The Abstract Domain of Piecewise-Defined Ranking Functions

Caterina Urban

Département d’Informatique
École Normale Supérieure

AVDCPS 2013
Changsha, China
Introduction

- 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 ranking functions over natural numbers\(^3\)
- instances based on ranking functions over ordinal numbers\(^4\)

---

\(^1\) 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)

\(^4\) Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (to appear)
Introduction

- **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 ranking functions **over natural numbers**\(^3\)
- instances based on ranking functions **over ordinal numbers**\(^4\)

---

\(^1\)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)
\(^4\)Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (to appear)
Introduction

- **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 ranking functions **over natural numbers**\(^3\)
- instances based on ranking functions **over ordinal numbers**\(^4\)

---

\(^1\)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)

\(^4\)Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (to appear)
Introduction

- **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 ranking functions over **natural numbers**\(^3\)
- instances based on ranking functions over **ordinal numbers**\(^4\)

---

\(^1\)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)
\(^4\)Urban & Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (to appear)
Introduction

- **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 ranking functions **over natural numbers**
- instances based on ranking functions **over ordinal numbers**

---

1. Floyd - *Assigning Meanings to Programs* (1967)
**Program Partial Correctness**

**Example**

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

---

Floyd - *Assigning Meanings to Programs* (1967)
Program Partial Correctness

affix **assertions** to each program control point...

Example

```
int : x
while 1(x ≤ 10) do
  if 2(x > 6) then
    3x := x + 2
  fi
od 4
```

...and prove they are consequences of the assertions of their predecessors

---

Floyd - *Assigning Meanings to Programs* (1967)
Program Partial Correctness

Example

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

Precondition

\[\text{start}\]

Postcondition

\[\text{end}\]

Floyd - Assigning Meanings to Programs (1967)
Program Partial Correctness

Example

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

- **precondition**: `x ≤ 10`
- **invariant**: `x ≤ 6`
- **postcondition**: `x > 10`

The program gives the correct result if and when it terminates.

Floyd - *Assigning Meanings to Programs* (1967)
Program Total Correctness

Example

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

Total Correctness = Partial Correctness + Termination

Floyd - Assigning Meanings to Programs (1967)
**Program Total Correctness**

associate a function over a **well-ordered set** to each program control point...

**Example**

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

**Total Correctness = Partial Correctness + Termination**

---

Floyd - *Assigning Meanings to Programs* (1967)
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

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

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 $\mapsto$ 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)
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)
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)
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

\begin{verbatim}
int : x
x := ?
while (x ≥ 0) do
  x := x - 1
od
\end{verbatim}

\[
\begin{array}{c}
\omega \\
0 \cdots 0 \ 1 \ 2 \ n \ \cdots \\
0 \ 1 \ \cdots \ n-1 \\
0 \\
1 \\
0
\end{array}
\]
Example

```plaintext
int : x
x := ?

while (x ≥ 0) do
  x := x - 1
od
```

The termination semantics is not computable.
Example

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

we need ordinals

the termination semantics is not computable
Piecewise-Defined Ranking Functions
Termination Semantics \[\gamma\] Abstract Termination Semantics

- States Abstract Domain \(S\)
  - Intervals Abstract Domain\(^5\) \(S\)
- Functions Abstract Domain \(F\)
  - Affine Ranking Functions \(F\)
- Piecewise-Defined Ranking Functions Abstract Domain \(V(S, F)\)

\(^5\) Cousot&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)
Concrete Semantics

Introduction

Piecewise-Defined Ranking Functions

Conclusion and Future Work

Natural-Valued Ranking Functions

Termination Semantics

\[ \langle \Sigma \rightarrow \emptyset, \subseteq \rangle \]

Abstract Termination Semantics

\[ \langle V\#, \subseteq\# \rangle \]

- States Abstract Domain
- Intervals Abstract Domain\(^5\)
- Functions Abstract Domain
- Affine Ranking Functions
- Piecewise-Defined Ranking Functions Abstract Domain

\[ V(S, F) \]

---

\(^5\) Cousot\&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)
Termination Semantics
\[ \langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle \]

