CS766: Analysis of concurrent programs (first half) 2022

Lecture 3: Weakest pre and Hoare logic

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2022-01-14

Topic 3.1

Weakest precondition

Executing program backwards.

The strongest post(sp) does not care about the error states.

Once we are done computing sp, we check that error states are reached.

Alternatively, we may think about executing backwards.

We start with good states and go backwards.

We find the states that are guaranteed to only reach to good states.

Exercise 3.1

How do we say that program is safe when we compute the states?

Weakest pre — dual of sp

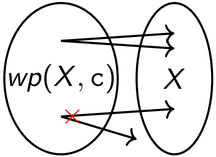
Now we define a an operator that executes the programs backwards!

Definition 3.1

Weakest pre operator $wp : \mathfrak{p}(\mathbb{Q}^{|V|}) \times \mathcal{P} \to \mathfrak{p}(\mathbb{Q}^{|V|})$ *is defined as follows.*

$$wp(X, \mathrm{c}) \triangleq \{v | \forall v' : (v', \mathtt{skip}) \in T^*((v, \mathrm{c})) \Rightarrow v' \in X\},$$

where $X \subseteq \mathbb{Q}^{|V|}$ *and* c *is a program.*



Example 3.1

Consider $V = [\mathrm{x}]$ *and* $X = \{[n] | 5 > n > 0\}$.

$wp(X, \mathrm{x} := \mathrm{x} + 1 [] \mathrm{x} := \mathrm{x} - 1) = \{[n] | 4 > n > 1\}$

Exercise 3.2

Why use of word "weakest"?

Logical weakest pre

We define symbolic wp that transforms formulas.

$$wp : \Sigma(V) \times \mathcal{P} \to \Sigma(V)$$

The equivalent definition of symbolic wp for data statements are

$$wp(F, \mathsf{c}) \triangleq (\forall V' : \rho(\mathsf{c}) \Rightarrow F[V'/V])$$

First rename then quantify

Example 3.2

- $wp(\top, \mathsf{c}) = \top$
- $wp(\bot, \mathsf{c}) =$ *those states that do not have any successor for* c

Notation altert: variable substitution

We write

$$F[V'/V]$$

we will replace variables V in the formula by expressions V'.

Example 3.3

- $(x + y)[x'/x] = x' + y$

We may also use a clearer alternative notation.

$$F\{x \mapsto exp1, y \mapsto exp2, ...\}$$

However, this notation is less common in literature.

Weakest pre for assignment

Theorem 3.1

$wp(F, \texttt{x := exp}) = F[\texttt{exp}/\texttt{x}]$

Proof.

Due to the definition of *wp*, $wp(F, \texttt{x := exp}) = (\forall V'.\ \underbrace{\texttt{x}' = \texttt{exp} \land frame(\texttt{x})}_{\text{equality for each variable}} \Rightarrow F[V'/V])$

We apply the equalities on the right hand side and remove prime variables.

$$= (\forall V'.\ \top \Rightarrow F[exp/\texttt{x}])$$

Since no primed variables left in the formula, we obtain.

$$= F[exp/\texttt{x}]$$

□

Recall, the similar simplification is not possible in the case of sp.(why?)

Exercise: wp for assignment

Exercise 3.3

1. $wp(\mathtt{i} = 2, \mathtt{i} \; := \; 1) =$
2. $wp(\mathtt{i} = 1, \mathtt{i} \; := \; 1) =$
3. $wp(\mathtt{i} \geq 0, \mathtt{i} \; := \; 1) =$
4. $wp((i \leq 3 \wedge r = (i-1)z + 1), \mathtt{i} \; := \; 1) =$
5. $wp((i < 3 \wedge r = iz + 1), \mathtt{r} \; := \; \mathtt{r} + \mathtt{z}) =$

Weakest pre for havoc

Theorem 3.2

$wp(F, \texttt{x} := \texttt{havoc()}) = \forall x.\ F$

Proof.

