Weakest Preconditions

Part 2: Calculating wp, wlp; Domain Predicates

CS 536: Science of Programming, Spring 2022

Calculating wlp for Loop-Free Programs

- It's easy to calculate the *wp* and *wlp* of a loop-free/error-free program *S* especially since for such programs, the *wp* and *wlp* are identical.
- The following algorithm takes *S* and *q* and syntactically calculates a particular predicate for *wlp(S, q)*, which is why it's described using *wlp(S, q)* ≡ ... instead of *wp(S, q)* ⇔ ....
  - *wlp(skip, q)* ≡ *q*
  - *wlp(x := e,* q*)* ≡ [e/x]q
  - *wlp(s₁;s₂, q)* ≡ *wlp(s₁, wlp(s₂,q))*
  - *wlp(if e then* {s₁*} else* {s₂*}, q)* ≡ *(e → w₁) ∧ (¬e → w₂)* where *w₁* ≡ *wlp(s₁,q)* and *w₂* ≡ *wlp(s₂,q)*.
    - Since it's equivalent, you can also use *(e ∧ w₁) ∨ (¬e ∧ w₂)*.

Some Examples of Calculating wp/wlp:

- The programs in these examples never end in "state" ⊥, so the *wp* and *wlp* are equivalent.
- Example 2: *wlp(x := x+1, x ≥ 0)* ≡ [x+1/x](x ≥ 0) = *x+1 ≥ 0*
- Example 3: *wlp(y := y+x; x := x+1, x ≥ 0)*
  ≡ *wlp(y := y+x, wlp(x := x+1, x ≥ 0))*
  ≡ *wlp(y := y+x, x+1 ≥ 0)* ≡ [y+x/y](x+1) = *x+1 ≥ 0*
- Example 4: *wlp(y := y+x; x := x+1, x ≥ y)*
  ≡ *wlp(y := y+x, wlp(x := x+1, x ≥ y))*
  ≡ *wlp(y := y+x, x+1 ≥ y)*
  ≡ [y+x/y](x+1 ≥ y) = *x+1 ≥ y+x*
- If we were asked only to calculate the *wlp*, we'd stop here. If we also wanted to logically simplify the *wlp* then *x+1 ≥ y+x ⇔ y ≤ 1*.
- Example 5: Swap the two assignments in Example 4:
  wlp(x := x+1; y := y+x, x ≥ y)
  ≡ *wlp(x := x+1, wlp(y := y+x, x ≥ y))*
  ≡ *wlp(x := x+1, x ≥ y+x))*
  ≡ *x+1 ≥ y+x+1 ⇔ y ≤ 0* if you want to logically simplify*]*

- Example 6: Calculate *wlp(if (x ≥ y) then {m := x} else {skip}, m = max(x, y)).*
  - *= (x ≥ y -> wlp(m := x, m = max(x, y))) ∧ (x < y -> wlp(skip, m = max(x, y))*

    ⇔ (x ≥ *y -> x = max(x, y)) ∧ (x < y -> m = max(x, y))*

    ⇔ *T ∧ (x < y -> m = y)*
- Example 6.5: *wlp(if y ≥ 0 then {x := y} else {skip}, x ≥ 0)*

  ≡ *(y ≥ 0 → wlp(x := y, x ≥ 0)) ∧ (y < 0 → wlp(skip, x ≥ 0))*

  ≡ *(y ≥ 0 → y ≥ 0) ∧ (y < 0 → x ≥ 0)*
  - It's also okay to use *(y ≥ 0 ∧ y ≥ 0) ∨ (y < 0 ∧ x ≥ 0)*.
  - If we want to simplify logically, we can continue with

    ⇔ *y ≥ 0 ∨ (y < 0 ∧ x ≥ 0)*

    ⇔ *(y ≥ 0 ∨ y < 0) ∧ (y ≥ 0 ∨ x ≥ 0)*

    ⇔ *y ≥ 0 ∨ x ≥ 0* *(*which is also ⇔ *y < 0 → x ≥ 0*, if you prefer*)*

Domain Predicates for Avoiding Runtime Errors in Expressions

- To get from wlp to wp (for loop-free programs), we also need to rule out failures.
- To avoid runtime failure of *σ(e)*, we'll take the context in which we're evaluating *e* and augment it with a predicate that guarantee non-failure of *σ(e)*. For example, for *{P(e)} x := 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 *σ* ⊨ *D(e)* implies *σ(e)* ≠ $\perp_e$.
  - This predicate has to be defined recursively, since we need to handle complex expressions like *D(b[b[k]]) ≡ 0 ≤ k < size(b) ∧ 0 ≤ b[k] < size(b)*.
  - As with *wp* and *sp*, the domain predicate for an expression is unique only up to logical equivalence. For example, *D(x/y + u/v) ≡ y ≠ 0 ∧ v ≠ 0 ⇔ v ≠ 0 ∧ y ≠ 0*.
- Definition *(*Domain predicate *D(e)* for expression *e)* We must define *D* for each kind of expression that can cause a runtime error:
  - *D(c) ≡ D(v) ≡ T* if where *c* is a constant and *v* is a variable.
    - Evaluation of a variable or constant doesn't cause failure.
  - *D(b[e]) ≡ D(e) ∧ 0 ≤ e < size(b)*
  - *D(e_1/e_2) ≡ D(e_1 % e_2) ⇔ D(e_1) ∧ D(e_2) ∧ e_2 ≠ 0*
  - *D(sqrt(e)) ≡ D(e) ∧ e ≥ 0*
    - And so on, depending on the datatypes and operations being used.
  - The various operations *(+, –,* etc.*)* and relations *(≤, =,* etc.*)* don't cause errors but we still have to check their subexpressions:
  - *D(e_1 op e_2) ≡ D(e_1) ∧ D(e_2)*, except for *op* ≡ */* or *%*
  - *D(op e) ≡ D(e)*, unless you add an operator that can cause runtime failure.
  - *D(e_1 ? e_2 : e_3) ≡ D(e_1) ∧ (e_1 → D(e_2)) ∧ (¬e_1 → D(e_3))*