Abstract Termination Semantics
\[ \langle \mathcal{V}^\#, \sqsubseteq^\# \rangle \]

- States Abstract Domain
- Intervals Abstract Domain\(^5\)
- Functions Abstract Domain
- Affine Ranking Functions
- Piecewise-Defined Ranking Functions Abstract Domain

\[ V(S, F) \]

---

\(^5\) Cousot & Cousot - *Static Determination of Dynamic Properties of Programs* (1976)
Termination Semantics
\[ \langle \Sigma \rightarrow \emptyset, \subseteq \rangle \]

Abstract Termination Semantics
\[ \langle V\# , \subseteq \# \rangle \]

- States Abstract Domain
- Intervals Abstract Domain \(^5\)
- Functions Abstract Domain
  - Affine Ranking Functions
- Piecewise-Defined Ranking Functions Abstract Domain

\(^5\) Cousot & Cousot - *Static Determination of Dynamic Properties of Programs* (1976)
Natural-Valued Ranking Functions
Natural-Valued Ranking Functions Domain

\[ (\Sigma \rightarrow \emptyset, \sqsubseteq) \]

\[ (\mathcal{V}\# \equiv \mathcal{P}(\mathcal{S}\# \times \mathcal{F}\#), \sqsubseteq\#) \]

\[ \nu\# \equiv \begin{cases} 
  s_1\# & \mapsto f_1\# \\
  s_2\# & \mapsto f_2\# \\
  \vdots \\
  s_k\# & \mapsto f_k\# 
\end{cases} \]

\[ \mathcal{F}\# \equiv \{ \bot_F \} \cup \{ f\# | f\# \in \mathbb{Z}^n \rightarrow \mathbb{N} \} \cup \{ T_F \} \]

where \( f\# \equiv y = f(x_1, \ldots, x_n) = m_1x_1 + \cdots + m_nx_n + q \)

---

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
• segmentation unification

---

**Example**

\[
\begin{array}{c}
y \uparrow \\
3 \quad 1 \\
x \downarrow \\
4 \\
\end{array}
\quad +
\quad
\begin{array}{c}
y \uparrow \\
1 \\
x \downarrow \\
2 \\
\end{array}
\quad =
\quad
\begin{array}{c}
y \uparrow \\
3 \\
x \downarrow \\
2 \\
\end{array}
\]

• join: \( \sqcap_V \)
• widening: \( \nabla_V \)
• backward assignments: \( \text{ASSIGN}_V \)
- segmentation unification
- join\(^6\): \(\sqcup_V\)

**Example**

\[ f_1(x_1, x_2) = -\frac{1}{2}x_2 + 2 \]
\[ f_2(x_1, x_2) = -\frac{1}{2}x_1 + 2 \]
\[ f(x_1, x_2) = -\frac{1}{2}x_1 - \frac{1}{2}x_2 + 4 \]

- widening: \(\nabla_V\)
- backward assignments: \(\text{ASSIGN}_V\)

\(^6\)Cousot&Halbwachs - *Automatic Discovery of Linear Restraints Among Variables of a Program* (POPL 1978)
- segmentation unification
- join: $\sqcup_V$
- widening: $\triangledown_V$

**Example**

\[
\begin{align*}
\text{before:} & \quad 6, 11 \\
\text{after:} & \quad 3, 6, 11
\end{align*}
\]

- backward assignments: ASSIGN$_V$
- segmentation unification
- join: $\sqcap_V$
- widening: $\triangledown_V$

**Example**

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

**Example**

![Diagram showing examples of segmentation unification, join, and widening]

- backward assignments: $\text{ASSIGN}_V$
- segmentation unification
- join: \( \sqcup_\mathcal{V} \)
- widening: \( \nabla_\mathcal{V} \)

**Example**

![Diagram illustrating segmentation, join, and widening with elements 6 and 11.]
- segmentation unification
- join: $\sqcup_V$
- widening: $\triangledown_V$
- backward assignments: $\text{ASSIGN}_V$

