Weakest Preconditions, v1.1

Part 2: Calculating wp, wlp; Domain Predicates

CS 536: Science of Programming, Fall 2022

v1.1 misc small changes

A. Why

- Weakest liberal preconditions *(wlp)* and weakest preconditions *(wp)* are the most general requirements that a program must meet to be correct under partial and total correctness.

B. Objectives

At the end of today you should understand

- How to calculate the wlp of loop-free programs.
- How to add error domain predicates to the *wlp* of a loop-free program to obtain its *wp*.

C. Calculating wlp for Loop-Free Programs

- Say a program is loop-free. If it is also error-free, then its *wp* and *wlp* are identical. Otherwise we will need to add error-avoiding information to the *wlp* to calculate the *wp*. Either way, calculating the *wlp* is the first step.
- The following algorithm takes S and q and calculates a predicate for $wlp(S, q)$..
- The calculation is syntactic, which is why it's described using $wlp(S, q) \equiv \ldots$ instead of $wp(S, q) \Leftrightarrow \ldots$.
  - $wlp(\textbf{skip}, q) \equiv q$
  - $wlp(v := e,\ Q(v)) \equiv Q(e)$ where Q is a predicate function over one variable .
    - The operation that takes us from $Q(v)$ to $Q(e)$ is called ***syntactic substitution***; we'll look at it in more detail in the next class, but for the examples here and in earlier classes, we've been using the simplest case, where we inspect the definition of Q and replacing each occurrence of the variable v with the expression e.
  - $wlp(S_1; S_2, q) \equiv wlp(S_1, wlp(S_2, q))$
    - The $wlp(S_2, q)$ guarantees that we'll run S_2 in a state that gets us to q. To guarantee that S_1 gets us to one of those states, we use the outer $wlp(S_1, \ldots)$.
  - $wlp(\textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}, q) \equiv (B \rightarrow w_1) \wedge (\neg B \rightarrow w_2)$ where $w_1 \equiv wlp(S_1, q)$ and $w_2 \equiv wlp(S_2, q)$.
    - This is $\Leftrightarrow (B \wedge w_1) \vee (\neg B \wedge w_2)$, so it's also acceptable as a result of this *wlp* calculation.
  - $wlp(\textbf{if}\ B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2\ \textbf{fi}, q) \equiv (B_1 \rightarrow w_1) \wedge (B_2 \rightarrow w_2)$ where $w_1 \equiv wlp(S_1, q)$ and $w_2 \equiv wlp(S_2, q)$.
    - For the nondeterministic ***if***, you ***must*** use $(B_1 \rightarrow w_1) \wedge (B_2 \rightarrow w_2)$, not $(B_1 \wedge w_1) \vee (B_2 \wedge w_2)$, because they're not equivalent *(*unlike the deterministic ***if*** statement*)*.

- When B_1 and B_2 are both true, either S_1 or S_2 can run, so we need $B_1 \wedge B_2 \rightarrow w_1 \wedge w_2$, and this is implied by $(B_1 \rightarrow w_1) \wedge (B_2 \rightarrow w_2)$.
- Using $(B_1 \wedge w_1) \vee (B_2 \wedge w_2)$ fails because it allows for the possibility that B_1 and B_2 are both true but only one of w_1 and w_2 is true. This isn't a problem when $B_2 \Leftrightarrow \neg B_1$, which is why we can use $(B \wedge w_1) \vee (\neg B \wedge w_2)$ with deterministic ***if*** statements.

D. Some Examples of Calculating wp/wlp:

- The programs in these examples never end in "state" $\perp$, so the *wp* and *wlp* are equivalent.
- ***Example 2***: $wlp(x := x+1, x \geq 0) \equiv x+1 \geq 0$

- ***Example 3***: $wlp(y := y+x;\ x := x+1, x \geq 0)$
  $\equiv wlp(y := y+x, wlp(x := x+1, x \geq 0))$
  $\equiv wlp(y := y+x, x+1 \geq 0) \equiv x+1 \geq 0$

- ***Example 4***: $wlp(y := y+x;\ x := x+1, x \geq y)$
  $\equiv wlp(y := y+x, wlp(x := x+1, x \geq y))$
  $\equiv wlp(y := y+x, x+1 \geq y)$
  $\equiv x+1 \geq y+x$
  (If we asked to calculate and logically simplify, not just calculate, the *wlp*, we'd continue)
  $\Leftrightarrow y \leq 1$.

