AAA528: Computational Logic

Lecture 10 — Invariant Generation (Static Analysis)

Hakjoo Oh
2026 Spring

Program Verification vs. Program Analysis

Essentially the same things with different trade-offs:

- Program verification
  - Pros: powerful to prove properties
  - Cons: hardly automated
- Program analysis
  - Pros: fully automatic
  - Cons: focus on rather weak properties

Contents

- Symbolic analysis
  - concrete, non-terminating
- Interval analysis
  - abstract, non-relational
- Octagon analysis
  - abstract, relational

Program Representation

Control-flow graph $(\mathbb{C}, \rightarrow)$

- $\mathbb{C}$: the set of program points in the program
- $(\rightarrow) \subseteq \mathbb{C} \times \mathbb{C}$: the control-flow relation
  - $c \rightarrow c'$: c is a predecessor of c'
- Each control-flow edge $c \rightarrow c'$ is associated with a command, denoted $\textbf{cmd}(c \rightarrow c')$:

$$cmd \quad \rightarrow \quad v := e \mid \text{assume } c \mid cmd_1; cmd_2$$

Weakest Precondition

Weakest precondition transformer

$$\textbf{wp} : \text{FOL} \times \text{stmts} \rightarrow \text{FOL}$$

computes the most general precondition of a given postcondition and program statement:

- $\textbf{wp}(F, \text{assume } c) \iff c \rightarrow F$
- $\textbf{wp}(F[v], v := e) \iff F[e]$
- $\textbf{wp}(F, S_1; \ldots; S_n) \iff \textbf{wp}(\textbf{wp}(F, S_n), S_1; \ldots; S_{n-1})$

Strongest Postcondition

Strongest postcondition transformer

$$\textbf{sp} : \text{FOL} \times \text{stmts} \rightarrow \text{FOL}$$

computes the most specific postcondition of a given precondition and program statement:

- $\textbf{sp}(F, \text{assume } c) \iff c \wedge F$
- $\textbf{sp}(F[v], v := e[v]) \iff \exists v^0.\ v = e[v^0] \wedge F[v^0]$
- $\textbf{sp}(F, S_1; \ldots; S_n) \iff \textbf{sp}(\textbf{sp}(F, S_1), S_2; \ldots; S_n)$

Examples

$$
\begin{aligned}
& \mathbf{sp}(i \geq n, i := i + k) \\
\iff & \exists i^0.\ i = i^0 + k \wedge i^0 \geq n \\
\iff & i - k \geq n
\end{aligned}
$$

$$
\begin{aligned}
& \mathbf{sp}(i \geq n, \text{assume } k \geq 0;\ i := i + k) \\
\iff & \mathbf{sp}(\mathbf{sp}(i \geq n, \text{assume } k \geq 0), i := i + k) \\
\iff & \mathbf{sp}(i \geq n \wedge k \geq 0, i := i + k) \\
\iff & \exists i^0.\ i = i^0 + k \wedge i^0 \geq n \wedge k \geq 0 \\
\iff & i - k \geq n \wedge k \geq 0
\end{aligned}
$$

Inductive Map

- The goal of static analysis is to find a map

$$\boldsymbol{T} : \mathbb{C} \rightarrow \text{FOL}$$

that stores inductive invariants for each program point and is implied by the precondition:

$$\boldsymbol{F_{pre}} \implies \boldsymbol{T(c_0)}.$$

- If the result $\boldsymbol{T(c_{exit})}$ implies the postcondition

$$\boldsymbol{T(c_{exit})} \implies \boldsymbol{F_{post}}$$

the function obeys the specification.

