

Class 09: Termination, Weakest Preconditions

A. Why?

- Diverging programs aren't useful.
- Weakest preconditions describe the most general start states for a correct program.

B. Outcomes

By the end of the class you should

- Understand the loop bound method of ensuring termination
- Know the extra obligations required to prove that a partially correct program is totally correct.
- Know what weakest preconditions (wp) and weakest liberal preconditions (wlp) are.
- Know the relationship between wp and wlp and how they are related.
- Know how to calculate the wp or wlp of loop-free programs.

C. Midterm Exam Solution

D. Semantics of Termination

- For us, program termination means normal termination (not including runtime errors).
- For semantics, we have \mathcal{M}_{tot} that is slightly different from \mathcal{M} because it explicitly indicates nontermination.
 - $\mathcal{M}_{tot}\llbracket S \rrbracket(\sigma) = \{\perp\}$ if $\mathcal{M}\llbracket S \rrbracket(\sigma) = \emptyset$
 - $\mathcal{M}_{tot}\llbracket S \rrbracket(\sigma) = \mathcal{M}\llbracket S \rrbracket(\sigma)$ if $\mathcal{M}\llbracket S \rrbracket(\sigma) \neq \emptyset$,
 - The “ \perp ” symbol is pronounced “bottom”.
- Properties of \perp
 - \perp is considered to be a state but not an actual memory state. For completeness
 - Define $\mathcal{M}_{tot}\llbracket S \rrbracket(\perp) = \{\perp\}$.
 - Define $\perp \not\models p$ and $\perp \not\models \neg p$.
 - So now $(\sigma \not\models p \text{ implies } \sigma \models \neg p)$ iff $\sigma \neq \perp$.
 - For validity of predicates, omit \perp : $(\sigma \models p)$ iff (for all $\sigma \neq \perp$, $\sigma \models p$).
- Total correctness semantics of correctness triples
 - Define $\sigma \models_{tot} \{p\} S \{q\}$ iff $(\sigma \models p)$ implies $(\perp \notin \mathcal{M}_{tot}\llbracket S \rrbracket(\sigma) \text{ and } \mathcal{M}_{tot}\llbracket S \rrbracket(\sigma) \models q)$
 - Since $\perp \not\models q$, we can abbreviate this to $(\sigma \models p)$ implies $(\mathcal{M}_{tot}\llbracket S \rrbracket(\sigma) \models q)$.

E. Proving Termination of Loops

- How can we ensure our loops terminate?

- For some loops, we can calculate number of iterations left:
 - E.g., at each loop test, $i := 0$; **while** $i < n$ **do** ...; $i := i + 1$ **od** has $n - i$ iterations left.
 - Can't calculate number of iterations for all loops (see advanced algorithms course for uncomputable functions).
- It's sufficient to find a decreasing upper bound for the number of iterations.
 - At each while test, the upper bound must be ≥ 0 .
 - Ensure by requiring loop invariant $p \rightarrow$ upper bound ≥ 0 .
 - Each iteration must reduce the upper bound.
 - $\{p \wedge B \wedge \text{upper bound} = \text{some value } z\}$
 - Loop body*
 - $\{\text{upper bound} < z\}$
- **Syntax:** Attach the upper bound to loop using **{bd t}**
 - t is the **bound expression** (a.k.a. **bound function**): It uses the loop variables to give an upper bound on the number of iterations to do.
- Example: For the $\text{sum}(0, n)$ program, use bound $n - i$:
 - $\{n \geq 0\} \ i := 0; \ s := 0;$
 - $\{\text{inv } p\} \ \{\text{bd } n - i\} \ \text{while } i < n \ \text{do } i := i + 1; \ s := s + i \ \text{od}$
 - $\{s = \text{sum}(0, n)\}$
 - At the loop test, we always have ≥ 0 iterations left
 - $(p \rightarrow n - i \geq 0)$ because p implies $0 \leq i \leq n$,
 - Execution of the loop body lowers the number of iterations:
 - $\{p \wedge i < n \wedge n - i = z\}$ loop body $\{n - i < z\}$ where z is a fresh variable.
- **The upper bound does not have to be tight!**
 - Can use tight bounds; they're just not required.
 - **Not required:** $p \wedge \neg B \rightarrow t = 0$.
 - **Not required:** $\{p \wedge B \wedge t = z\}$ loop body $\{t = z - 1\}$
- Example:
 - For binary search, $t = rt - lt$ is a loose upper bound (ceiling \log_2 is tighter).