- *(*For a conditional expression, we only need safety of the one branch we execute.*)*

- Example 7: $D(b[b[k]]) \equiv D(b[k]) \wedge 0 \le b[k] < size(b)$
  $\equiv D(k) \wedge 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$
  $\Leftrightarrow 0 \le k < size(b) \wedge 0 \le b[k] < size(b)$
- Example 8: $D((-b + sqrt(b*b - 4*a*c))/(2*a))$
  $\equiv D(e) \wedge D(2*a) \wedge 2*a \neq 0$ where $e \equiv -b + sqrt(b*b - 4*a*c)$
  $\equiv D(-b) \wedge D(sqrt(b*b - 4*a*c)) \wedge D(2*a) \wedge 2*a \neq 0$
  $\Leftrightarrow D(sqrt(b*b - 4*a*c)) \wedge 2*a \neq 0$ since $D(-b) \equiv D(2*a) \equiv T$
  $\equiv D(b*b - 4*a*c) \wedge (b*b - 4*a*c \ge 0) \wedge 2*a \neq 0$
  $\Leftrightarrow b*b - 4*a*c \ge 0 \wedge 2*a \neq 0$
- Example 9: $D(0 \le k < size(b) \ ? \ b[k] : 0)$
  $\equiv D(e) \wedge (e \rightarrow D(b[k])) \wedge (\neg e \rightarrow D(0))$ where $e \equiv 0 \le k < size(b)$
  $\equiv (e \rightarrow D(b[k])) \wedge (\neg e \rightarrow T)$ since $D(B)$ and $D(0) \equiv T$
  $\Leftrightarrow e \rightarrow D(b[k])$ since everything implies true
  $\equiv e \rightarrow (D(k) \wedge 0 \le k < size(b))$ expanding $D(b[k])$
  $\Leftrightarrow e \rightarrow (T \wedge e)$ since $e \equiv 0 \le k < size(b)$
  $\Leftrightarrow T$

Domain Predicates for Avoiding Runtime Errors in Programs

- 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 for loops we can't in general calculate the *wlp/wp*, we can calculate the domain predicate for them.
- Definition: For statement S, the predicate $D(S)$ gives a sufficient condition to avoid runtime errors.
  - $D(skip) \equiv T$
  - $D(x := e) \equiv D(e)$
  - $D(b[e_1] := e_2) \equiv D(b[e_1]) \wedge D(e_2)$
  - $D(s_1 ; s_2) \equiv D(s_1) \wedge wp(s_1, D(s_2))$

    [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.]
    - If $\sigma \models D(s_1)$ then $\perp_e \notin M(s_1, \sigma)$.
    - If $\sigma \models wp(s_1, D(s_2))$, then $M(s_1, \sigma) \models D(s_2)$.
      - If $M(s_1, \sigma) \models D(s_2)$, then $\perp \notin M(s_2, M(s_1, \sigma))$.
  - $D(\text{if } e \text{ then } \{s_1\} \text{ else } \{s_2\}, q) \equiv D(e) \wedge (e \rightarrow D(s_1)) \wedge (\neg e \rightarrow D(s_2))$

- $D(\text{while } e \text{ do } \{s_1\}) \equiv D(e) \wedge (e \rightarrow D(s_1))$

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: $wp(s, q) \equiv D(s) \wedge wlp(s, q)$.
  - $D(S)$ tells us that running S won't cause an error, and $wlp(s, q)$ tells us that running S will establish q (if S terminates).
- Example 10: If a program does a division, then the *wp* and *wlp* can differ.
  - Let $p_2 \equiv wp(x := y; z := v/x, z > x+2) \equiv wp(x := y, p_1)$
  - Where $p_1 \equiv wp(z := v/x, z > x+2) \equiv D(z := v/x) \wedge w$
    where $w \equiv wlp(z := v/x, z > x+2) \equiv v/x > x+2$
    $p_1 \equiv D(z := v/x) \wedge v/x > x+2 \equiv x \neq 0 \wedge v/x > x+2$
  - So $p_2 \equiv wp(x := y, p_1) \equiv wp(x := y, x \neq 0 \wedge v/x > x+2)$
    $\equiv wlp(x := y, x \neq 0 \wedge v/x > x+2)$, since $x := y$ causes no errors
    $\equiv y \neq 0 \wedge v/y > y+2$