An Abstract Interpretation Framework for Termination

Patrick Cousot

cims.nyu.edu/~pcousot/
www.di.ens.fr/~cousot/

Radhia Cousot

www.di.ens.fr/~rcousot/

Three principles

Principle I

Program verification methods (formal proof or static analysis methods) are abstract interpretations of a semantics of the programming language

Refinement to principle II

Safety as well as termination verification methods are abstract interpretations of a maximal trace semantics of the programming language
Comments on principle II

- This is well-known for instances of safety (like invariance) using prefix trace semantics (*1)
- This is proved in the paper for full safety (omitted in this presentation)
- New for termination

Comments on principle III

- Syntactic instances have been known for long (different variant functions for nested loops, Hoare logic for total correctness, ...)
- Semantic instances have been ignored for long (Burstall’s total correctness proof method using intermittent assertions) and very successful recently (Podelski-Rybalchenko)


New principle III

More expressive and powerful verification methods are derived by structuring the trace semantics (into a hierarchy of segments)

Maximal trace semantics
Maximal trace semantics

- Program $\mathcal{P} \xrightarrow{\tau^{\infty}} \tau^{\infty}[\mathcal{P}] \in \varphi(\Sigma^{+\phi})$

**Fixpoint maximal trace semantics**

- **Complete lattice**
  $\langle \varphi(\Sigma^{\infty}), \sqsubseteq, \Sigma^{\infty}, \Sigma^{*}, \cup, \cap \rangle$

- **Computational ordering**
  $(T_1 \subseteq T_2) \triangleq (T_1^{+} \subseteq T_2^{+}) \land (T_1^{\infty} \supseteq T_2^{\infty})$
  $T^{+} \triangleq T \cap \Sigma^{+}$
  $T^{\infty} \triangleq T \cap \Sigma^{\infty}$

- **Fixpoint semantics**
  $\tau^{\infty}[\mathcal{P}] = \text{lfp}_{\varphi} \subseteq \phi_{\tau}^{+\infty}[\mathcal{P}]$
  $\phi_{\tau}^{+\infty}[\mathcal{P}] T \triangleq \beta_{T}[\mathcal{P}] \sqcup \tau[\mathcal{P}] \downarrow T$

(Trace) properties

**Program properties**

- A program property $\mathcal{P}$ is the set of semantics which have this property:
  $\mathcal{P} \in \varphi(\varphi(\Sigma^{+\infty}))$

- Example:
  $\mathcal{P} = \{0, 1\}$

- Strongest property of program $\mathcal{P}$:
  $\{\tau^{\infty}[\mathcal{P}]\}$
Trace property abstraction

- Trace property abstraction:

\[ \alpha_\theta(P) \triangleq \bigcup P \langle \psi(\Sigma^{+\infty}), \subseteq \rangle \overset{\gamma_\theta}{\longrightarrow} \langle \psi(\Sigma^{+\infty}), \subseteq \rangle \]

- Example:

\[ P = \begin{array}{c}
| & 0 & 0 & 0 & 1 & | \\
| & - & - & - & - & |
\end{array} \]

\[ \alpha_\theta(P) = \begin{array}{c}
| & 0 & 0 & 1 & | \\
| & - & - & - & - & |
\end{array} \]

- The strongest trace property of a trace semantics is this trace semantics \( \alpha_\theta(\tau^{+\infty}[P]) = \tau^{+\infty}[P] \)

- Safety/liveness (termination) are trace properties, not general program properties

The Termination Problem

The termination proof problem

- Termination abstraction:

\[ \alpha'(T) \triangleq T \cap \Sigma^+ \]

- Termination proof:

\[ \alpha'(\tau^{+\infty}[P]) = \tau^{+\infty}[P] \]

- Termination proofs are not very useful since programs do not always terminate

Example

- Arithmetic mean of integers \( x \) and \( y \)

```java
while (x <> y) {
    x := x - 1;
    y := y + 1
}
```

- Does not always terminate e.g.

\( <x,y> = <1,0> \rightarrow <0,1> \rightarrow <-1,2> \rightarrow <-2,3> \rightarrow \ldots \)
The termination inference problem

- Determine a necessary condition for program termination and prove it sufficient

Example:

- (1) Under which necessary conditions

```plaintext
while (x <> y) {
    x := x - 1;
    y := y + 1
}
```
does terminate?

- (2) Prove these conditions to be sufficient

---

Potential termination

- For non-deterministic programs, we may be interested in potential termination

---

Definite termination abstraction

- or in definite termination

- Potential and definite termination coincide for deterministic programs. Only definite termination in this presentation.
Definite termination trace abstraction

- Prefix Abstraction

\[
\begin{align*}
\text{pf}(\sigma) & \triangleq \{\sigma' \in \Sigma^{+\omega} \mid \exists \sigma'' \in \Sigma^{+\omega} : \sigma = \sigma' \sigma''\} \\
\text{pf}(T) & \triangleq \bigcup \{\text{pf}(\sigma) \mid \sigma \in T\}.
\end{align*}
\]