- ***Example 5***: Swap the two assignments in Example 4:
  $wlp(x := x+1;\ y := y+x, x \geq y)$
  $\equiv wlp(x := x+1, wlp(y := y+x, x \geq y))$
  $\equiv wlp(x := x+1, x \geq y+x))$
  $\equiv x+1 \geq y+x+1$ [$\Leftrightarrow y \leq 0$ if you want to logically simplify]

- ***Example 6***: $wlp(\textbf{if}\ y \geq 0\ \textbf{then}\ x := y\ \textbf{fi}, x \geq 0)$
  $\equiv wlp(\textbf{if}\ y \geq 0\ \textbf{then}\ x := y\ \textbf{else skip fi}, x \geq 0)$
  $\equiv (y \geq 0 \rightarrow wlp(x := y, x \geq 0)) \wedge (y < 0 \rightarrow wlp(\textbf{skip}, x \geq 0))$
  $\equiv (y \geq 0 \rightarrow y \geq 0) \wedge (y < 0 \rightarrow x \geq 0)$ or $(y \geq 0 \wedge y \geq 0) \vee (y < 0 \wedge x \geq 0)$
  (If we were asked to calculate and logically simplify the *wlp*, we'd continue):
  $\Leftrightarrow y \geq 0 \vee (y < 0 \wedge x \geq 0)$
  $\Leftrightarrow (y \geq 0 \vee y < 0) \wedge (y \geq 0 \vee x \geq 0)$
  $\Leftrightarrow (y \geq 0 \vee x \geq 0)$ or $(y < 0 \rightarrow x \geq 0)$ — both are okay.

E. Avoiding Runtime Errors in Expressions with Domain Predicates

- To avoid runtime failure of $\sigma(e)$, we'll take the context in which we're evaluating e and augment it with a predicate that guarantee non-failure of $\sigma(e)$. For example, for $\{P(e)\}\ v := e\ \{P(v)\}$, we'll augment the precondition to guarantee that evaluation of e won't fail.
- For each expression e, we will define a ***domain predicate*** $D(e)$ such that $\sigma \vDash D(e)$ implies $\sigma(e) \neq \bot_e$.
  - This predicate has to be defined recursively, since we need to handle complex expressions like $b[b[k]]$. As we'll see, $D(b[b[k]]) \equiv 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$.
  - As with wp, the domain predicate for an expression is unique only up to logical equivalence. For example, $D(x/y + u/v) \equiv y \neq 0 \wedge v \neq 0 \Leftrightarrow v * y \neq 0$. (As for me, I prefer $y \neq 0 \wedge v \neq 0$.)
- ***Definition***: ***(Domain predicate D(e) for expression e):*** We must define D for each kind of expression that can cause a runtime error:
  - First, a shortcut: if e contains no operations that can fail, then $D(e) \equiv T$.
    - For example, for a constant c or variable v, $D(c) \equiv T$ and $D(v) \equiv T$ because evaluation of a variable or constant doesn't cause failure,
  - The basic requirement is to define domain expressions for operations that can cause errors. For us, that's array lookup, division / modulus, and square root. Adding other operations or datatypes might introduce other cases.
    - $D(b[e]) \equiv D(e) \wedge 0 \le e < size(b)$.
    - $D(e_1 / e_2) \equiv D(e_1 \% e_2) \Leftrightarrow D(e_1) \wedge D(e_2) \wedge e_2 \neq 0$.
    - $D(sqrt(e)) \equiv D(e) \wedge e \ge 0$.
  - For operations that don't themselves cause errors, we simply check the subexpressions. This includes the arithmetic operators $+$, $-$, $*$, and the relational operators $\le$, $<$, $=$, $\neq$, $>$, and $\ge$.
    - $D(e_1\ op\ e_2) \equiv D(e_1) \wedge D(e_2)$, except for $op \equiv /$ or $\%$.
    - $D(op\ e) \equiv D(e)$.
    - $D(f(e_1, e_2, \ldots)) \equiv D(e_1) \wedge D(e_2) \wedge \ldots$, except for $f \equiv sqrt$.
  - For conditional statements, we need safety of the tests and safety of the arms / branches.
    - $D(\textbf{if}\ B\ \textbf{then}\ e_1\ \textbf{else}\ e_2\ \textbf{fi}) \equiv D(B) \wedge (B \rightarrow D(e_1)) \wedge (\neg B \rightarrow D(e_2))$
      - We have to guarantee that all the tests won't cause failure because failure of even one test will cause failure of the entire conditional. We need at least one guard to be true because otherwise the statement fails.