**Example**

$$\langle x \mapsto (-\infty, 5], \bot_F \rangle$$

$$\langle x \mapsto [6, +\infty), y = 4 \rangle$$

$$\xRightarrow{x := x + [0, 4]}$$

$$\langle x \mapsto (-\infty, 5], \bot_F \rangle$$

$$\langle x \mapsto [2, +\infty), y = 4 + 1 \rangle$$
• segmentation unification
• join: ⊔_V
• widening: ▽_V
• backward assignments: ASSIGN_V

Example

\[ x := x + [0, 4] \]

\[ \langle x \mapsto (-\infty, 5], \bot_F \rangle \]
\[ \langle x \mapsto [6, +\infty), y = 4 \rangle \]

\[ \Rightarrow \]

\[ \langle x \mapsto (-\infty, 1], \bot_F \rangle \]
\[ \langle x \mapsto [2, 5], \bot_F \rangle \]
\[ \langle x \mapsto [6, +\infty), y = 5 \rangle \]
Example

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

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 > 0) do
  2x := x - 1
od
```

we take into account $x \leq 0$ and we have 1 step to termination

we start at the end with 0 steps before termination
we consider $x > 0$ and we do the join

we consider the assignment $x := x - 1$ and we are at 2 steps to termination
Example

int : x
while \( x > 0 \) do
  \( x := x - 1 \)
end

\( x \leq 0 \)
\( x > 0 \)
Example

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

we do the widening

\[\text{Natural-Valued Ranking Functions}\]
\[\text{Ordinal-Valued Ranking Functions}\]
\[\text{Abstract Termination Semantics}\]
\[\text{Implementation}\]
Example

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