F. Heuristics for finding a bound function

- How can we find a bound function t ? No algorithm but exist guidelines:
 1. t can't be constant: Must include ≥ 1 variable that changes.
 2. Don't allow $t \geq 0 \rightarrow$ while loop test (otherwise you diverge).
 3. If the loop body makes a variable x smaller, include x as a term in t .
 4. If the loop body makes a variable y larger, include $(-y)$ as a term in t .
 5. If your current guess for t can be < 0 , find a large value to add to it to ensure $t \geq 0$ [big constants are nice].

- Example 1: For a loop that sets $j := j - 1$, try j for t .
 - If invariant includes $j \geq 0$, then (invariant $\rightarrow t \geq 0$).
- Example 2: For a loop that sets $i := i + 1$, try $(-i)$ for t .
 - If $(-i)$ can be < 0 , we should add something to $(-i)$.
 - If invariant includes a clause $i \leq s$, add s to the bound function to ensure that (invariant $\rightarrow t = s - i \geq 0$). Try $(-i) + s$ (i.e., $s - i$) for t .
- Example 3: $gcd(a, b)$ sometimes makes a smaller, sometimes makes b smaller.
 - Try $a + b$ for t . **Invariant implies $a \geq 0$ and $b \geq 0$, so $a + b \geq 0$.**
- Example 4: $binsearch(a, n, x)$ sometimes makes lt **larger**, sometimes makes rt **smaller**.
 - Try $(-lt) + rt = rt - lt$ for t . Invariant implies $lt < rt$, so $rt - lt \geq 0$.
- The heuristics can fail. Example:


```
while m ≥ 0 do
  if m % 3 != 0 then m := m - 2 else m := m + 1 fi od
```
- The heuristics have us try $t = m - m$, which doesn't work.

G. System TD (Total Correctness of Deterministic Programs)

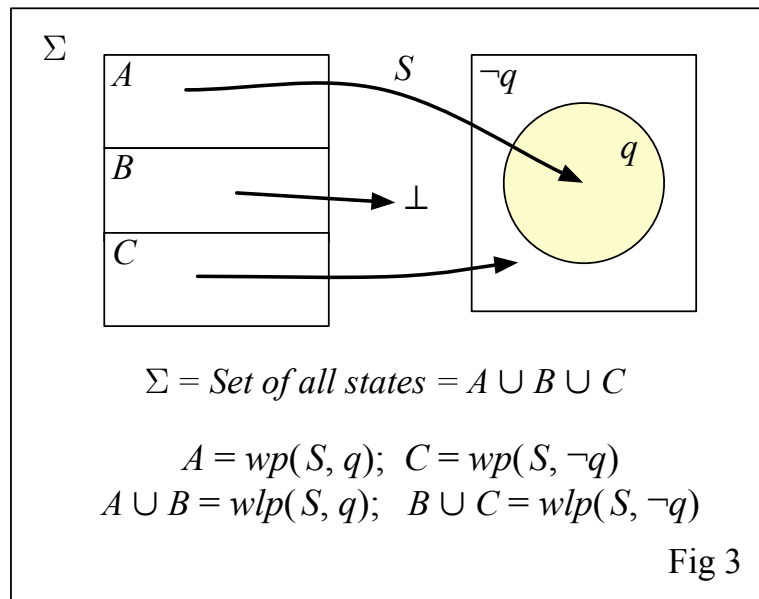
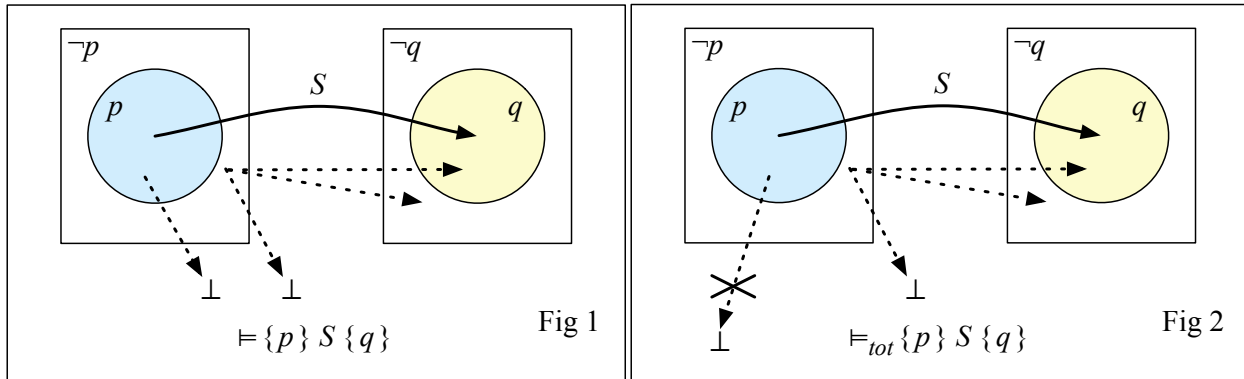
- System TD is System PD (partial correctness) with a modified loop rule to prove termination.
 - System PD loop rule is
 - If $\{p \wedge B\} S \{p\}$, then **$\{\text{inv } p\} \text{ while } B \text{ do } S \text{ od } \{p \wedge \neg B\}$**
 - For system TD, we add the bound function t and get the rule
 - If $\{p \wedge B\} S \{p\}$, $p \rightarrow t \geq 0$, and $\{t = z\} S \{t < z\}$, then


```
\{\text{inv } p\} \{\text{bd } t\} \text{ while } B \text{ do } S \text{ od } \{p \wedge \neg B\}
```
- To show that a loop is totally correct, you need
 1. $\{p \wedge B \wedge t = z\} S \{p \wedge t < z\}$ (compared to PD's $\{p \wedge B\} S \{p\}$)
 2. The predicate logic obligation $p \rightarrow t \geq 0$.