- ***Example 7***: $D(b[b[k]]) \equiv D(b[k]) \wedge 0 \le b[k] < size(b)$
  $\equiv D(k) \wedge 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$
  $\equiv T \wedge 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$
  $\equiv 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$

- ***Example 8***: *D((–b + sqrt(b*b – 4*a*c))/(2*a))*

  ≡ *D(e) ∧ D(2*a) ∧ 2*a ≠* 0 — where *e ≡ –b + sqrt(b*b – 4*a*c)*

  ≡ *D(–b) ∧ D(sqrt(b*b – 4*a*c)) ∧ D(2*a) ∧ 2*a ≠ 0*

  ≡ *D(sqrt(b*b – 4*a*c))*
  *∧ 2*a ≠ 0* — since *D(–b) ≡ D(2*a) ≡ T*

  ≡ *D(b*b – 4*a*c) ∧ (b*b – 4*a*c ≥ 0) ∧ 2*a ≠ 0*

  ≡ *b*b – 4*a*c ≥ 0 ∧ 2*a ≠ 0* — since *(b*b – 4*a*c ≥ 0) ≡ T*

  ⇔ *b*b – 4*a*c ≥ 0 ∧ a ≠ 0* — If asked to simplify arithmetically

- ***Example 9***: *D(**if** 0 ≤ k < size(b) **then** b[k] **else** 0 **fi**).* Here, the test guarantees that the array lookup won't fail.

  ≡ *D(B) ∧ (B → D(b[k]) ∧ (¬B → D(0))* — where *B ≡ 0 ≤ k < size(b)*

  ≡ *(B → D(b[k]) ∧ (¬B → T)* — since *D(B)* and *D(0) ≡ T*

  ⇔ *B → D(b[k])* — since *¬B → T ⇔ T*

  ≡ *B → D(k) ∧ 0 ≤ k < size(b)* — expanding *D(b[k])*

  ⇔ ***if*** *0 ≤ k < size(b) → T ∧ 0 ≤ k < size(b)* — since *B ≡ 0 ≤ k < size(b)*

  ⇔ *T*

F. Avoiding Runtime Errors in Statements with Domain Predicates

- Recall that we extended our notion of operational semantics to include $\langle S, \sigma \rangle \rightarrow^* \langle E, \perp_e \rangle$ to indicate that evaluation of *S* causes a runtime failure.
- We can avoid runtime failure of statements by adding domain predicates to the preconditions of statements. Though we can't in general calculate the *wlp/wp* of a loop, we can calculate a domain predicate for it.
- ***Definition***: For statement *S*, the predicate *D(S)* gives a sufficient condition to avoid runtime errors. For loops, avoiding divergence is a separate problem we'll look at later.
  - *D(**skip**) ≡ T*
  - *D(v := e) ≡ D(e)*
  - *D(b[e₁] := e₂) ≡ D(b[e₁]) ∧ D(e₂)*
  - *D(S₁ ; S₂) ≡ D(S₁) ∧ wp(S₁, D(S₂))*
    - Running S_1 when $D(S_1)$ holds tells us S_1 won't cause an error. Running S_1 when $wp(S_1, D(S_2))$ holds tells us that S_1 will establish $D(S_2)$, so running S_2 won't cause an error. To see this,
      - If $\sigma \vDash D(S_1)$ then $\perp_e \notin M(S_1, \sigma)$.
      - If $\sigma \vDash wp(S_1, D(S_2))$, then $M(S_1, \sigma) \vDash D(S_2)$, which implies $\perp_e \notin M(S_2, M(S_1, \sigma))$.
      - Combining $\perp_e \notin M(S_1, \sigma)$ and $\perp_e \notin M(S_2, M(S_1, \sigma))$ tells us $\perp_e \notin M(S_1 ; S_2, \sigma)$.
  - *D(**if** B **then** S₁ **else** S₂ **fi**, q)*
    ≡ *D(B) ∧ (B → D(S₁)) ∧ (¬B → D(S₂))*

