Weakest precondition II

Farzaneh Derakhshan
based on material by Stefan Muller and Jim Sasaki

CS 536: Science of Programming, Fall 2023
Lecture 12

In this lecture, we answer the question of how we can construct the weakest preconditions. We only consider a loop-free system for now (no while constructs in the program), but the program may still result in runtime errors. We will first describe how to construct the weakest liberal precondition (WLP) for programs, as it turns out to be an easier task. Then, we will continue with building the weakest precondition(WP). In loop-free programs, WP is defined quite similarly to WLP, but WP also ensures that the program is error-free. We build WP using the constructed WLP and an algorithm that generates a condition of the program stating it never results in an error.

Note. Based on what we saw in the previous lectures, for *error-free* and *loop-free* programs WLP and WP are equal, and thus the algorithms in Sections 1 and 2 return the same result.

1 Algorithm for weakest liberal precondition

We define a recursive algorithm that takes the statement S and postcondition Q and constructs the predicate $wlp(S, Q)$.

$$
\begin{array}{lll}
(1) & wlp(\mathsf{skip}, Q) & ::= \quad Q \\
(2) & wlp(x := e, Q) & ::= \quad [e/x]Q \\
(3) & wlp(S_1; S_2, Q) & ::= \quad wlp(S_1, wlp(S_2, Q)) \\
(4) & wlp(\mathsf{if}\ e\ \mathsf{then}\ S_2\ \mathsf{else}\ S_2\ \mathsf{fi}, Q) & ::= \quad (e \to wlp(S_1, Q)) \wedge (\neg e \to wlp(S_2, Q))
\end{array}
$$

Example 1.

$$
\begin{array}{lll}
wlp(x := x + 1, x \geq 0) & ::= \quad [x + 1/x](x \geq 0) & \textit{by line (2) in the algorithm} \\
& \quad = x + 1 \geq 0 & \textit{by substitution}
\end{array}
$$

Example 2.

$$
\begin{array}{lll}
wlp(y := y + x; x := x + 1, x \geq 0) & ::= \quad wlp(y := y + x, wlp(x := x + 1), x \geq 0) & \textit{by line (3)} \\
& \quad = wlp(y := y + x, x + 1 \geq 0) & \textit{by Example 1} \\
& \quad = [y + x/y]x + 1 \geq 0 & \textit{by line (2)} \\
& \quad = x + 1 \geq 0 & \textit{by substitution}
\end{array}
$$

Example 3.

$$
\begin{array}{lll}
wlp(y := y + x; x := x + 1, x \geq y) & ::= \quad wlp(y := y + x, wlp(x := x + 1), x \geq y) & \textit{by line (3)} \\
& \quad = wlp(y := y + x, [x + 1/x]x \geq y) & \textit{by line (2)} \\
& \quad = wlp(y := y + x, x + 1 \geq y) & \textit{by substitution} \\
& \quad = [y + x/y]x + 1 \geq y & \textit{by line (2)} \\
& \quad = x + 1 \geq y + x & \textit{by substitution}
\end{array}
$$

We could simplify the condition $x + 1 \geq y + x$ further to get $y \leq 1$.

Exercise 1. Calculate $wlp(x := x + 1; y := y + x, x \geq y)$.

Example 4. Put S_1 to be the program if $x \geq y$ then $m := x$ else skip fi.

$$
\begin{array}{lll}
wlp(S_1, m = \max(x,y)) ::= & & \\
\quad x \geq y \to wlp(m := x, m = \max(x,y)) \wedge \neg(x \geq y) \to wlp(\mathsf{skip}, m = \max(x,y)) & & \textit{by line (4)} \\
\quad = x \geq y \to [x/m]m = \max(x,y) \wedge \neg(x \geq y) \to wlp(\mathsf{skip}, m = \max(x,y)) & & \textit{by line (2)} \\
\quad = x \geq y \to [x/m]m = \max(x,y) \wedge \neg(x \geq y) \to m = \max(x,y) & & \textit{by line (1)} \\
\quad = x \geq y \to x = \max(x,y) \wedge \neg(x \geq y) \to m = \max(x,y) & & \textit{by substitution}
\end{array}
$$

We can simpilfy $x \geq y \to x = \max(x,y) \wedge \neg(x \geq y) \to m = \max(x,y)$ further. We know that $x \geq y \to x = \max(x,y)$ is a tautology, hence equivalent to T. We can rewrite the condition as $T \wedge \neg(x \geq y) \to m = \max(x,y)$, which is equivalent to $\neg(x \geq y) \to m = \max(x,y)$, and thus $x < y \to m = \max(x,y)$.

Exericse 2. Calculate $wlp(\text{if } y \geq 0 \text{ then } x := y \text{ else skip fi}, x \geq 0)$.

2 Algorithm for weakest precondition

To build WLP from WP (for loop-free programs), we also need to rule out errors. We first define a predicate that ensures the expression/statement never results in an error. Then, we augment it with the WLP, as explained above, to get WP.

