12. Invariant Generation

Invariant Generation

Discover inductive assertions of programs

- General procedure
- Concrete analysis

  - interval analysis

    invariants of form

    $$c \leq v \text{ or } v \leq c$$

    for program variable v and constant c
  - Karr's analysis

    invariants of form

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

    for program variables x_i and constants c_i

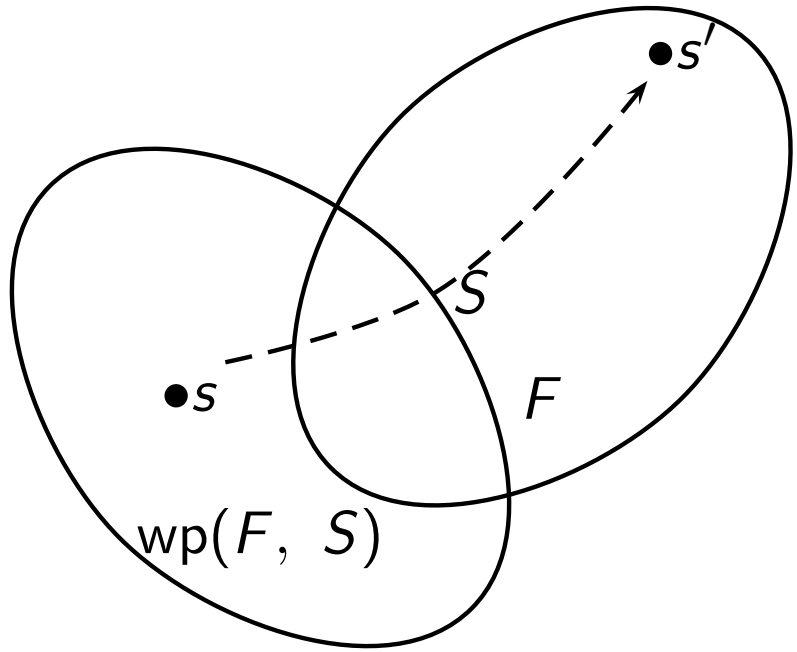
Other invariant generation algorithms in literature:

- linear inequalities

  $c_0 + c_1 x_1 + \cdots + c_n x_n \leq 0$
- polynomial equalities and inequalities

Background

Weakest Precondition



For FOL formula F and program statement S, the weakest precondition wp(F, S) is a FOL formula s.t. if for state s

$$s \models \text{wp}(F, S)$$

and if statement S is executed on state s to produce state s', then

$$s' \models F .$$

In other words, the weakest precondition moves a formula backwards over a series of statements:
for F to hold after executing $S_1; \ldots; S_n$,
$\mathrm{wp}(F,\ S_1; \ldots; S_n)$ must hold before executing the statements.

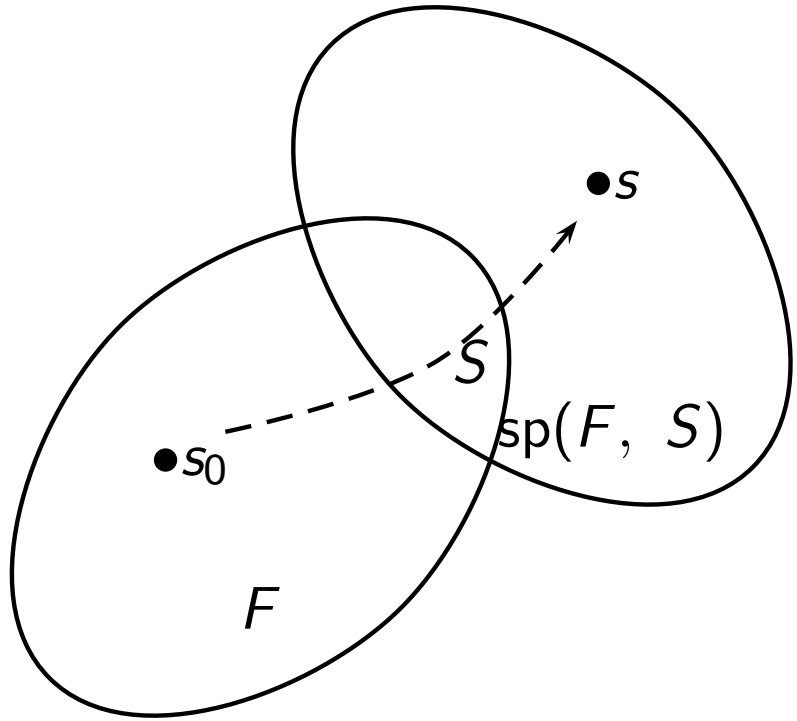
For assume and assignment statements

- $\mathrm{wp}(F,\ \texttt{assume}\ c) \ \Leftrightarrow\ c \ \rightarrow\ F$, and
- $\mathrm{wp}(F[v],\ v := e) \ \Leftrightarrow\ F[e]$;