- $D(\textbf{if } B_1 \to S_1 \;\square\; B_2 \to S_2 \textbf{ fi}, q)$
  $\equiv D(B_1 \vee B_2) \wedge (B_1 \vee B_2) \wedge (B_1 \to D(S_1)) \wedge (B_2 \to D(S_2))$
  - Note we need $(B_1 \vee B_2)$ to avoid failure of the nondeterministic ***if-fi*** due to none of the guards holding.
  - This definition extends easily to ***if-fi*** with $\neq$ two guarded commands.
- $D(\textbf{while } B \textbf{ do } S_1 \textbf{ od}) \equiv D(B) \wedge (B \to D(S_1))$
- $D(\textbf{do } B_1 \to S_1 \;\square\; B_2 \to S_2 \textbf{ od})$
  $\equiv D(B_1 \vee B_2) \wedge (B_1 \to D(S_1)) \wedge (B_2 \to D(S_2))$
  - This definition extends easily to ***do-od*** with $\neq$ two guarded commands.
  - The domain predicate for nondeterministic ***do-od*** is like that for ***if-fi*** except that having none of the guards hold does not cause an error.
  - Note ***while*** B ***do*** S_1 ***od*** is equivalent to ***do*** $B \to S$ ***od***, and happily, their D results match.

Calculating wp for loop-free programs

- With the domain predicates, it's easy to extend *wlp* for *wp* for loop-free programs because we don't have to argue for termination of a loop.
- ***Definition***: $wp(S, q) \equiv D(S) \wedge w \wedge D(w)$, where $w \equiv wlp(S, q)$.
  - $D(S)$ tells us that running S won't cause an error
  - w tells us that running S will establish q (if S terminates).
  - $D(w)$ tells us that w makes sense.

- ***Example 10***: If a program does a division, then the *wp* and *wlp* can differ.
  - We'll calculate $w_1 \equiv wp(S_1; S_2, q)$ where
    $S_1 \equiv x := y$, $S_2 \equiv z := v/x$, and $q \equiv z > x+2$.
  - Since $w_1 \equiv wp(S_1; S_2, q) \equiv wp(S_1, wp(S_2, q))$, we should calculate $w_2 \equiv wp(S_2, q)$ first.

    $w_2 \equiv wp(S_2, q)$
    $\equiv wp(z := v/x, z > x+2)$
    $\equiv D(z := v/x) \wedge w \wedge D(w)$ *where* $w \equiv wlp(z := v/x, z > x+2) \equiv v/x > x+2$
    $\equiv (x \neq 0) \wedge (v/x > x+2) \wedge D(v/x > x+2)$
    $\equiv x \neq 0 \wedge v/x > x+2 \wedge x \neq 0$
    $\equiv x \neq 0 \wedge v/x > x+2$ [1]
  - So now we can calculate $w_1 \equiv wp(S_1, w_2)$.

    $w_1 \equiv wp(S_1, w_2)$
    $\equiv wp(x := y, x \neq 0 \wedge v/x > x+2)$

[1] To simplify syntactic/semantic calculations, let's again extend our notion of $\equiv$ so that $p \wedge p \equiv p \vee p \equiv p$.

$\equiv wlp(x := y, x \neq 0 \wedge v/x > x+2)$ since the assignment $x := y$ never fails
$\equiv y \neq 0 \wedge v/y > y+2$

- ***Example 11***: Let's calculate $w_0 \equiv wp(x := b[k], sqrt(x) \geq 1)$.
  - Let $S \equiv x := b[k]$, $q \equiv sqrt(x) \geq 1$, and $w \equiv wlp(S, q)$.
  - We can expand $w \equiv wlp(S, q) \equiv wlp(x := b[k], sqrt(x) \geq 1) \equiv sqrt(b[k]) \geq 1$.
  - It's also useful to calculate

    $D(sqrt(b[k]) \geq 1)$
    $\equiv D(b[k]) \wedge b[k] \geq 0$
    $\equiv 0 \leq k < size(b) \wedge b[k] \geq 0$
  - So then

    $w_0 \equiv wp(S, q)$
    $\equiv D(S) \wedge w \wedge D(w)$
    $\equiv D(x := b[k])$
    $\wedge (sqrt(b[k]) \geq 1)$
    $\wedge D(sqrt(b[k]) \geq 1)$
    $\equiv (0 \leq k < size(b))$
    $\wedge (sqrt(b[k]) \geq 1)$
    $\wedge (0 \leq k < size(b) \wedge b[k] \geq 0)$
    $\equiv 0 \leq k < size(b) \wedge sqrt(b[k]) \geq 1 \wedge b[k] \geq 0$ Dropping repeated $0 \leq k < size(b)$.
  - If further simplification is requested, we have

    $\Leftrightarrow 0 \leq k < size(b) \wedge b[k] \geq 1$ Arithmetic simplification

