Automatic Inference of Ranking Functions by Abstract Interpretation

Caterina Urban

Project-Team ANTIQUE
INRIA Paris-Rocquencourt

Département d’Informatique
École Normale Supérieure

18th March 2014
INRIA Junior Seminar
INRIA Paris-Rocquencourt, France
Project-Team ANTIQUE

ANalyse Statique par Interprétation Abstraite
Proving Program Termination? Why?

30GB Zunes all over the world fail en masse | TechCrunch

It seems that a random bug is affecting a bunch, if not every, 30GB Zunes. Real early this morning, a bunch of Zune 30s just stopped working. No official word from Redmond on this one yet but we might have a gadget Y2K going on here. Fan boards and support forums all have the same mantra saying that at 2:00 AM this morning, the Zune 30s reset on their own and doesn't fully reboot. We're sure Microsoft will get flooded with angry Zune owners as soon as the phone lines open up for the last time in 2008. More as we get it.

Update 2: The solution is ... kind of weak: let your Zune run out of battery and it'll be fixed when you wake up tomorrow and charge it.
Proving Program Termination? Why?

Zune bug explained in detail

Posted Dec 31, 2008 by Devin Coldewey

Earlier today, the sound of thousands of Zune owners crying out in terror made ripples across the blogosphere. The response from Microsoft is to wait until tomorrow and all will be well. You’re probably wondering, what kind of bug fixes itself?

Well, I’ve got the code here and it’s very simple, really; if you’ve taken an introductory programming class, you’ll see the error right away.

```c
while (days > 365) {
    if (IsLeapYear(year))
        if (days > 366)
            days -= 366;
        year += 1;
    else
        days -= 365;
        year += 1;
}
```

You can see the details here, but the important bit is that today, the day count is 366. As you can
Outline

- **ranking functions**
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below

- **idea**: computation of ranking functions by abstract interpretation

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination

- instances based on **affine ranking functions**

---

1. Floyd - *Assigning Meanings to Programs* (1967)
Outline

- **ranking functions**¹
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below

- **idea**: computation of ranking functions by abstract interpretation²

- family of abstract domains for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination

- instances based on affine ranking functions³

---
¹ Floyd - *Assigning Meanings to Programs* (1967)
² Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
³ Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Outline

- **ranking functions**
  - functions that strictly decrease at each program step...
  - ... and that are bounded from below

- **idea**: computation of ranking functions by abstract interpretation

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination

- instances based on **affine ranking functions**

---

1. **Floyd** - *Assigning Meanings to Programs* (1967)
Outline

- **ranking functions**\(^1\)
  - functions that strictly **decrease** at each program step...
  - ... and that are **bounded** from below

- **idea**: computation of ranking functions by abstract interpretation\(^2\)

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - **backward** invariance analysis
  - **sufficient conditions** for termination

- **instances based on** **affine ranking functions**\(^3\)

---

\(^1\) Floyd - *Assigning Meanings to Programs* (1967)
\(^2\) Cousot& Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
\(^3\) Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Abstract Interpretation

\[ \langle C, \sqsubseteq C \rangle \]

---

\(4\) Cousot&Cousot - Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. (POPL 1977)
Abstract Interpretation\textsuperscript{4}

\[ \langle C, \sqsubseteq_C \rangle \]

\[ \gamma \]

\[ S \]

\[ [P] \]

\[ \alpha \]

\[ S^\# \]

\[ \langle A, \sqsubseteq_A \rangle \]

\[ [P^\#] \]

\textsuperscript{4}Cousot&Cousot - Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. (POPL 1977)
Abstract Interpretation$^4$

\[ \langle C, \sqsubseteq_C \rangle \]

\[ \langle A, \sqsubseteq_A \rangle \]

\[ S \]

\[ S^\# \]

\[ \alpha([P]) \]

\[ \alpha([P]) \]

---

$^4$ Cousot & Cousot - Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. (POPL 1977)
Abstract Interpretation

\[ \langle C, \sqsubseteq_C \rangle \]

\[ \langle A, \sqsubseteq_A \rangle \]

\[ \gamma(\lceil P\rceil) \]

\[ \gamma(\lceil P\rceil) \]

\[ \alpha(\lceil P\rceil) \]

\[ \alpha(\lceil P\rceil) \]

\[ \gamma(\lceil P\rceil) \]

\[ \gamma(\lceil P\rceil) \]

Cousot & Cousot - Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. (POPL 1977)
Concrete Semantics
program $\mapsto$ trace semantics

finite traces $\Sigma^+$

infinite traces $\Sigma^\infty$

$\Sigma$ states

$\tau$ transition relation

$\beta$ final states
program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