```
int : x
while 1 (x > 0) do
  2 x := x - 1
od
```

the analysis gives true as sufficient precondition for termination
Ordinal-Valued Ranking Functions
Ordinal-Valued Ranking Functions Domain

Urban&Miné - An Abstract Domain to Infer Ordinal-Valued Ranking Functions (to appear)
Ordinal-Valued Ranking Functions Domain

Urban&Miné - An Abstract Domain to Infer Ordinal-Valued Ranking Functions (to appear)
Ordinal-Valued Ranking Functions Domain

\[ \langle \Sigma \rightarrow \mathbb{O}, \sqsubseteq \rangle \]

\[ \langle \mathcal{V}^\# \triangleq \mathcal{P}(S^\# \times \mathcal{P}^\#), \sqsubseteq^\# \rangle \]

\[ \mathcal{P}^\# \triangleq \{ \bot_P \} \cup \{ p^\# \mid p^\# \in \mathbb{Z}^n \rightarrow \mathbb{O} \} \cup \{ \top_P \} \]

\[ \gamma \]

Urban&Miné - An Abstract Domain to Infer Ordinal-Valued Ranking Functions (to appear)
Ordinal-Valued Ranking Functions Domain

\[ \langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle \quad \langle \mathcal{V}^\# \triangleq \mathcal{P}(S^\# \times \mathcal{P}^\#), \sqsubseteq^\# \rangle \]

- \( \mathcal{P}^\# \triangleq \{ \bot_P \} \cup \{ p^\# \mid p^\# \in \mathbb{Z}^n \rightarrow \emptyset \} \cup \{ \top_P \} \)

- \( \mathcal{V}^\# \triangleq \{ s^\#_{1} \mapsto p^\#_{1}, s^\#_{2} \mapsto p^\#_{2}, \ldots, s^\#_{k} \mapsto p^\#_{k} \} \)

\[ \mathcal{P}^\# = \{ \bot_P \} \cup \{ \sum_{i} \omega_i \cdot f_i^\#, f_i^\# \in \mathcal{F}^\# \} \cup \{ \top_P \} \]

Urban&Míné - An Abstract Domain to Infer Ordinal-Valued Ranking Functions (to appear)
Ordinal-Valued Ranking Functions Domain

\[
\langle \Sigma \rightarrow \emptyset, \sqsubseteq \rangle
\]

\[
\langle V^\# \triangleq \mathcal{P}(S^\# \times P^\#), \sqsubseteq^\# \rangle
\]

- \( P^\# \triangleq \{ \bot_P \} \cup \{ p^\# \mid p^\# \in \mathbb{Z}^n \rightarrow \emptyset \} \cup \{ \top_P \} \)
- \( = \{ \bot_P \} \cup \{ p^\# \mid p^\# = \sum_i \omega_i \cdot f_i^\#, f_i^\# \in F^\# \} \cup \{ \top_P \} \)
  - where \( f^\# \equiv y = f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q \)

Urban&Miné - An Abstract Domain to Infer Ordinal-Valued Ranking Functions (to appear)
**join:** \( \sqcup_V \)

### Example

| \( v_1^\# \) | \( \triangleq \) | \([-\infty, +\infty]\) | \( \mapsto \) | \( \omega \cdot x_1 + x_2 \) |
| \( v_2^\# \) | \( \triangleq \) | \([-\infty, +\infty]\) | \( \mapsto \) | \( \omega \cdot (x_1 - 1) - x_2 \) |

\[ v_1^\# \sqcup_V v_2^\# \triangleq ? \mapsto ? \]

**backward assignments:** ASSIGN_V
join: $\sqcap_V$

**Example**

$$
\begin{align*}
\nu_1^\# &\triangleq [-\infty, +\infty] \mapsto \omega \cdot x_1 + x_2 \\
\nu_2^\# &\triangleq [-\infty, +\infty] \mapsto \omega \cdot (x_1 - 1) - x_2
\end{align*}
$$

$$
\nu_1^\# \sqcap_V \nu_2^\# \triangleq [-\infty, +\infty] \mapsto ?
$$

backward assignments: $\text{ASSIGN}_V$
• join: $\sqcup_V$

**Example**

\[
\begin{align*}
\nu_1^\# & \triangleq [\infty, +\infty] \rightarrow \omega \cdot x_1 + x_2 \\
\nu_2^\# & \triangleq [\infty, +\infty] \rightarrow \omega \cdot (x_1 - 1) - x_2 \\
\nu_1^\# \sqcup V \nu_2^\# & \triangleq [\infty, +\infty] \rightarrow 1 + 0
\end{align*}
\]

• backward assignments: $\text{ASSIGN}_V$
• join: $\sqcup_V$

**Example**

\[
\begin{align*}
 v_1^\# & \triangleq [-\infty, +\infty] \mapsto \omega \cdot x_1 + x_2 \\
 v_2^\# & \triangleq [-\infty, +\infty] \mapsto \omega \cdot (x_1 - 1) - x_2 \\
 v_1^\# \sqcup v_2^\# & \triangleq [-\infty, +\infty] \mapsto \omega \cdot x_1^1 + 0
\end{align*}
\]

• backward assignments: ASSIGN$\_V$
join: $\sqcup_V$

Example

\[
\begin{align*}
\nu_1^# &\triangleq [−\infty, +\infty] \mapsto \omega \cdot x_1 + x_2 \\
\nu_2^# &\triangleq [−\infty, +\infty] \mapsto \omega \cdot (x_1 - 1) - x_2 \\
\nu_1^# \sqcup_V \nu_2^# &\triangleq [−\infty, +\infty] \mapsto \omega \cdot (x_1 + 1) + 0
\end{align*}
\]

backward assignments: $\text{ASSIGN}_V$
- **join:** $\sqcap_V$

### Example

| \(v^\#_1\) | $\triangleq$ | \([-\infty, +\infty]\) | $\mapsto$ | $\omega \cdot x_1 + x_2$ |
| \(v^\#_2\) | $\triangleq$ | \([-\infty, +\infty]\) | $\mapsto$ | $\omega \cdot (x_1 - 1) - x_2$ |

\[
v^\#_1 \sqcap v^\#_2 \triangleq \left[ -\infty, +\infty \right] \mapsto \omega \cdot (x_1 + 1)
\]

- **backward assignments:** ASSIGN\(\_V\)
- join: $\sqcup_V$
- backward assignments: $\text{ASSIGN}_V$

**Example**

$$p^\# \triangleq \omega \cdot x_1 + x_2$$

$$\Downarrow \quad x_1 := ?$$

$$p^\# \triangleq ?$$
• join: $\sqcup_V$
• backward assignments: $\text{ASSIGN}_V$

### Example

\[
p^\# \triangleq \omega \cdot x_1 + x_2
\]

\[
\downarrow \quad x_1 ::= ?
\]

\[
p^\# \triangleq + 1
\]
- join: $\sqcup_V$
- backward assignments: ASSIGN$_V$

**Example**

\[
\begin{align*}
p^\# & \triangleq \omega \cdot x_1 + x_2 \\
\downarrow & x_1 := ? \\
p^\# & \triangleq + x_2 + 1
\end{align*}
\]
- join: $\sqcap_V$
- backward assignments: $\text{ASSIGN}_V$

**Example**

\[
p^\# \triangleq \omega \cdot x_1 + x_2
\]

$\Downarrow x_1 := \_ ?$

\[
p^\# \triangleq 1 + \omega \cdot 0 + x_2 + 1
\]
join: \( \sqcup \)

backward assignments: ASSIGN\(\_V\)

**Example**

\[
p\# \triangleq \omega \cdot x_1 + x_2
\]

\[
\downarrow \quad x_1 ::= ?
\]

\[
p\# \triangleq \omega^2 \cdot 1 + \omega \cdot 0 + x_2 + 1
\]
• join: $\sqcup_V$
• backward assignments: $\text{ASSIGN}_V$

**Example**

\[
p# \triangleq \omega \cdot x_1 + x_2 \\
\downarrow \quad x_1 := ? \\
p# \triangleq \omega^2 + x_2 + 1
\]
Theorem (Soundness)

*the abstract termination semantics is sound*

to prove the termination of programs
**Simple Loops**

**Example**