Due to the definition of wp, $wp(F, \texttt{x} := \texttt{havoc()}) = (\forall V'.\ \underbrace{frame(\texttt{x})}_{\text{equality for each variable except } \texttt{x}'} \Rightarrow F[V'/V])$

We apply the equalities on the rhs and left with only $\texttt{x}'$ from V'.

$$= (\forall V'.\ \top \Rightarrow F[\texttt{x}'/\texttt{x}])$$

After simplification, we obtain.

$$= \forall x'.\ F[\texttt{x}'/\texttt{x}].$$

Since the outside world does not care about the quantified variable name, no need for the renaming.

$$= \forall x.\ F$$

□

Commentary: Similar to the local variables in a programming language, the outside world does not care about the nume of quantified variable.

Example: wp for havoc

Example 3.4

1. $wp(\mathtt{i} = 2, \mathtt{i} := \mathtt{havoc()}) = \bot$
2. $wp(\mathtt{x} = 1, \mathtt{i} := \mathtt{havoc()}) = (\mathtt{x} = 1)$

Exercise 3.4

Compute the following

1. $wp(\mathtt{i} + \mathtt{x} \geq 0, \mathtt{i} := \mathtt{havoc()}) =$
2. $wp(\ (\mathtt{x} + \mathtt{i} \geq 0 \land \mathtt{x} - \mathtt{i} \leq 0) \lor (\mathtt{x} + \mathtt{i} \leq 2 \land \mathtt{x} - \mathtt{i} \geq 0)\ , \mathtt{i} := \mathit{havoc}()) =$

Commentary: Hint: for the forth problem draw the two dimensional diagram for the formula and look for values of x such that for all i the formula is true.

Weakest pre for assume

Theorem 3.3

$wp(F, \texttt{assume(G)}) = G \Rightarrow F$

Proof.

Due to the definition of wp, $wp(F, \texttt{assume(G)}) = (\forall V'.\ G \wedge \underbrace{frame(\emptyset)}_{\text{equality for each variable}} \Rightarrow F[V'/V])$

We apply the equalities on the rhs and left with no variables from V'.

$$= (\forall V'.\ G \Rightarrow F)$$

We can trivially remove V'.

$$= (G \Rightarrow F)$$

□

Example 3.5

- $wp(x < 0, \texttt{assume(x > 0)}) = (x > 0 \Rightarrow x < 0) = x \leq 0$

Weakest pre for assert

Theorem 3.4

$wp(F, \texttt{assert(G)}) = G \land F$ $\qquad$ *if* $F \not\equiv \top$

Proof.

Due to the definition of *wp*, $wp(F, \texttt{assert(G)}) = (\forall V'.\ (G \Rightarrow \underbrace{frame(\emptyset)}_{\text{equality for each variable}}) \Rightarrow F[V'/V])$

$= \forall V'.\ \neg(G \Rightarrow frame(\emptyset)) \lor F[V'/V]$ $\qquad$ ($\Rightarrow$ to $\lor$)

$= \forall V'.\ \neg(\neg G \lor frame(\emptyset)) \lor F[V'/V]$ $\qquad$ ($\Rightarrow$ to $\lor$)

$= \forall V'.\ (G \land \neg frame(\emptyset)) \lor F[V'/V]$ $\qquad$ (push $\neg$ inside)

$= \forall V'.\ (G \lor F[V'/V]) \land (\neg frame(\emptyset) \lor F[V'/V])$ $\qquad$ (DeMorgan's law)

...

Weakest pre for assert (contd.)

Proof(contd.)

$= \forall V'. (G \lor F[V'/V]) \land \forall V'. (\neg frame(\emptyset) \lor F[V'/V])$ ($\forall$ distributes over $\land$)

$= (\underbrace{G \lor \forall V'. F[V'/V]}_{\text{since } G \text{ has no V' variables}}) \land \forall V'. \underbrace{(frame(\emptyset) \Rightarrow F[V'/V])}_{\lor \text{ to } \Rightarrow}$

$= (G \lor \forall V. F) \land \forall V'. (\top \Rightarrow F)$

$= (G \lor \bot) \land F$ $\forall V. F$ is false(why?)