**Example**

Idea $=$ define a ranking function that **counts the number of program steps** from the end of the program

**Theorem (Soundness and Completeness)**

*the termination semantics is sound and complete to prove the termination of programs*

---

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
program \mapsto \text{trace semantics} \mapsto \text{termination semantics}

Example

\begin{center}
\begin{tikzpicture}[scale=0.8]
\node (root) at (0,0) {0};
\node (left) at (-1,-1) {0};
\node (right) at (1,-1) {0};
\draw (root) -- (left);
\draw (root) -- (right);
\end{tikzpicture}
\end{center}

Theorem (Soundness and Completeness)

*the termination semantics is sound and complete* to prove the termination of programs

---

Cousot & Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
program \mapsto \text{trace semantics} \mapsto \text{termination semantics}

\textbf{Example}

Theorem (Soundness and Completeness)

\textit{the termination semantics is} \textbf{sound and complete} \textit{to prove the termination of programs}

\textit{Cousot&Cousot - An Abstract Interpretation Framework for Termination (POPL 2012)}
program $\mapsto$ trace semantics $\mapsto$ termination semantics

Example

Theorem (Soundness and Completeness)

*the* termination semantics is **sound and complete** to prove the termination of programs

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
**Theorem (Soundness and Completeness)**

The termination semantics is **sound and complete** to prove the termination of programs.

---

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
The termination semantics is **sound and complete** to prove the termination of programs.

*Cousot&Cousot - An Abstract Interpretation Framework for Termination (POPL 2012)*
program $\mapsto$ trace semantics $\mapsto$ termination semantics

Example

Theorem (Soundness and Completeness)

*the* termination semantics *is* sound and complete
*to prove the termination of programs*

Cousot & Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
Example

\[
\text{int : } x \\
x := ? \\
\text{while } (x \geq 0) \text{ do} \\
x := x - 1 \\
\text{od}
\]
Example

```plaintext
int : x
x := ?
while (x ≥ 0) do
  x := x - 1
od
```

the termination semantic is not computable!
Piecewise-Defined Ranking Functions
Introduction

Concrete Semantics

Piecewise-Defined Ranking Functions

Conclusion and Future Work

Affine Ranking Functions

Implementation

Termination Semantics

Abstract Termination Semantics

- States Abstract Domain $S$
- Functions Abstract Domain $F$
- Piecewise-Defined Ranking Functions Abstract Domain $V(S, F)$
Termination Semantics $\langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle$

Abstract Termination Semantics $\langle \mathcal{V}, \sqsubseteq_{\mathcal{V}} \rangle$

- States Abstract Domain
- Functions Abstract Domain
- Piecewise-Defined Ranking Functions Abstract Domain $\gamma$

\[ \gamma \]

$\langle V, \sqsubseteq \rangle$

\[ V(S, F) \]

\[ \gamma \]

$\gamma$

\[ 5, 9 \]

\[ x \]

\[ S \]

\[ F \]
Termination Semantics  \( \langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle \)

Abstract Termination Semantics  \( \langle \mathcal{V}, \sqsubseteq_{\mathcal{V}} \rangle \)

- States Abstract Domain
- Functions Abstract Domain
- Piecewise-Defined Ranking Functions Abstract Domain  \( V(S, F) \)

\[ x \]

5
9
Termination Semantics
\[ \langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle \]

Abstract Termination Semantics
\[ \langle V, \sqsubseteq_V \rangle \]

- States Abstract Domain
- Functions Abstract Domain
- Piecewise-Defined Ranking Functions Abstract Domain

\[ V(S, F) \]
Why Piecewise-Defined Ranking Functions?

Example

```c
int : x
while 1(x ≠ 0) {
  if 2(x < 0) {
    3x := x + 1;
  } else {
    4x := x - 1;
  }
}
```

```
f(x) ≜ \begin{cases} 
-3x + 1 & x < 0 \\
1 & x = 0 \\
3x + 1 & x > 0 
\end{cases}
```

Example

```c
int : x
while 1(x ≥ 0) {
  2x := -2x + 10;
}
```

```
f(x) ≜ \begin{cases} 
1 & x < 0 \\
5 & 0 ≤ x ≤ 2 \\
9 & x = 3 \\
7 & 4 ≤ x ≤ 5 \\
3 & 5 < x 
\end{cases}
```
Affine Ranking Functions
Affine Ranking Functions Domain

States Abstract Domain
- $S \triangleq$ Intervals Abstract Domain

Functions Abstract Domain
- $F \triangleq \{ \bot_F \} \cup \{ f \mid f \in \mathbb{Z}^n \to \mathbb{N} \} \cup \{ \top_F \}$
  where $f \equiv f(x_1, \ldots, x_n) = m_1x_1 + \cdots + m_nx_n + q$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Affine Ranking Functions Domain