Forward Symbolic Analysis Procedure

- Sets of reachable states are represented by formulas.
- Strongest postcondition (**sp**) executes statements over formulas.

$$
\begin{array}{l}
\boldsymbol{W := \{c_0\}} \\
\boldsymbol{T(c_0) := F_{pre}} \\
\boldsymbol{T(c) := \bot} \text{ for } \boldsymbol{c \in \mathbb{C} \setminus \{c_0\}} \\
\text{while } \boldsymbol{W \neq \emptyset} \\
\quad \boldsymbol{c := \textbf{Choose}(W)} \\
\quad \boldsymbol{W := W \setminus \{c\}} \\
\quad \text{foreach } \boldsymbol{c' \in \textbf{succ}(c)} \\
\quad\quad \boldsymbol{F := \textbf{sp}(T(c), \textbf{cmd}(c \rightarrow c'))} \\
\quad\quad \text{if } \boldsymbol{F \not\Longrightarrow T(c')} \\
\quad\quad\quad \boldsymbol{T(c') := T(c') \vee F} \\
\quad\quad\quad \boldsymbol{W := W \cup \{c'\}} \\
\quad \text{done} \\
\text{done}
\end{array}
$$

Issues

- The implication checking

$$F \not\Rightarrow T(c')$$

  is undecidable in general. The underlying logic must be restricted to a decidable theory or fragment.
- Nontermination of loops.

Example

$$
\begin{array}{l}
@c_0 : i = 0 \wedge n \geq 0; \\
\text{while } @c_1 \\
(i < n)\ \{ \\
\qquad i := i + 1; \\
\} \\
@c_2 : i = n
\end{array}
$$

Initial map:

$$
\begin{aligned}
T(c_0) &\iff i = 0 \wedge n \geq 0 \\
T(c_1) &\iff \bot
\end{aligned}
$$

Following basic path $c_0 \to c_1$:

$$
\begin{aligned}
T(c_0) &\iff i = 0 \wedge n \geq 0 \\
T(c_1) &\iff T(c_1) \vee i = 0 \wedge n \geq 0 \iff i = 0 \wedge n \geq 0
\end{aligned}
$$

Example

Following basic path $c_1 \to c_1$:

1. Symbolic execution:

$$\begin{aligned}
& \mathbf{sp}(T(c_1), \text{assume } i < n; i := i + 1) \\
\iff & \mathbf{sp}(i = 0 \wedge n \geq 0, \text{assume } i < n; i := i + 1) \\
\iff & \mathbf{sp}(i < n \wedge i = 0 \wedge n \geq 0, i := i + 1) \\
\iff & \exists i^0.\ i = i^0 + 1 \wedge i^0 < n \wedge i^0 = 0 \wedge n \geq 0 \\
\iff & i = 1 \wedge n \geq 1
\end{aligned}$$

2. Checking the implication:

$$i = 1 \wedge n \geq 1 \implies i = 0 \wedge n \geq 0$$

3. Join the result:

$$T(c_1) \iff (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1)$$

Example

At the end of the next iteration:

$$T(c_1) \iff (i = 0 \land n \geq 0) \lor (i = 1 \land n \geq 1) \lor (i = 2 \land n \geq 2)$$

and at the end of kth iteration:

$$T(c_1) \iff (i = 0 \land n \geq 0) \lor (i = 1 \land n \geq 1) \lor \cdots \lor (i = k \land n \geq k)$$

This process does not terminate because

$$(i = k \land n \geq k) \not\Longrightarrow (i = 0 \land n \geq 0) \lor \cdots \lor (i = k-1 \land n \geq k-1)$$

for any k. However,

$$0 \leq i \leq n$$

is an obvious inductive invariant that proves the postcondition:

$$0 \leq i \leq n \land i \geq n \implies i = n.$$

Addressing the Issues

- Unsound approach, e.g., unrolling loops for a fixed number
  - incapable of verifying properties but still useful for bug-finding
- Sound approach ensures correctness but cannot be complete.
- **Abstract interpretation** is a general method for obtaining sound and computable static analysis.
  - abstract domain
  - abstract semantics
  - widening and narrowing

1. Choose an Abstract Domain

The abstract domain D is a restricted subset of formulas; each member $d \in D$ represents a set of program states: e.g.,

