 \( \tau'_3 \).

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 \textit{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:

<table>
<thead>
<tr>
<th>#ins.</th>
<th>CppInv</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set1</td>
<td>449</td>
<td>238</td>
</tr>
<tr>
<td>Set2</td>
<td>472</td>
<td>276</td>
</tr>
</tbody>
</table>

**Table:** Results with benchmarks from T2

<table>
<thead>
<tr>
<th>#ins.</th>
<th>CppInv</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P11655</td>
<td>367</td>
<td>324</td>
</tr>
<tr>
<td>P12603</td>
<td>149</td>
<td>143</td>
</tr>
<tr>
<td>P12828</td>
<td>783</td>
<td>707</td>
</tr>
<tr>
<td>P16415</td>
<td>98</td>
<td>81</td>
</tr>
<tr>
<td>P24674</td>
<td>177</td>
<td>171</td>
</tr>
<tr>
<td>P33412</td>
<td>603</td>
<td>478</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#ins.</th>
<th>CppInv</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P40685</td>
<td>362</td>
<td>324</td>
</tr>
<tr>
<td>P45965</td>
<td>854</td>
<td>780</td>
</tr>
<tr>
<td>P70756</td>
<td>280</td>
<td>243</td>
</tr>
<tr>
<td>P81966</td>
<td>3642</td>
<td>2663</td>
</tr>
<tr>
<td>P82660</td>
<td>196</td>
<td>174</td>
</tr>
<tr>
<td>P84219</td>
<td>413</td>
<td>325</td>
</tr>
</tbody>
</table>

**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!
<table>
<thead>
<tr>
<th>Introduction</th>
<th>SMT/Max-SMT solving</th>
<th>Invariant generation</th>
<th>Termination analysis</th>
<th>Further work</th>
</tr>
</thead>
</table>

Proving Termination of Imperative Programs Using Max-SMT
Example of students’ code

```c
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;
}
```

```c
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:

\[
\begin{pmatrix}
  a_{11}x_1 + \cdots + a_{1n}x_n + b_1 \leq 0 \\
  \vdots & \vdots & \vdots & \leq 0 \\
  a_{m1}x_1 + \cdots + a_{mn}x_n + b_m \leq 0
\end{pmatrix} \Rightarrow \varphi : c_1x_1 + \cdots + c_nx_n + d \leq 0
\]

\[
\Leftrightarrow \exists \lambda_0, \lambda_1, \ldots, \lambda_m \geq 0,
\]

\[
c_1 = \sum_{i=1}^{m} \lambda_i a_{i1}, \ldots, c_n = \sum_{i=1}^{m} \lambda_i a_{in},
\]

\[
d = \left( \sum_{i=1}^{m} \lambda_i b_i \right) - \lambda_0
\]