States Abstract Domain
- \( S \triangleq \text{Intervals Abstract Domain} \)

Functions Abstract Domain
- \( F \triangleq \{ \bot_F \} \cup \{ f \mid f \in \mathbb{Z}^n \rightarrow \mathbb{N} \} \cup \{ \top_F \} \)
  where \( f \equiv f(x_1, \ldots, x_n) = m_1x_1 + \cdots + m_nx_n + q \)

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Affine Ranking Functions Domain

- States Abstract Domain
  - $S \triangleq$ Intervals Abstract Domain
- Functions Abstract Domain
  - $\mathcal{F} \triangleq \{\bot_F\} \cup \{f \mid f \in \mathbb{Z}^n \rightarrow \mathbb{N}\} \cup \{\top_F\}$
  - where $f \equiv f(x_1, \ldots, x_n) = m_1x_1 + \cdots + m_nx_n + q$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Example

\[
\text{int : } x \\
\text{while }^1(x > 0) \{ \\
\quad 2x := x - 1 \\
\}^3
\]

we map each point to a function of \(x\) giving an upper bound on the steps before termination

\[
\begin{align*}
1 & \quad x \leq 0 \\
\downarrow & \\
2 & \quad x > 0 \\
\downarrow & \\
3 &
\end{align*}
\]

\[x := x - 1\]
Example

```plaintext
int : x
while \( x > 0 \) { 
  \( x \) := \( x - 1 \)
}
```

we start at the end with 0 steps before termination
Example

```plaintext
int : x
while \( x > 0 \) {
    \( x := x - 1 \)
}
```

we take into account \( x \leq 0 \) and we have 1 step to termination
Example

\[
\begin{align*}
\text{int} :& \ x \\
\text{while } & 1(x > 0) \{ \\
& 2x := x - 1 \}
\end{align*}
\]

we consider the assignment \( x := x - 1 \) and we are at 2 steps to termination
we consider $x > 0$ and we do the join $\sqcup \mathcal{V}$.
Example

int : x
while 1(x > 0) {
  2x := x − 1
} 3
Example

```latex
\begin{align*}
\text{int : } & x \\
\text{while } & 1(x > 0) \{ \\
& 2x := x - 1 \\
\} \tag{3}
\end{align*}
```

We do the widening $\nabla \nabla$.
Example

int : x
while $1(x > 0)$ {
  $2x := x - 1$
}

1. $x \leq 0$
2. $x > 0$
3. $x := x - 1$
Example

```plaintext
int : x
while ^1(x > 0) {
  x := x − 1
} ^3
```

the analysis gives true as sufficient precondition for termination
- segmentation unification

**Example**

- join: $\sqcup_N$
- widening: $\nabla_N$
- backward assignments: $\text{ASSIGN}_N$

\[
\begin{align*}
4 & \quad + \quad 2 & \quad = & \quad 2 & \quad 4
\end{align*}
\]
- segmentation unification
- join: $\sqcup_V$

**Example**

- widening: $\nabla_V$
- backward assignments: $\text{ASSIGN}_V$
• segmentation unification
• join: $\sqcup_V$
• widening: $\triangledown_V$

Example

• backward assignments: $\text{ASSIGN}_V$
• segmentation unification
• join: $\sqcup_V$
• widening: $\triangledown_V$

**Example**

![Diagram showing segmentation unification, join, and widening](attachment:image.png)

• backward assignments: $\text{ASSIGN}_V$
- segmentation unification
- join: $\sqcup_V$
- widening: $\nabla_V$

**Example**

- backward assignments: $\text{ASSIGN}_V$
- segmentation unification
- join: $\sqcup_V$
- widening: $\nabla_V$
- backward assignments: $\text{ASSIGN}_V$

**Example**

```
Example

$$x := x + [0, 5]$$
```
- segmentation unification
- join: $\sqcup_V$
- widening: $\nabla_V$
- backward assignments: $\text{ASSIGN}_V$

Example

$$x := x + [0, 5]$$
Theorem (Soundness)

the abstract termination semantics is **sound** to prove the termination of programs
Example

\[
\text{int : } x \\
\text{while } 1(x \leq 10) \text{ do} \\
\quad \text{if } 2(x > 6) \text{ then} \\
\quad \quad 3x := x + 2 \\
\quad \text{fi} \\
\text{od}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4
\end{array}
\]

\[
\begin{array}{c}
x > 10 \\
x \leq 10 \\
x \leq 6 \\
x > 6
\end{array}
\]

\[
\begin{array}{c}
x := x + 2
\end{array}
\]
we map each point to a function of $x$ giving an upper bound on the steps before termination