- Definite termination abstraction

\[
\alpha_{\text{Mt}}(T) \triangleq \{\sigma \in T^+ \mid \text{pf}(\sigma) \cap \text{pf}(T^{\omega}) = \emptyset\}
\]

Definite termination

- The semantics/set of traces \( T \) \textit{definitely terminates} if and only if

\[
\alpha_{\text{Mt}}(T) = T
\]
Reachability analysis

- A forward invariance analysis infers states potentially reachable from initial states (by over-approximating an abstract fixpoint $\text{lfp } F$) (*)

Accessibility analysis

- A backward invariance analysis infers states potentially / definitely accessing final states (by over-approximating an abstract fixpoint $\text{lfp } B$) (*)

Combined reachability/accessibility analyses

- An iterated forward/backward invariance analysis infers reachable states potentially/definitely accessing final states (by over-approximating $\text{lfp } F \cap \text{lfp } B$) (*)

\[
\begin{align*}
X^0 &= \top \\
X^{2n+1} &= \text{lfp } \lambda Y. X^{2n} \cap F(Y) \\
X^{2n+2} &= \text{lfp } \lambda Y. X^{2n+1} \cap B(Y) \\
&\ldots
\end{align*}
\]

Example

- Arithmetic mean of two integers $x$ and $y$
  \[
  \{x=y\} \\
  \text{while } (x \neq y) \{ \\
  \{x=y+2\} \\
  x := x - 1; \\
  \{x=y+1\} \\
  y := y + 1 \\
  \{x=y\} \\
  \{x=y\}
  \}
\]

- Necessarily $x \geq y$ for proper termination
Example (cont'd)

- Arithmetic mean of two integers $x$ and $y$ (cont'd)

```java
while (x <> y) {
    k := k - 1;
    x := x - 1;
    y' := y + 1
    }
```

**Example (cont'd)**

- Arithmetic mean of two integers $x$ and $y$ (cont'd)

```java
{x = y + 2k, x >= y} while (x <> y) {
    {x = y + 2k, x >= y + 2}
    {x = y + 2k + 2, x >= y + 2}
    x := x - 1;
    {x = y + 2k + 1, x >= y + 1}
    y := y + 1
    {x = y + 2k, x >= y}
    } {x = y, k = 0}
assume (k = 0)
{x = y, k = 0}
```

- The difference $x - y$ must initially be even for proper termination

Observations

- $k$ provides the value of the variant function in the sense of Turing/Floyd
- The constraints on $k$ (hence the variant function) are computed backwards
  $\implies$ a backward analysis should be able to infer the variant function

The Turing-Floyd termination proof method

The hierarchy of termination semantics

- **Maximal trace** concrete backward trace semantics
  \[ \alpha^{Mt} \]

  Definite termination abstract backward trace semantics
  \[ \alpha^W \]

  Weakest pre-condition abstract backward state semantics (termination domain)
  \[ \alpha^{rk} \]

  Variant function abstract ordinal backward semantics

Fixpoint definition of the variant function

- We now apply the **abstract interpretation** methodology:
  - The maximal trace semantics has a **fixpoint definition**
  - The variant function is an **abstraction** of the maximal trace semantics
  - With this abstraction, we construct a **fixpoint definition of the abstract variant semantics**

  \[ \implies Fixpoint \text{ induction provides a termination proof method} \]

  \[ \implies \text{Further abstractions and widenings provide a static analysis method} \]

The ranking abstraction

\[ \alpha^{rk}(r) \in \varphi(\Sigma \times \Sigma) \mapsto (\Sigma \not\ni \emptyset) \]

\[ \alpha^{rk}(r)s \triangleq 0 \text{ when } \forall s' \in \Sigma : (s, s') \notin r \]

\[ \alpha^{rk}(r)s \triangleq \sup \left\{ \alpha^{rk}(r)s' + 1 \mid \exists s' \in \Sigma : (s, s') \in r \land \forall s' \in \Sigma : (s, s') \in r \implies s' \in \text{dom}(\alpha^{rk}(r)) \right\} \]

- \( \alpha^{rk}(r) \) extracts the well-founded part of relation \( r \)
- provides the rank of the elements \( s \) in its domain
- strictly decreasing with transitions of relation \( r \)

\[ \implies \text{the most precise variant function} \]

Example 1

- **Maximal trace semantics:**

- **Ranking fixpoint iterates:**
Example II

- Program
  int x; while (x > 0) { x = x - 2; }
- Fixpoint
  \( \nu = \text{lfp}_0^{\nu} \phi_r^{\nu}[P] \)
  \( \phi_r^{\nu}[P](\nu)x = \{ x \leq 0 \ ? 0 \ ? \sup \{ \nu(x-2) + 1 \mid x - 2 \in \text{dom}(\nu) \} \} \)
- Iterates
  \( \nu^0 = \emptyset \)
  \( \nu^1 = \lambda x \in [-\infty, 0] \cdot 0 \)
  \( \nu^2 = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2] \cdot 1 \)
  \( \nu^3 = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2] \cdot 1 \cup \lambda x \in [3, 4] \cdot 2 \)
  \( \ldots \)
  \( \nu^\omega = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2 \times (n-1)] \cdot (x+1) + 2 \)
  \( \ldots \)
  \( \nu^\omega = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, +\infty] \cdot (x+1) + 2 \).