$= G \land F$ □

Exercise: weakest pre for assert and assume

Exercise 3.5

Compute the following

1. $wp(\mathtt{i} = 2, \mathtt{assume}(\mathtt{i} = 3)) =$
2. $wp(\mathtt{i} = 2, \mathtt{assert}(\mathtt{i} = 3)) =$
3. $wp(\mathtt{i} = 2, \mathtt{assume}(\mathtt{i} = 2)) =$
4. *Are there F and G such that* $wp(F, \mathtt{assume}(\mathtt{G})) = \bot$*?*
5. *Are there F and G such that* $wp(F, \mathtt{assert}(\mathtt{G})) = \top$*?*

Commentary: wp over assume returns large set of states and over asserts returns small set. sp behaves in the opposite way.

Logical wp for control statements (other than while)

The definition of symbolic wp for control statements are

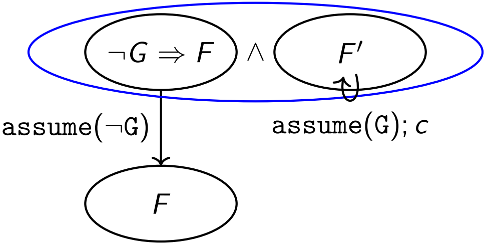
$$wp(F, c_1; c_2) = wp(wp(F, c_2), c_1)$$
$$wp(F, c_1[]c_2) = wp(F, c_1) \wedge wp(F, c_2)$$
$$wp(F, \texttt{if}(\texttt{F}_1)\ \texttt{c}_1\ \texttt{else}\ \texttt{c}_2) = wp(F, \texttt{assume}(\texttt{F}_1); c_1) \wedge wp(F, \texttt{assume}(\neg\texttt{F}_1); c_2)$$

Example 3.6

$$wp(x = 0, \texttt{if}(\texttt{y} > 0)\ \texttt{x} := \texttt{x} + 1\ \texttt{else}\ \texttt{x} := \texttt{x} - 1) =$$
$$wp(x = 0, \texttt{assume}(\texttt{y} > 0); \texttt{x} := \texttt{x} + 1) \wedge wp(x = 0, \texttt{assume}(\texttt{y} \leq 0); \texttt{x} := \texttt{x} - 1)$$
$$= wp(x = -1, \texttt{assume}(\texttt{y} > 0)) \wedge wp(x = 1, \texttt{assume}(\texttt{y} \leq 0))$$
$$= (x = -1 \vee y \leq 0) \quad \wedge \quad (x = 1 \vee y > 0)$$

Logical weakest pre for control statements for loop

$$wp(F, \texttt{while}(G)c) = gfp_{F'}((G \vee F) \wedge wp(F', \texttt{assume(G)}; c))$$



Exercise 3.6

Why gfp not lfp?

Exercise: wp over loops

Exercise 3.7 (Give intuitive answers!)

Compute the following

1. $wp(y < 2, \texttt{while}(\mathrm{y} < 10)\ \mathrm{y} := \mathrm{y} + 1) =$

2. $wp(y \geq 10, \texttt{while}(\mathrm{y} < 10)\ \mathrm{y} := \mathrm{y} + 1) =$

3. $wp(y = 11, \texttt{while}(\mathrm{y} < 10)\ \mathrm{y} := \mathrm{y} + 1) =$

4. $wp(y = 10, \texttt{while}(\mathrm{y} < 10)\ \mathrm{y} := \mathrm{y} + 1) =$

5. $wp(y = 0, \texttt{while}(\top)\ \mathrm{y} := \mathrm{y} + 1) =$

Commentary: Again, we have not seen an algorithm to compute gfp. However, we should be able to answer the above using our understanding of programming.

Handling non-terminating executions

I have cheated a bit. Our wp allows states that may lead to non-terminating executions and never reach to the final state.

Our definition of wp is called (check wikipedia)

weakest liberal precondition(wlp),

since our wp includes non-terminating executions.

There is a stricter definition of wp for loops that insists to include states that lead to only terminating executions.