Example

```plaintext
int : x
while 1(x ≤ 10) do
  if 2(x > 6) then
    3x := x + 2
  fi
od
```

```
x > 10

1

x ≤ 6
2

x ≤ 10

3

x > 6
x := x + 2
4
```
Example

```plaintext
int : x
while \( x \leq 10 \) do
  if \( x > 6 \) then
    \( x := x + 2 \)
  fi
od
```

we start at the end with 0 steps before termination

\[ x > 10 \]

\[ x \leq 6 \]

\[ x \leq 10 \]

\[ x > 6 \]

\[ x := x + 2 \]
we take into account $x > 10$ and we have now 1 step to termination

Example

```plaintext
int : x
while $1(x \leq 10)$ do
  if $2(x > 6)$ then
    $3x \leftarrow x + 2$
  fi
od$^4$
```
Example

int : x
while \( x \leq 10 \) do
  if \( x > 6 \) then
    \( x := x + 2 \)
  fi
od

we consider the assignment \( x := x + 2 \)
or the test \( x \leq 6 \) and we are now at 2 steps to termination
Example

int : x
while 1(x ≤ 10) do
  if 2(x > 6) then
    3x := x + 2
  fi
od

we consider x > 6 and we do the join
we consider $x \leq 10$ and we do the join

**Example**

```
int : x
while $1(x \leq 10)$ do
  if $2(x > 6)$ then
    $3x := x + 2$
  fi
od
```

Graph showing the flow of the program with the initial state at $x = 0$, progressing through the while loop, and the join operations.
Example

\begin{align*}
\text{int} : & \ x \\
\text{while } & 1(x \leq 10) \text{ do } \\
& \text{if } 2(x > 6) \text{ then } \\
& \quad 3x := x + 2 \\
& \text{fi} \\
\text{od}
\end{align*}
we do the widening

Example

\begin{align*}
\text{int} & : x \\
\text{while } & 1 (x \leq 10) \text{ do} \\
& \text{if } 2 (x > 6) \text{ then} \\
& \quad 3 \ x := x + 2 \\
& \quad \text{fi} \\
& \text{od}
\end{align*}
Example

int : x
while \(1(x \leq 10)\) do
  if \(2(x > 6)\) then
    \(3x := x + 2\)
  fi
od
the analysis provides $x > 6$ as sufficient precondition for termination

Example

int : $x$
while $1(x \leq 10)$ do
  if $2(x > 6)$ then
    $3x := x + 2$
  fi
od

1. $x \leq 6$
2. $x \leq 10$
3. $x > 6$
4. $x := x + 2$

Affine Ranking Functions
Implementation

Concrete Semantics
Piecewise-Defined Ranking Functions
Conclusion and Future Work

Introduction
http://www.di.ens.fr/~urban/FuncTion.html

- written in OCaml
Experiments

**Benchmarks:** 87 terminating C programs collected from the literature

**Tools:**
- AProVE
- T2
- Ultimate Büchi Automizer

**Results:**

<table>
<thead>
<tr>
<th></th>
<th>Tot</th>
<th>FuncTion</th>
<th>AProVE</th>
<th>T2</th>
<th>Ultimate</th>
<th>Time</th>
<th>Timeouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuncTion</td>
<td>51</td>
<td>—</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>6s</td>
<td>5</td>
</tr>
<tr>
<td>AProVE</td>
<td>60</td>
<td>17</td>
<td>—</td>
<td>7</td>
<td>2</td>
<td>35s</td>
<td>19</td>
</tr>
<tr>
<td>T2</td>
<td>73</td>
<td>30</td>
<td>20</td>
<td>—</td>
<td>3</td>
<td>2s</td>
<td>0</td>
</tr>
<tr>
<td>Ultimate</td>
<td>79</td>
<td>31</td>
<td>21</td>
<td>9</td>
<td>—</td>
<td>9s</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusions

- family of abstract domains for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on natural-valued functions
  - affine ranking functions
- instances based on ordinal-valued functions
  - ordinals remove the burden of finding lexicographic orders
  - analysis not limited to programs with linear computational complexity

Future Work

- more abstract domains (e.g., non-linear ranking functions)
- other liveness properties
- complexity analysis
Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **natural-valued functions**
  - affine ranking functions
- instances based on **ordinal-valued functions**
  - ordinals remove the burden of finding lexicographic orders
  - analysis not limited to programs with linear computational complexity

Future Work

- more **abstract domains** (e.g., non-linear ranking functions)
- other liveness properties
- complexity analysis
Thank You!

Questions?

“...the purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.”
(Edsger Dijkstra)