2.1 Error-free predicate for expressions

We define predicate $D(e)$ such that if for any state σ we have $\sigma \vDash D(e)$, then $\sigma(e) \neq \perp$. In other words, if a state σ satisfies $D(e)$, then $\sigma(e)$ will not result in an error. We will define the predicate recursively for each expression in the grammar:

(1)	$D(c)$	$=$	T	$c \in \{\mathbf{true}, \mathbf{false}, \overline{n}, x\}$
(2)	$D(a[e])$	$=$	$D(e) \wedge 0 \leq e < size(a)$	
(3)	$D(e_1/e_2)$	$=$	$D(e_1) \wedge D(e_2) \wedge e_2 \neq 0$	
(4)	$D(sqrt(e))$	$=$	$D(e) \wedge e \geq 0$	
(5)	$D(e_1 \mathit{op} e_2)$	$=$	$D(e_1) \wedge D(e_2)$	$op \in \{+, -, *, \wedge, \vee, \leq, <, =, \geq, >\}$
(6)	$D(\neg e)$	$=$	$D(e)$	
(7)	$D(\mathit{if}\ e\ \mathit{then}\ e_1\ \mathit{else}\ e_2)$	$=$	$D(e) \wedge (e \to D(e_1)) \wedge (\neg e \to D(e_2))$	

Lines 2-4 in the algorithm above describe those expressions that are susceptible to errors; they need extra conditions on the state to ensure that the error does not occur. We only consider these forms of errors, but the algorithm can be extended to other errors depending on the datatypes and operations being used. Lines 5 and 6 say that various operations ($+$, $-$, $\leq$, $=$, $\wedge$, $\vee$, etc.) don't cause an error as long as the subexpressions don't cause one. Line 7, says that for a conditional expression, we only need safety of the branch we execute.

Example 5.

$$
\begin{array}{lll}
D(a[a[\overline{n}]]) := & D(a[\overline{n}]) \wedge 0 \leq a[\overline{n}] \leq size(a) & \textit{by line 2} \\
& D(\overline{n}) \wedge 0 \leq \overline{n} \leq size(a) \wedge 0 \leq a[\overline{n}] \leq size(a) & \textit{by line 2} \\
& T \wedge 0 \leq \overline{n} \leq size(a) \wedge 0 \leq a[\overline{n}] \leq size(a) & \textit{by line 1}
\end{array}
$$

We can simplify the result to the equivalent formula $0 \leq \overline{n} \leq size(a) \wedge 0 \leq a[\overline{n}] \leq size(a)$.

Example 6.

$$
\begin{array}{lll}
D(sqrt(x*y)/(\overline{2}*x)) := & D(sqrt(x*y)) \wedge D(\overline{2}*x) \wedge (\overline{2}*x) \neq 0 & \textit{by line 3} \\
& D(x*y) \wedge (x*y) \geq 0 \wedge D(\overline{2}*x) \wedge (\overline{2}*x) \neq 0 & \textit{by line 4} \\
& D(x) \wedge D(y) \wedge (x*y) \geq 0 \wedge D(\overline{2}*x) \wedge (\overline{2}*x) \neq 0 & \textit{by line 5} \\
& D(x) \wedge D(y) \wedge (x*y) \geq 0 \wedge D(\overline{2}) \wedge D(x) \wedge (\overline{2}*x) \neq 0 & \textit{by line 5} \\
& T \wedge T \wedge (x*y) \geq 0 \wedge T \wedge T \wedge (\overline{2}*x) \neq 0 & \textit{by line 1}
\end{array}
$$

We can simplify the result to the equivalent formula $\wedge(x*y) \geq 0 \wedge (\overline{2}*x) \neq 0$.

Example 7. $D(\mathit{if}\ 0 \le k < \mathit{size}(b)\ \mathit{then}\ b[k]\ \mathit{else}\ 0)$

$$
\begin{array}{ll}
D(\mathit{if}\ \overline{0} \le x < \mathit{size}(b)\ \mathit{then}\ b[x]\ \mathit{else}\ \overline{0}) := & \\
\quad D(\overline{0} \le x < \mathit{size}(b)) \wedge ((\overline{0} \le x < \mathit{size}(b)) \to D(b[x])) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to D(\overline{0})) & \textit{by line 7} \\
\quad D(\overline{0}) \wedge D(x) \wedge D(\mathit{size}(b)) \wedge ((\overline{0} \le x < \mathit{size}(b)) \to D(b[x])) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to D(\overline{0})) & \textit{by line 5} \\
\quad T \wedge T \wedge T \wedge ((\overline{0} \le x < \mathit{size}(b)) \to D(b[x])) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to D(\overline{0})) & \textit{by line 1} \\
\quad T \wedge T \wedge T \wedge ((\overline{0} \le x < \mathit{size}(b)) \to (D(x) \wedge (\overline{0} \le x < \mathit{size}(b))) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to D(\overline{0})) & \textit{by line 1} \\
\quad T \wedge T \wedge T \wedge ((\overline{0} \le x < \mathit{size}(b)) \to (T \wedge (\overline{0} \le x < \mathit{size}(b))) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to D(\overline{0})) & \textit{by line 1} \\
\quad T \wedge T \wedge T \wedge ((\overline{0} \le x < \mathit{size}(b)) \to (T \wedge (\overline{0} \le x < \mathit{size}(b))) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to T) & \textit{by line 1}
\end{array}
$$