Example III

- Program:
  \{ even(x-y), x >= y \}
  while (x <> y) {
    x := x - 1;
    y := y + 1
  }
  (x = y)

- Iterates (linear abstraction):
  \( \exists k : \nu(x, y) = k, x - y - 2k = 0, k \geq 0 \)

Example IV

- In general a widening is needed to enforce convergence
- Program: int x; while (x > 0) { x = x - 2; }
- Iterates with widening:
  \( \nu_0 = \lambda x \in [-\infty, +\infty] \cdot \bot \)
  \( \nu_1 = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, +\infty] \cdot \bot \)
  \( \nu_2 = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2] \cdot 1 \cup \lambda x \in [3, +\infty] \cdot \bot \)
  \( \nu_3 = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2] \cdot 1 \cup \lambda x \in [3, 4] \cdot 2 \)
  \( \ldots \)
  \( \nu^\omega = \lambda x \in [-\infty, 0] \cdot 0 \cup \lambda x \in [1, 2 \times (n-1)] \cdot (x+1) \cdot 1 \)
  \( \ldots \)
  \( \nu^\omega = \nu^\omega. \)
Objection I: Turing/Floyd's method goes forward not backward!

- An analysis can be inverted using auxiliary variables

```plaintext
int x;
while (c(x)) {
  x := f(x)
}
```

```plaintext
int x, x0;
while (c(x)) {
  x0 := x;
  x := f(x)
}
```

Backward variant \( v \):

\[
\begin{align*}
& v(x_{\text{before}}) = v(x_{\text{after}}) + 1 \\
\iff & v(x_{\text{before}}) = v(f(x_{\text{before}})) + 1
\end{align*}
\]

Forward variant \( v \):

\[
\begin{align*}
& v(x_0) = v(x) + 1 \\
\iff & v(x_0) = v(f(x_0)) + 1
\end{align*}
\]

Objection II: you need ordinals!

- Example: \( x := ?; \) while \((x \geq 0)\) do \( x := x - 1 \) od

- Ranking:

```
0 ------ 0 ------ 1 ------ 2 ------ n ------ ...
```

- To avoid transfinite ordinals/well-founded orders for unbounded non-determinism, the computations need to be structured!

Structuring trace semantics with segments

Floyd/Turing termination proof method

- Trivial postfix structuring of traces into segments

- Also used for termination of straight-line code (no need for variant functions)
**Floyd with nested loops**

- The trace semantics is recursively structured in segments according to loop nesting

Prove termination of outer loop assuming termination of body/nested inner loops

(equivalent to lexicographic orderings)

---

**Hoare logic**

- The trace semantics is recursively structured in segments according to the program syntax

- while (c) { b; a }...

---

**Burstall’s proof method by hand-simulation and a little induction**

- Program
  ```
  do odd(x) and x ≥ 3 → x := x+1
  □ even (x) and x ≥ 2 → x := x/2
  od
  ```

- Proof chart

---

**Burstall’s proof method by hand-simulation and a little induction**

- Iterative program but recursive proof structure

- Inductive trace cover by segments
Podelski-Rybalchenko

- Transition invariants are abstractions of trace segments covering the trace semantics by their extremities

Termination based on Ramsey theorem on colored edges of a complete graph, no recursive structure

Trace semantics segmentation

- Recursive trace segmentation

**Definition 2.** An inductive trace segment cover of a non-empty set $\chi \in \varphi(\Sigma^{\infty})$ of traces is a set $C \in \mathcal{C}(\chi)$ of members $B$ of $\varphi(\alpha^+(\chi))$ such that

1. if $SS' \in C$ then $S \in C$ (prefix-closure)
2. if $S \in C$ then $\exists S' : S = \chi S'$ (root)
3. if $SBB' \in C$ then $B \supseteq B'$ (well-foundedness)
4. if $SBB' \in C$ then $B \subseteq \bigcup_{SBB' \in C} B'$ (cover).

- Proof by induction on the possibly infinite but well-founded trace segmentation tree

- Orthogonal to proofs on segment sets (using variant functions, Ramsey theorem, etc.)

Rely-guarantee

- Example of abstraction of segments into rely-guarantee/contracts state properties:

Conclusion
More in the paper

- The presentation was deliberately intended to be simple and intuitive
- The paper provides
  - More topics (e.g. abstract trace covers/proofs)
  - More technical details (e.g. fixpoint definitions of the various abstract termination semantics)
  - More examples (e.g. a more detailed piecewise linear termination abstraction)

Contributions

- Formalization of existing termination proof methods as abstract interpretations
- Pave the way for new backward termination static analysis methods (going beyond reduction of termination to safety analyzes)
- The new concept of trace semantics segmentation is not specific to termination and applies to all specification/verification/analysis methods

Future work

- Abstract domains for termination
- Semantic techniques for segmentation inference
- Eventuality verification/static analysis
- (General) liveness\(^(*)\) verification/static analysis

\(^(*)\) Beyond LTL, as defined in

Beyond LTL, as defined in