```plaintext
int : x_1, x_2

while \(1(x_1 \geq 0 \land x_2 \geq 0)\) do
  if \(2(?)\) then
    \(3x_1 := x_1 - 1\)
  else
    \(4x_2 := x_2 - 1\)
  fi
od \(5\)
```
Lexicographic Ranking Functions

Example

\[
\begin{align*}
\text{int} : & \ x_1, x_2 \\
\text{while} \ 1 (x_1 \geq 0 \land x_2 \geq 0) \ do \\
& \text{if} \ 2 (?) \ then \\
& \quad x_1 := x_1 - 1 \\
& \quad x_2 := ? \\
& \quad \text{else} \\
& \quad x_2 := x_2 - 1 \\
& \quad \text{fi} \\
& \text{od} \\
\end{align*}
\]

\[
f(x_1, x_2) = \begin{cases} 
1 & x_1 \leq 0 \lor x_2 \leq 0 \\
3x_2 + 2 & x_1 = 1 \\
\omega + 3x_2 + 9 & x_1 = 2 \\
\omega \cdot (x_1 - 1) + 7x_1 + 3x_2 - 5 & \text{otherwise}
\end{cases}
\]
Sufficient Preconditions for Termination

Example

\[
\text{int : } x \\
\text{while } (x < 10) \text{ do } \\
\quad x := 2 \times x \\
\text{od}
\]

\[
f(x) = \begin{cases} 
3 & 5 \leq x \leq 9 \\
1 & 10 \leq x 
\end{cases}
\]

\[
f(x) = \begin{cases} 
9 & x = 1 \\
7 & x = 2 \\
5 & 3 \leq x \leq 4 \\
3 & 5 \leq x \leq 9 \\
1 & 10 \leq x 
\end{cases}
\]
Sufficient Preconditions for Termination

Example

\[
\begin{align*}
\text{int} : & x \\
\text{while} (x < 10) \text{ do} & \\
& x := 2 \times x \\
\text{od}
\end{align*}
\]

\[
f(x) = \begin{cases} 
3 & 5 \leq x \leq 9 \\
1 & 10 \leq x 
\end{cases}
\]

\[
f(x) = \begin{cases} 
9 & x = 1 \\
7 & x = 2 \\
5 & 3 \leq x \leq 4 \\
3 & 5 \leq x \leq 9 \\
1 & 10 \leq x 
\end{cases}
\]
Non-Linear Computational Complexity

Example

\[
\text{int} : x_1, x_2
\]

1. \(x_1 := N\)

2. while \(x_1 \geq 0\) do

3. \(x_2 := N\)

4. while \(x_2 \geq 0\) do

5. \(x_2 := x_2 - 1\)

6. \(x_1 := x_1 - 1\)

7. od

\[
f(x_1, x_2) = \begin{cases} 
1 & x_1 \leq 0 \\
\omega + 2 & \text{otherwise}
\end{cases}
\]
http://www.di.ens.fr/~urban/FuncTion.html

- written in OCaml
- implemented on top of Apron\(^6\)

- forward reachability analysis to improve precision

---

Example

\[
\text{int : } x_1, x_2 \\
1 \ x_2 := 1 \\
\text{while } 2(\ x_1 < 10 ) \text{ do} \\
3 \ x_1 := x_1 + x_2 \\
\text{od} 4
\]

\(^6\)http://apron.cri.ensmp.fr/library/
written in OCaml
implemented on top of Apron\(^6\)

forward reachability analysis to improve precision

Example

\[
\begin{align*}
\text{int} &: x_1, x_2 \\
1& x_2 := 1 \\
\text{while} & (x_1 < 10) \text{ do} \\
3& x_1 := x_1 + x_2 \\
\text{od} & 4
\end{align*}
\]

\(^6\)http://apron.cri.ensmp.fr/library/
Experiments

**Benchmarks:** 38 programs

- 25 always terminating programs
- 13 conditionally terminating programs
- 9 simple loops
- 7 nested loops
- 13 non-deterministic programs

**Results:** proved 30 out of 38 programs

- proved 8 out of 9 simple loops
- proved 4 out of 7 nested loops
  - proved 2 out of 4 using ordinals
- proved 10 out of 13 non-deterministic programs
  - proved 5 out of 10 using ordinals
Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
- instances based on **natural-valued functions**
  - analysis not limited to simple loops
  - sufficient conditions for termination
- 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
- other liveness properties
- complexity analysis
Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
- instances based on **natural-valued functions**
  - analysis not limited to simple loops
  - sufficient conditions for termination
- 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**
- other liveness properties
- complexity analysis
Thank You!

Questions?
Example