We ignore the exact encoding for the stricter wp for most of this course.

Safety verification via wp

Lemma 3.1

For a program c, if $err = 0 \Rightarrow wp(err = 0, c)$ *is valid then c is safe.*

Exercise 3.8

Prove the above lemma.

Exercise 3.9

Is wp any better than the sp based verification?

Topic 3.2

Hoare logic

Hoare logic - our first method of verification

- Computing a super set of the reachable states(lfp) that does not intersect with error states should be suffice for our goal
- Since we do not know how to compute lfp, we will first see a method of writing pen-paper proofs of program safety
- Proof method has following steps
  - guess a super set of reachable states
  - show guess is correct
  - show the guess does not intersect with error states
- Invented by Tony Hoare
  - sometimes called axiomatic semantics

Hoare triple

Definition 3.2

$$\{P\}\mathtt{c}\{Q\}$$

- $P : \Sigma(V)$, *usually called precondition*
- $\mathtt{c} : \mathcal{P}$
- $Q : \Sigma(V)$, *usually called postcondition*

Definition 3.3

$\{P\}\mathtt{c}\{Q\}$ *is valid if all the executions of* $\mathtt{c}$ *that start from* P *end in* Q*, i.e.,*

$$\forall v, v'.\ v \models P \wedge ((v, \mathtt{c}), (v', \mathtt{skip})) \in T^* \Rightarrow v' \models Q.$$

P and Q are not tightly related by weakest pre or strongest post.

Hoare proof obligation/goal

The safety verification problem is slightly differently stated in Hoare logic.

We remove assert statement from the language and no *err* variable.

Here, a verification problem is proving validity of a Hoare triple.

Example 3.7

Program		*Hoare triple*
$\mathtt{assume}(\top)$ $\mathtt{r} := 1;$ $\mathtt{i} := 1;$ $\mathtt{while}(\mathtt{i} < 3)$ $\{$ $\quad \mathtt{r} := \mathtt{r} + \mathtt{z};$ $\quad \mathtt{i} := \mathtt{i} + 1$ $\}$ $\mathtt{assert}(\mathtt{r} = 2\mathtt{z} + 1)$	$\rightarrow$	$\{\top\}$ $\mathtt{r} := 1;$ $\mathtt{i} := 1;$ $\mathtt{while}(\mathtt{i} < 3)$ $\{$ $\quad \mathtt{r} := \mathtt{r} + \mathtt{z};$ $\quad \mathtt{i} := \mathtt{i} + 1$ $\}$ $\{r = 2z + 1\}$

Notation alert: deduction rules

$$\textsc{RuleName}\ \frac{\text{Stuff-already-there}}{\text{Stuff-to-be-added}}\ \text{Conditions to be met}$$

Hoare Proof System – data statements

$$[\textsc{SkipRule}]\,\frac{}{\{P\}\texttt{skip}\{P\}}$$

$$[\textsc{AssignRule}]\,\frac{}{\{P[exp/x]\}\texttt{x := exp}\{P\}}$$

$$[\textsc{HavocRule}]\,\frac{}{\{\forall x.P\}\texttt{x := havoc()}\{P\}}$$

$$[\textsc{AssumeRule}]\,\frac{}{\{P\}\texttt{assume(F)}\{\texttt{F} \land P\}}$$

We may freely choose any of sp and wp for pre/post pairs for data statements.

Example: Hoare proof system – data statements

Example 3.8

We may have the following derivations due to the rules.

$$\frac{}{\{\mathtt{i} = 0\}\mathtt{i} \; := \; \mathtt{i} + 1\{\mathtt{i} = 1\}}$$

$$\frac{}{\{\mathtt{i} = 0\}\mathtt{assume}(\mathtt{x} > 0)\{\mathtt{i} = 0 \land \mathtt{x} > 0\}}$$

$$\frac{}{\{\mathtt{r} = 1\}\mathtt{i} := 1\{i \leq 3 \land r = (i-1)z + 1\}}$$

Hoare Proof System – sequential composition

$$[\textsc{SeqRule}]\ \frac{\{P\}\mathsf{c}_1\{Q\} \quad \{Q\}\mathsf{c}_2\{R\}}{\{P\}\mathsf{c}_1;\mathsf{c}_2\{R\}}$$