and on sequences of statements $S_1; \ldots; S_n$:

$$\mathrm{wp}(F,\ S_1; \ldots; S_n) \ \Leftrightarrow\ \mathrm{wp}(\mathrm{wp}(F,\ S_n),\ S_1; \ldots; S_{n-1})\ .$$

Strongest Postcondition



For FOL formula F and program statement S, the strongest postcondition $\mathrm{sp}(F,\ S)$ is a FOL formula s.t. if s is the current state and

$$s \models \mathrm{sp}(F,\ S)$$

then statement S was executed from a state s_0 s.t.

$$s_0 \models F\ .$$

- On assume statements,

$$\mathrm{sp}(F,\ \texttt{assume}\ c)\ \Leftrightarrow\ c\ \wedge\ F\ ,$$

  for if program control makes it past the statement, then c must hold.
- Unlike in the case of wp, there is no simple definition of sp on assignments:

$$\mathrm{sp}(F[v],\ v := e[v])\ \Leftrightarrow\ \exists v^0.\ v = e[v^0]\ \wedge\ F[v^0]\ .$$

- On a sequence of statements $S_1; \ldots; S_n$:

$$\mathrm{sp}(F,\ S_1; \ldots; S_n)\ \Leftrightarrow\ \mathrm{sp}(\mathrm{sp}(F,\ S_1),\ S_2; \ldots; S_n)\ .$$

Example: Compute

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

since $i^0 = i - k$.

Example: Compute

$$\begin{aligned}
&\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0;\ i := i + k) \\
&\quad \Leftrightarrow\ \mathrm{sp}(\mathrm{sp}(i \geq n,\ \texttt{assume}\ k \geq 0),\ i := i + k) \\
&\quad \Leftrightarrow\ \mathrm{sp}(k \geq 0\ \wedge\ i \geq n,\ i := i + k) \\
&\quad \Leftrightarrow\ \exists i^0.\ i = i^0 + k\ \wedge\ k \geq 0\ \wedge\ i^0 \geq n \\
&\quad \Leftrightarrow\ k \geq 0\ \wedge\ i - k \geq n
\end{aligned}$$

Verification Condition

VCs in terms of wp:

$$\{F\}S_1;\ldots;S_n\{G\}:\ F\ \Rightarrow\ \mathrm{wp}(G,\ S_1;\ldots;S_n)\ .$$

VCs in terms of sp:

$$\{F\}S_1;\ldots;S_n\{G\}:\ \mathrm{sp}(F,\ S_1;\ldots;S_n)\ \Rightarrow\ G\ .$$

Static Analysis: basic definition

- Program P with locations $\mathcal{L}$ (L_0 — initial location)
- Cutset of $\mathcal{L}$
  each path from one cutpoint (location in the cutset) to the next cutpoint is basic path (does not cross loops)
- Assertion map
  $$\mu : \mathcal{L} \to \text{FOL}$$
  (map from $\mathcal{L}$ to first-order assertions).
  It is inductive (inductive map) if for each basic path

$$\frac{}{\begin{array}{l} L_i : \ @\ \mu(L_i) \\ S_i; \\ \vdots \\ S_j; \\ L_j : \ @\ \mu(L_j) \end{array}} \ (\cdot)$$

for $L_i, L_j \in \mathcal{L}$, the verification condition

$$\{\mu(L_i)\} S_i; \ldots; S_j \{\mu(L_j)\} \qquad \text{(VC)}$$

is valid.

Invariant Generation

Find inductive assertion maps μ s.t. the $\mu(L_i)$ satisfies (VC) for all basic paths.

Method: Symbolic execution (formward propagation)

- Initial map μ_0:

$$\begin{array}{lll} \mu(L_0) := F_{\text{pre}} \ , & \text{and} & \\ \mu(L) := \bot & \text{for} & L \in \mathcal{L} \setminus \{L_0\}. \end{array}$$

- Maintain set $S \subseteq \mathcal{L}$ of locations that still need processing. Initially, let $S = \{L_0\}$. Terminate when $S = \emptyset$.
- Iteration i: We have so far constructed μ_i. Choose some $L_j \in S$ to process and remove it from S.