- In the interval abstract domain D_I, a domain element $d \in D_I$ is a conjunction of constraints of the forms

$$c \leq x \qquad \text{and} \qquad x \leq c$$

- In the octagon abstract domain D_O, a domain element $d \in D_I$ is a conjunction of constraints of the forms

$$\pm x_1 \pm x_2 \leq c$$

- In the Karr's abstract domain D_K, a domain element $d \in D_K$ is a conjunction of constraints of the forms

$$c_0 + c_1 x_1 + \cdots c_n x_n = 0$$

2. Construct an Abstraction Function

The abstraction function:

$$\alpha_D : \text{FOL} \rightarrow D$$

such that $F \implies \alpha_D(F)$. For example, the assertion

$$F : i = 0 \wedge n \geq 0$$

can be represented in the interval abstract domain by

$$\alpha_{D_I}(F) : 0 \leq i \wedge i \leq 0 \wedge 0 \leq n$$

and in Karr's abstract domain by

$$\alpha_{D_K}(F) : i = 0$$

3. Define an Abstract Strongest Postcondition

Define an abstract strongest postcondition operator $\widehat{\mathbf{sp}}_D$, also known as abstract semantics or transfer function:

$$\widehat{\mathbf{sp}}_D : D \times \text{stmts} \to D$$

such that $\widehat{\mathbf{sp}}_D$ over-approximates $\mathbf{sp}$:

$$\mathbf{sp}(F, S) \implies \widehat{\mathbf{sp}}_D(F, S).$$

3. Define an Abstract Strongest Postcondition

For example, the strongest postcondition for assume:

$$\mathbf{sp}(F, \text{assume } c) \iff c \wedge F$$

is abstracted by

$$\widehat{\mathbf{sp}}(F, \text{assume } c) \iff \alpha_D(c) \sqcap_D F$$

where abstract conjunction $\sqcap_D : D \times D \to D$ is such that

$$F_1 \wedge F_2 \implies F_1 \sqcap_D F_2.$$

When the domain D consists of conjunctions of constraints of some form, $\sqcap_D$ is exact and equals to the usual conjunction $\wedge$:

$$F_1 \wedge F_2 \iff F_1 \sqcap_D F_2.$$

4. Define Abstract Disjunction and Implication Checking

- Define abstract disjunction $\sqcup_D : D \times D \to D$ such that

$$F_1 \vee F_2 \implies F_1 \sqcup_D F_2$$

  Usually abstract disjunction is not exact.
- With a proper abstract domain, the implication checking

$$F \not\Rightarrow T(c_k)$$

  can be performed by a custom solver without querying a full SMT solver.

5. Define Widening

A widening operator $\bigtriangledown_D$ is a binary operator

$$\bigtriangledown_D : D \times D \to D$$

such that

$$F_1 \vee F_2 \implies F_1 \bigtriangledown_D F_2$$

and the following property holds. For all increasing sequence $F_1, F_2, F_3, \ldots$ (i.e. $F_i \implies F_{i+1}$ for all i), the sequence G_i defined by

$$G_i = \begin{cases} F_1 & \text{if } i = 1 \\ G_{i-1} \bigtriangledown_D F_i & \text{if } i > 1 \end{cases}$$

eventually converges:

$$\text{for some } k \text{ and for all } i \geq k, G_i \iff G_{i+1}.$$

Abstract Interpretation Algorithm

$W := \{c_0\}$
$T(c_0) := \alpha_D(F_{pre})$
$T(c) := \bot$ for $c \in \mathbb{C} \setminus \{c_0\}$
while $W \neq \emptyset$
    $c := \textbf{Choose}(W)$
    $W := W \setminus \{c\}$
    foreach $c' \in \textbf{succ}(c)$
        $F := \widehat{\textbf{sp}}(T(c), \textbf{cmd}(c \to c'))$
        if $F \not\Longrightarrow T(c')$
            if widening is needed
                $T(c') := T(c') \bigtriangledown (T(c') \sqcup_D F)$
            else
                $T(c') := T(c') \sqcup_D F$
            $W := W \cup \{c'\}$
    done
done

Program Analysis

Automated techniques for computing program invariants:

- Generic symbolic analysis procedure
- Abstraction examples: Interval and octagon analyses