Example 3.9

$$\frac{\overline{\{\top\}\mathtt{r} := 1\{\mathtt{r} = 1\}} \qquad \overline{\{\mathtt{r} = 1\}\mathtt{i} := 1\{i \leq 3 \wedge r = (i-1)z + 1\}}}{\{\top\}\mathtt{r} := 1; \mathtt{i} := 1; \{i \leq 3 \wedge r = (i-1)z + 1\}}$$

Hoare proof system – nondeterminism

$$[\textsc{NondetRule}]\ \frac{\{P\}\mathtt{c}_1\{Q\} \quad \{P\}\mathtt{c}_2\{Q\}}{\{P\}\mathtt{c}_1[]\mathtt{c}_2\{Q\}}$$

Example 3.10

$$\frac{\overline{\{\top\}\mathtt{r} := 1\{\mathtt{r} \geq 1\}} \qquad \overline{\{\top\}\mathtt{r} := 2\{\mathtt{r} \geq 1\}}}{\{\top\}\mathtt{r} := 1[]\mathtt{r} := 2\{\mathtt{r} \geq 1\}}$$

Example 3.11

$$\frac{\overline{\{\top\}\mathtt{r} := 1\{\mathtt{r} \geq 1\}} \qquad \overline{\{\top\}\mathtt{r} := 2\{\mathtt{r} \geq 2\}}}{\{\top\}\mathtt{r} := 1[]\mathtt{r} := 2\{??\}} ✗$$

Both pre and post must match to apply the rule

Hoare proof system – branching

$$[\textsc{IfRule}]\frac{\{\mathtt{F} \land P\}\mathtt{c}_1\{Q\} \quad \{\neg\mathtt{F} \land P\}\mathtt{c}_2\{Q\}}{\{P\}\mathtt{if(F)}\ \mathtt{c}_1\ \mathtt{else}\ \mathtt{c}_2\{Q\}}$$

IfRule is the combination of NondetRule and SeqRule.

Hoare proof system – semantic weakening

$$[\textsc{ConsequenceRule}]\frac{P_1 \Rightarrow P_2 \quad \{P_2\}\mathtt{c}\{Q_2\} \quad Q_2 \Rightarrow Q_1}{\{P_1\}\mathtt{c}\{Q_1\}}$$

We can strengthen pre and weaken post, without loosing soundness.

The rule is useful for matching pre/posts for compositions.

Example 3.12

$$\cfrac{\cfrac{}{\{\top\}\mathtt{r} := 1\{\mathtt{r} \geq 1\}} \quad \cfrac{\cfrac{}{\{\top\}\mathtt{r} := 2\{\mathtt{r} \geq 2\}} \quad \mathtt{r} \geq 2 \Rightarrow \mathtt{r} \geq 1}{\{\top\}\mathtt{r} := 2\{\mathtt{r} \geq 1\}}}{\{\top\}\mathtt{r} := 1[]\mathtt{r} := 2\{\mathtt{r} \geq 1\}}$$

Commentary: We could not have non-deterministic composed the Hoare triples if post conditions did not match.

Hoare proof system – loop statement

$$[\textsc{WhileRule}]\,\frac{\{\mathtt{I} \wedge \mathtt{F}\}\mathtt{c}\{\mathtt{I}\}}{\{\mathtt{I}\}\mathtt{while(F)\ c}\{\neg F \wedge \mathtt{I}\}}$$

Usually, the loop is sitting between a pre and post, i.e.,

$$\{pre\}\mathtt{while(F)\ c}\{post\}.$$

We guess I such that we can prove the antecedent of WHILERULE and after weakening we obtain the pre and post. The proof will flow as follows.

$$\frac{pre \Rightarrow \mathtt{I} \qquad \dfrac{\{\mathtt{I} \wedge \mathtt{F}\}\mathtt{c}\{\mathtt{I}\}}{\{\mathtt{I}\}\mathtt{while(F)\ c}\{\neg F \wedge \mathtt{I}\}} \qquad \neg F \wedge \mathtt{I} \Rightarrow post}{\{pre\}\mathtt{while(F)\ c}\{post\}}$$

Non-mechanical step: invent I such that the above works. I is called loop-invariant.