For each basic path (starting at L_j)

$(\cdot)$

L_j : @ $\mu(L_j)$
S_j;
⋮
S_k;
L_k : @ $\mu(L_k)$

compute and set

$$\mu(L_k) \;\Leftrightarrow\; \mu(L_k) \;\vee\; \mathrm{sp}(\mu(L_j),\; S_j; \ldots; S_k)$$

If

$$\mathrm{sp}(\mu(L_j),\; S_j; \ldots; S_k) \;\Rightarrow\; \mu_i(L_k)$$

that is, if sp does not represent new states not already represented in $\mu_i(L_k)$, then $\mu_{i+1}(L_k) \;\Leftrightarrow\; \mu_i(L_k)$ (nothing new is learned)

Otherwise add L_k to S.

For all other locations $L_\ell \in \mathcal{L}, \mu_{i+1}(L_\ell) \;\Leftrightarrow\; \mu_i(L_\ell)$

When $S = \emptyset$ (say iteration i^*), then μ_{i^*} is an inductive map.

The algorithm

$$
\begin{array}{l}
\texttt{let } \textsc{ForwardPropagate}\ P\ F_{\text{pre}}\ \mathcal{L} = \\
\quad S := \{L_0\}; \\
\quad \mu(L_0) := F_{\text{pre}}; \\
\quad \mu(L) := \bot \texttt{ for } L \in \mathcal{L} \setminus \{L_0\}; \\
\quad \texttt{while } S \neq \emptyset \texttt{ do} \\
\qquad \texttt{let } L_j = \textsc{choose}\ S \texttt{ in} \\
\qquad S := S \setminus \{L_j\}; \\
\qquad \texttt{foreach } L_k \in \text{succ}(L_j) \texttt{ do} \quad \left[\begin{array}{l} L_k \in \text{succ}(L_j) \text{ is a } \textbf{successor} \text{ of } L_j \\ \text{if there is a basic path from } L_j \text{ to } L_k \end{array}\right] \\
\qquad\quad \texttt{let } F = \text{sp}(\mu(L_j),\ S_j; \ldots; S_k) \texttt{ in} \\
\qquad\quad \texttt{if } F \not\Rightarrow \mu(L_k) \\
\qquad\quad \texttt{then } \mu(L_k) := \mu(L_k) \vee F; \\
\qquad\qquad\quad S := S \cup \{L_k\}; \\
\qquad \texttt{done;} \\
\quad \texttt{done;} \\
\quad \mu
\end{array}
$$

Problem: algorithm may not terminate

Example: Consider loop with integer variables i and n:

$@L_0 : i = 0 \ \wedge \ n \geq 0;$
`while`
$@L_1 : ?$
$(i < n)$ {
$i := i + 1;$
}

There are two basic paths:

(1)

$@L_0 : \ i = 0 \ \wedge \ n \geq 0;$
$@L_1 : ?;$

and

(2)

$@L_1 : ?;$
`assume` $i < n;$
$i := i + 1;$
$@L_1 : ?;$

- Initially,

$$\boxed{\begin{array}{lcl} \mu(L_0) & \Leftrightarrow & i = 0 \ \wedge \ n \geq 0 \\ \mu(L_1) & \Leftrightarrow & \bot \end{array}}$$

- Following path **(1)** results in setting

$$\mu(L_1) := \mu(L_1) \ \vee \ (i = 0 \ \wedge \ n \geq 0)$$

$\mu(L_1)$ was $\bot$, so that it becomes

$$\boxed{\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0 \ .}$$

- On the next iteration, following path **(2)** yields

$$\mu(L_1) := \mu(L_1) \ \vee \ \mathrm{sp}(\mu(L_1), \ \texttt{assume}\ i < n; \ i := i + 1) \ .$$

Currently $\mu(L_1) \ \Leftrightarrow \ i = 0 \ \wedge \ n \geq 0$, so

$$\begin{array}{rcl} F: \ \mathrm{sp}(i = 0 \ \wedge \ n \geq 0, \ \texttt{assume}\ i < n; \ i := i + 1) & & \\ \Leftrightarrow \ \mathrm{sp}(i < n \ \wedge \ i = 0 \ \wedge \ n \geq 0, \ i := i + 1) & & \\ \Leftrightarrow \ \exists i^0 .\ i = i^0 + 1 \ \wedge \ i^0 < n \ \wedge \ i^0 = 0 \ \wedge \ n \geq 0 & & \\ \Leftrightarrow \ i = 1 \ \wedge \ n > 0 & & \end{array}$$

Since the implication

$$\underbrace{i = 1 \ \wedge \ n > 0}_{F} \ \Rightarrow \ \underbrace{i = 0 \ \wedge \ n \geq 0}_{\mu(L_1)}$$

is invalid,

$$\mu(L_1) \ \Leftrightarrow \ \underbrace{(i = 0 \ \wedge \ n \geq 0)}_{\mu(L_1)} \ \vee \ \underbrace{(i = 1 \ \wedge \ n > 0)}_{F}$$

at the end of the iteration.