We can simplify the result to the equivalent formula $(\overline{0} \le x < \mathit{size}(b)) \to (\overline{0} \le x < \mathit{size}(b)) \wedge (\neg(\overline{0} \le x < \mathit{size}(b)) \to T)$, and even further to T. When $D(e) = T$, it means that the expression never results in a failure. **Exercise 2.** Why?

2.2 Error-free predicate for statements

Similar to above, we can construct a predicate $D(S)$ that ensures an statement doesn't result in a runtime error. We have if $\sigma \vDash D(S)$ then $\langle S, \sigma\rangle \not\to \langle \mathsf{skip}, \bot\rangle$, i.e., running the configuration $\langle S, \sigma\rangle$ won't result in a run-time error.

$$
\begin{array}{lllll}
(1) & D(\mathsf{skip}) & = & T \\
(2) & D(x := e) & = & D(e) \\
(3) & D(S_1; S_2) & = & D(S_1) \wedge \mathit{wlp}(S_1, D(S_2)) \\
(4) & D(\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}) & = & D(e) \wedge (e_1 \to D(S_1)) \wedge (e_2 \to D(S_2))
\end{array}
$$

Example 8.

$$
\begin{array}{lll}
D(\mathsf{if}\ x \ge y\ \mathsf{then}\ m := x\ \mathsf{else\ skip\ fi}) := & D(x \ge y) \wedge (x \ge y \to D(m := x)) \wedge (\neg(x \ge y) \to D(\mathsf{skip})) & \textit{by line 4} \\
& = D(x \ge y) \wedge (x \ge y \to D(x)) \wedge (\neg(x \ge y) \to D(\mathsf{skip})) & \textit{by line 2} \\
& = D(x \ge y) \wedge (x \ge y \to D(x)) \wedge (\neg(x \ge y) \to T) & \textit{by line 1} \\
& = T \wedge (x \ge y \to T) \wedge (\neg(x \ge y) \to T) & \textit{Section 2.1}
\end{array}
$$

The result is a tautology and thus logically equivalent to T. When $D(S) = T$, it means that the statement S never results in a failure, no matter what the starting state is (after all, every state σ satisfies T).

You may wonder wht we don't define line (3) for sequential composition as

$$(3') \quad D(S_1; S_2) \quad = \quad D(S_1) \wedge D(S_2)$$

Let's look at an example to understand the reason.
Example 9. We want to calculate $D(y := 1; x := x/y)$. If we use the definition $(3')$, we get

$$D(y := 1; x := x/y) = D(y := 1) \wedge D(x := x/y) = y \neq 0.$$

However, this is not acurrate enough: the condition $y \neq 0$ says that the statement $y{:=}1; x{:=}x/y$ is error-free when the initial value of y is not equal to 0, but, in fact, the statement is error-free no matter what the initial value of y is. The reason is y is first assigned to 1 and only after that we perform the division.

Now, let's see what the definition given in line (3) gives us:

$$D(y{:=}1; x{:=}x/y) = D(y{:=}1) \wedge \mathit{wlp}(y{:=}1, D(x{:=}x/y)) = \mathit{wlp}(y{:=}1, D(x{:=}x/y)) = \mathit{wlp}(y{:=}1, y \neq 0) = [1/y](y \neq 0) = 1 \neq 0 = T$$

This construction correctly returns the condition T, meaning the program is error-free without any extra condition on the initial state.

2.3 Weakest precondition

Now that we defined the error-free predicates, it's easy to extend WLP to WP for loop-free programs. We can simply write

$$\mathit{wp}(S, Q) := D(S) \wedge \mathit{wlp}(S, Q)$$

$D(S)$ tells us that running S won't cause an error, and $wlp(s, q)$ provides the weakest precondition on the states such that running S establishes the postcondition Q if no error occurs.
When a statement is error-free on all states, WP and WLP are the same. In this case, we have $D(S) = T$, and

$$wp(S, Q) = wlp(S, Q) \wedge D(S) = wlp(S, Q) \wedge T = wlp(S, Q).$$

Exercise 3. Build $wlp(x := y; z := v/x, z > x + 2)$ using the Algorithm in Section 2. Next use error-free predicates in Sections 2.2 and 2.1 to calculate $D(x := y; z := v/x, z > x + 2)$. Observe that $wp(S, Q) = wlp(x := y; z := v/x, z > x + 2) \wedge D(x := y; z := v/x, z > x + 2)$