Example: Hoare proof

Example 3.13

Consider loop invariant: $\mathtt{I} = (i \leq 3 \land r = (i-1)z + 1)$

$\{\top\}$
$\mathtt{r} := 1;$
$\{\mathtt{r} = 1\}$
$\mathtt{i} := 1;$
$\{\mathtt{I}\}$
$\mathtt{while}(\mathtt{i} < 3)$
$\{$
$\quad \{\mathtt{I} \land \mathtt{i} < 3\}$
$\quad \mathtt{r} := \mathtt{r} + \mathtt{z}$
$\quad \{i < 3 \land r = iz + 1\}$
$\quad \mathtt{i} := \mathtt{i} + 1$
$\}$
$\{r = 2z + 1\}$

$$\dfrac{\overline{\{\top\}\mathtt{r} := 1\{\mathtt{r} = 1\}} \qquad \overline{\{\mathtt{r} = 1\}\mathtt{i} := 1\{\mathtt{I}\}}}{\{\top\}\mathtt{r} := 1; \mathtt{i} := 1; \{\mathtt{I}\}}$$

$$\dfrac{\dfrac{\overline{\{i < 3 \land \mathtt{I}\}\mathtt{r} := \mathtt{r} + \mathtt{z}\{i < 3 \land r = iz + 1\}} \qquad \overline{\{i < 3 \land r = iz + 1\}\mathtt{i} := \mathtt{i} + 1\{\mathtt{I}\}}}{\{i < 3 \land \mathtt{I}\}\mathtt{r} := \mathtt{r} + \mathtt{z}; \mathtt{i} := \mathtt{i} + 1\{\mathtt{I}\}}}{\{\mathtt{I}\}\mathtt{while}(..)..\{\mathtt{I} \land \mathtt{i} \geq 3\}}$$

$$\dfrac{\dfrac{\{\top\}\mathtt{r} := 1; \mathtt{i} := 1; \{\mathtt{I}\} \qquad \{\mathtt{I}\}\mathtt{while}(..)..\{\mathtt{I} \land \mathtt{i} \geq 3\}}{\{\top\}\mathtt{r} := 1; ..; \mathtt{while}(..)..\{\mathtt{I} \land \mathtt{i} \geq 3\}} \qquad \mathtt{I} \land \mathtt{i} \geq 3 \Rightarrow r = 2z + 1}{\{\top\}\underbrace{\mathtt{r} := 1; ..; \mathtt{while}(..)..}_{\text{full program}}\{r = 2z + 1\}}$$

Commentary: Rule names are not explicitly written, which should be clear by the context.

Topic 3.3

Problems

Exercise: sp vs wp

Exercise 3.10

Prove that $sp(wp(F, \mathrm{c}), \mathrm{c}) \subseteq F \subseteq wp(sp(F, \mathrm{c}), \mathrm{c})$

Exercise: post using z3

Exercise 3.11

Write a C++ program that reads a SMT2 formula from command line and performs quantifier elimination using Z3 for the variables that do not end with '

Exercise: Hoare triple

Exercise 3.12

Prove the following Hoare triple is valid

```
{true}
assume( n > 1);
i = n;
x = 0;
while(i > 0) {
  x = x + i;
  i = i - 1;
}
{ 2x = n*(n+1) }
```

Exercise: translating to Hoare triple

Exercise 3.13

Consider the following program. Use Hoare logic to prove the program correct.

```
int main( int n ) {
  assume( p == 0 );
  while( n > 0 ) {
    assert( p != 0 );
    if( n== 0 ) {
       p = 0;
    }
    n--;
  }
}
```

Note that assert is not at the every end of the program.

End of Lecture 3