- At the end of the next iteration,

$$\mu(L_1) \ \Leftrightarrow \ \underbrace{(i = 0 \ \wedge \ n \geq 0) \ \vee \ (i = 1 \ \wedge \ n > 0)}_{\mu(L_1)} \ \vee \ \underbrace{(i = 2 \ \wedge \ n > 1)}_{F}$$

- At the end of the kth iteration,

$$\boxed{\mu(L_1) \;\Leftrightarrow\; \begin{array}{c}(i = 0 \;\wedge\; n \geq 0) \;\vee\; (i = 1 \;\wedge\; n \geq 1) \\ \vee\; \cdots \;\vee\; (i = k \;\wedge\; n \geq k)\end{array}}$$

  It is never the case that the implication

$$\begin{array}{c} i = k \;\wedge\; n \geq k \\ \Downarrow \\ (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \cdots \vee (i = k - 1 \wedge n \geq k - 1) \end{array}$$

  is valid, so the main loop of while never finishes.

- However, it is obvious that

$$0 \leq i \leq n$$

  is an inductive annotation of the loop.

Solution: Abstraction

A state s is reachable for program P if it appears in some computation of P.

The problem is that ForwardPropagate computes the exact set of reachable states.

Inductive annotations usually over-approximate the set of reachable states: every reachable state s satisfies the annotation, but other unreachable states can also satisfy the annotation.

Abstract interpretation cleverly over-approximate the reachable state set to guarantee termination.

Abstract interpretation is constructed in 6 steps.

Step 1: Choose an abstract domain D.

The **abstract domain** D is a syntactic class of Σ-formulae of some theory T.

- **interval abstract domain** D_{I} consists of conjunctions of $\Sigma_{\mathbb{Q}}$-literals of the forms

$$c \leq v \quad \text{and} \quad v \leq c \ ,$$

  for constant c and program variable v.

- **Karr's abstract domain** D_{K} consist of conjunctions of $\Sigma_{\mathbb{Q}}$-literals of the form

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

  for constants $c_0, c_1, \ldots, c_n$ and variables $x_1, \ldots, x_n$.

Step 2: Construct a map from FOL formulae to D.

Define

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

to map a FOL formula F to element $\nu_D(F)$ of D, with the property that for any F,

$$F \;\Rightarrow\; \nu_D(F)\ .$$

Example:

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

at L_0 of the loop can be represented in the interval abstract domain by

$$\nu_{D_{\text{I}}}(F) :\; 0 \leq i \;\wedge\; i \leq 0 \;\wedge\; 0 \leq n$$

and in Karr's abstract domain by

$$\nu_{D_{\text{K}}}(F) :\; i = 0$$

with some loss of information.

Step 3: Define an abstract sp**.**

Define an **abstract strongest postcondition** $\overline{\text{sp}}_D$ for assumption and assignment statements such that

$$\text{sp}(F,\ S)\ \Rightarrow\ \overline{\text{sp}}_D(F,\ S)\quad\text{and}\quad\overline{\text{sp}}_D(F,\ S)\in D$$

for statement S and $F \in D$.

- statement `assume` c:

$$\text{sp}(F,\ \texttt{assume}\ c)\ \Leftrightarrow\ c\ \wedge\ F\ .$$

Conjunction $\wedge$ is used.
Define abstract conjunction $\sqcap_D$, such that

$$F_1 \wedge F_2\ \Rightarrow\ F_1\ \sqcap_D\ F_2\quad\text{and}\quad F_1\ \sqcap_D\ F_2 \in D$$

for $F_1, F_2 \in D$. Then if $F \in D$,

$$\overline{\text{sp}}_D(F,\ \texttt{assume}\ c)\ \Leftrightarrow\ \nu_D(c)\ \sqcap_D\ F\ .$$