```
int : x
while 1(x ≤ 10) do
  if 2(x > 6) then
    x := x + 2
  fi
fi
od
```
we map each point to a function of $x$ giving an upper bound on the steps before termination.

**Example**

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

A diagram is shown illustrating the flow of the program:

- **Start**: $x > 10$
- **Step 1**: $x \leq 6$
- **Step 2**: $x \leq 10$
- **Step 3**: $x > 6$
- **Step 4**: $x := x + 2$

The diagram visually represents the logic of the while loop and conditional statements.
Example

\begin{itemize}
\item int : x
\item while $^1(x \leq 10)$ do
\item \hspace{1em} if $^2(x > 6)$ then
\item \hspace{2em} $^3x := x + 2$
\item fi
\item od $^4$
\end{itemize}

we start at the end with 0 steps before termination

\begin{align*}
0 & \rightarrow 1 & \rightarrow 2 & \rightarrow 4 \rightarrow 0 \\
& x \rightarrow x > 10 \\
& \downarrow x \leq 6 \\
& \downarrow x \leq 10 \\
& \downarrow x > 6 \\
& x := x + 2
\end{align*}
Example

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

We take into account x > 10 and we have now 1 step to termination.
Example

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

if \( x > 6 \) then

\( x := x + 2 \)

fi

od

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

Example

\begin{align*}
\text{int} & : x \\
\text{while} \ & 1(x \leq 10) \ \text{do} \\
\quad \text{if} \ & 2(x > 6) \ \text{then} \\
\quad & 3x := x + 2 \\
\quad \text{fi} \\
\text{od} & 4
\end{align*}
Example

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

x \leq 6
\( x \leq 10 \)
\( x > 6 \)
\( x := x + 2 \)
we do the widening

Example

int : x
while 1(x ≤ 10) do
  if 2(x > 6) then
    3x := x + 2
  fi
od 4
Example

```plaintext
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**

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