

THE CALCULUS OF COMPUTATION:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

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

$$c_0 + c_1x_1 + \cdots + c_nx_n = 0$$

for program variables x_i and constants c_i

Other invariant generation algorithms in literature:

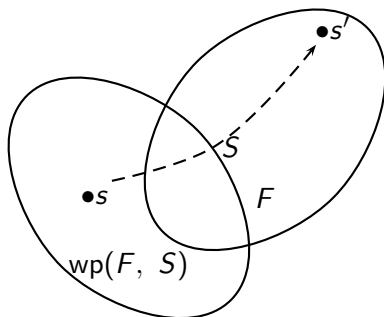
- ▶ linear inequalities

$$c_0 + c_1x_1 + \cdots + c_nx_n \leq 0$$

- ▶ polynomial equalities and inequalities

Background

Weakest Precondition



For FOL formula F and program statement S , the weakest precondition $\text{wp}(F, S)$ 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 $S_1; \dots; S_n$,
 $wp(F, S_1; \dots; S_n)$ must hold before executing the statements.

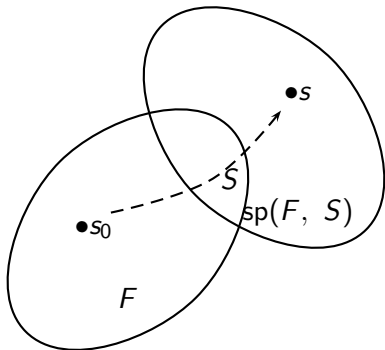
For assume and assignment statements

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

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

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

Strongest Postcondition



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

$$s \models sp(F, S)$$

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

$$s_0 \models F .$$

- ▶ On assume statements,

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

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:

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

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

$$\text{sp}(F, S_1; \dots; S_n) \Leftrightarrow \text{sp}(\text{sp}(F, S_1), S_2; \dots; S_n) .$$

Example: Compute

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

since $i^0 = i - k$.

Example: Compute

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

Verification Condition

VCs in terms of wp:

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

VCs in terms of sp:

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

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

$$\mu : \mathcal{L} \rightarrow \text{FOL}$$

(map from \mathcal{L} to first-order assertions).

It is inductive (inductive map) if for each basic path

$$L_i : @ \mu(L_i)$$

$$S_i;$$

$$\vdots$$

$$S_j;$$

$$L_j : @ \mu(L_j)$$

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

$$\{\mu(L_i)\} S_i; \dots; S_j \{\mu(L_j)\} \quad (\text{VC})$$

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 :

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

- ▶ 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)

$L_j : @ \mu(L_j)$

$S_j;$

\vdots

$S_k;$

$L_k : @ \mu(L_k)$

compute and set

$$\mu(L_k) \Leftarrow \mu(L_k) \vee \text{sp}(\mu(L_j), S_j; \dots; S_k)$$

If

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

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

Otherwise add L_k to S .

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

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

The algorithm

let FORWARDPROPAGATE $P F_{\text{pre}} \mathcal{L} =$

$S := \{L_0\};$

$\mu(L_0) := F_{\text{pre}};$

$\mu(L) := \perp$ for $L \in \mathcal{L} \setminus \{L_0\};$

while $S \neq \emptyset$ do

 let $L_j = \text{CHOOSE } S$ in

$S := S \setminus \{L_j\};$

 foreach $L_k \in \text{succ}(L_j)$ do $\left[\begin{array}{l} L_k \in \text{succ}(L_j) \text{ is a } \mathbf{successor} \text{ of } L_j \\ \text{if there is a basic path from } L_j \text{ to } L_k \end{array} \right]$

 let $F = \text{sp}(\mu(L_j), S_j; \dots; S_k)$ in

 if $F \not\Rightarrow \mu(L_k)$

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

$S := S \cup \{L_k\};$

 done;

done;

μ

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 : ?;

- ▶ Initially,

$$\begin{array}{l} \mu(L_0) \Leftrightarrow i = 0 \wedge n \geq 0 \\ \mu(L_1) \Leftrightarrow \perp \end{array}$$

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

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

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

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

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

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

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

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

Since the implication

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

is invalid,

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

at the end of the iteration.

- ▶ At the end of the next iteration,

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

- ▶ At the end of the k th iteration,

$$\mu(L_1) \Leftrightarrow \begin{array}{l} (i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \\ \vee \dots \vee (i = k \wedge n \geq k) \end{array}$$

It is never the case that the implication

$$i = k \wedge n \geq k$$

↓

$$(i = 0 \wedge n \geq 0) \vee (i = 1 \wedge n \geq 1) \vee \dots \vee (i = k - 1 \wedge n \geq k - 1)$$

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

$$c_0 + c_1x_1 + \cdots + c_nx_n = 0 ,$$

for constants c_0, c_1, \dots, c_n and variables x_1, \dots, 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

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

and in Karr's abstract domain by

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

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`:

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

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 $F \in D$,

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

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

$$\overline{\text{sp}}_{D_1}(F, \text{assume } c) \Leftrightarrow \nu_{D_1}(c) \wedge F .$$

► assignment statements:

More complex, for suppose that we use the standard definition

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

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

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

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

A **widening operator** ∇_D is a binary function

$$\nabla_D : D \times D \rightarrow D$$

such that

$$F_1 \vee F_2 \Rightarrow F_1 \nabla_D F_2$$

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

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

Define the sequence

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

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.

```

let ABSTRACTFORWARDPROPAGATE  $P$   $F_{\text{pre}}$   $\mathcal{L} =$ 
   $S := \{L_0\};$ 
   $\mu(L_0) := \nu_D(F_{\text{pre}});$ 
   $\mu(L) := \perp$  for  $L \in \mathcal{L} \setminus \{L_0\};$ 
  while  $S \neq \emptyset$  do
    let  $L_j = \text{CHOOSE } S$  in
       $S := S \setminus \{L_j\};$ 
      foreach  $L_k \in \text{succ}(L_j)$  do
        let  $F = \overline{\text{sp}}_D(\mu(L_j), S_j; \dots; S_k)$  in
          if  $F \not\approx \mu(L_k)$ 
            then if WIDEN()
              then  $\mu(L_k) := \mu(L_k) \nabla_D (\mu(L_k) \sqcup_D F);$ 
              else  $\mu(L_k) := \mu(L_k) \sqcup_D F;$ 
               $S := S \cup \{L_k\};$ 
            done;
      done;
  done;
 $\mu$ 

```