If the abstract domain D consists of conjunctions of literals, $\sqcap_D$ is just $\wedge$. For example, in the interval domain,

$$\overline{\text{sp}}_{D_\text{I}}(F,\ \texttt{assume}\ c)\ \Leftrightarrow\ \nu_{D_\text{I}}(c)\ \wedge\ F\ .$$

- assignment statements:
  More complex, for suppose that we use the standard definition

$$\mathrm{sp}(F[v],\ v := e[v]) \ \Leftrightarrow\ \underbrace{\exists v^0.\ v = e[v^0]\ \wedge\ F[v^0]}_{G}\ ,$$

  which requires existential quantification. Then, later, when we compute the validity of

$$G\ \Rightarrow\ \mu(L)\ ,\quad \text{i.e.,}\quad \forall \overline{b}.\ G\ \rightarrow\ \mu(L)\ ,$$

  $\mu(L)$ can contain existential quantification, resulting in a quantifier alternation. Most decision procedures, apply only to quantifier-free formulae. Therefore, introducing existential quantification in $\overline{\mathrm{sp}}$ is undesirable.

Step 4: Define abstract disjunction.

Disjunction is applied in ForwardPropagate

$$\mu(L_k) := F \ \vee \ \mu(L_k)$$

Define abstract disjunction $\sqcup_D$ for this purpose, such that

$$F_1 \vee F_2 \ \Rightarrow \ F_1 \ \sqcup_D \ F_2 \quad \text{and} \quad F_1 \ \sqcup_D \ F_2 \in D$$

for $F_1, F_2 \in D$.
Unlike conjunction, exact disjunction is usually not represented in the domain D.

Step 5: Define abstract implication checking.

On each iteration of the inner loop of ForwardPropagate, validity of the implication

$$F \ \Rightarrow \ \mu(L_k)$$

is checked to determine whether $\mu(L_k)$ has changed. A proper selection of D ensures that this validity check is decidable.

Step 6: Define widening.

Defining an abstraction is not sufficient to guarantee termination in general. Thus, abstractions that do not guarantee termination are equipped with a widening operator $\bigtriangledown_D$.

A **widening operator** $\bigtriangledown_D$ is a binary function

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

such that

$$F_1 \vee F_2 \;\Rightarrow\; F_1 \bigtriangledown_D F_2$$

for $F_1, F_2 \in D$. It obeys the following property. Let $F_1, F_2, F_3, \ldots$ be an infinite sequence of elements $F_i \in D$ such that for each i,

$$F_i \;\Rightarrow\; F_{i+1} \; .$$

Define the sequence

$$G_1 = F_1 \quad \text{and} \quad G_{i+1} = G_i \bigtriangledown_D F_{i+1} \; .$$

For some i^* and for all $i \geq i^*$,

$$G_i \;\Leftrightarrow\; G_{i+1} \; .$$

That is, the sequence G_i converges even if the sequence F_i does not converge. A proper strategy of applying widening guarantees that the forward propagation procedure terminates.

$$
\begin{array}{l}
\texttt{let } \textsc{AbstractForwardPropagate}\ P\ F_{\mathsf{pre}}\ \mathcal{L} = \\
\quad S := \{L_0\}; \\
\quad \mu(L_0) := \nu_D(F_{\mathsf{pre}}); \\
\quad \mu(L) := \bot \texttt{ for } L \in \mathcal{L} \setminus \{L_0\}; \\
\quad \texttt{while } S \neq \emptyset \texttt{ do} \\
\qquad \texttt{let } L_j = \textsc{choose}\ S \texttt{ in} \\
\qquad S := S \setminus \{L_j\}; \\
\qquad \texttt{foreach } L_k \in \mathsf{succ}(L_j) \texttt{ do} \\
\qquad\quad \texttt{let } F = \overline{\mathsf{sp}}_D(\mu(L_j),\ S_j; \ldots; S_k) \texttt{ in} \\
\qquad\quad \texttt{if } F \not\Rightarrow \mu(L_k) \\
\qquad\quad \texttt{then if } \textsc{Widen}() \\
\qquad\qquad\qquad \texttt{then } \mu(L_k) := \mu(L_k) \bigtriangledown_D (\mu(L_k) \sqcup_D F); \\
\qquad\qquad\qquad \texttt{else } \mu(L_k) := \mu(L_k) \sqcup_D F; \\
\qquad\qquad\qquad S := S \cup \{L_k\}; \\
\qquad \texttt{done}; \\
\quad \texttt{done}; \\
\quad \mu
\end{array}
$$