H. Weakest Preconditions and Weakest Liberal Preconditions

- We know $\{p\} S \{q\}$ doesn't generally say anything about $\{\neg p\} S \{???\}$
 - But, take $\{p\} x := x + 1 \{x \geq 0\}$ and imagine these possible values for p :
 - $x \geq 3, x \geq 2, x \geq 1, x \geq 0, x \geq -1, x \geq -2, x \geq -3, \dots$
 - Going left to right, the predicates get weaker.
 - Every predicate from $x \geq -1$ leftward works for p .
 - All the predicates right of $x \geq -1$ are too weak to use for p .
 - $x \geq -1$ is the **weakest precondition** for $x := x + 1$ and $x \geq 0$.
 - The most general predicate that still works as a precondition.
- Definition: p is the **weakest precondition** for S and q (**notation**: p is $wp(S, q)$) if
 - It's a precondition: $\models_{tot} \{p\} S \{q\}$

- It's weakest: For any precondition p' where $\models_{tot} \{p'\} S \{q\}$, we have $\models p' \rightarrow p$.
- For partial correctness, we have weakest liberal precondition (notation $wlp(S, q)$):
 - Definition: p is the weakest liberal precondition for S and q if
 - It's a precondition: $\models \{p\} S \{q\}$
 - It's weakest: For any precondition p' where $\models_{tot} \{p'\} S \{q\}$, we have $\models p' \rightarrow p$.



(Offline discussion: What do the labels “ p ”, “ $\neg p$ ”, A , B , C , etc. mean?)

- Every predicate corresponds to the set of states that satisfy it; the book uses
 - $\llbracket p \rrbracket = \{\sigma \in \Sigma \mid \sigma \models p\}$, where Σ is the set of all states.
 - (Σ doesn't include \perp .)
- In the other direction, every set of states corresponds to some predicate.
 - (Very weird sets may require an infinitely long predicate, but one can prove that the kinds of sets we deal with do in fact correspond to plain old predicates.)
- In the figures above
 - The labels “ p ” and “ $\neg p$ ” are abbreviations for $\llbracket p \rrbracket$ and $\llbracket \neg p \rrbracket$; they partition Σ .

- The names A , B , and C refer to sets of states that form a partition of Σ .
- So saying “ $A = wp(S, q)$ ” and “ $A \cup B = wlp(S, q)$ ” talks about sets of states.
- If we say “ p is the $wp(S, q)$ ”, then that’s shorthand for $\llbracket p \rrbracket = wp(S, q)$.
- Uniqueness:
 - The set of states $wp(S, q)$ is unique, but if $\llbracket p \rrbracket = wp(S, q)$, there are infinitely many other predicates that are logically equivalent to p (e.g. $p \wedge \mathbf{true}$).
 - So saying “ p is **the** $wp(S, q)$ ” is a bit of a misnomer; p is only unique up to logical equivalence.
 - Similarly, $wlp(S, q)$ is a unique set of states but a predicate for $wlp(S, q)$ is unique only up to logical equivalence.

(End of offline discussion)

I. Calculating wp and wlp of loop-free programs

- If S has no loops, then $wp(S, q)$ and $wlp(S, q)$ are logically equivalent.
- $wp(u := t, p) \equiv p[u := t]$
- $wp(S_1; S_2) \equiv wp(S_1, wp(S_2, q))$
- $wp(\mathbf{if } B \mathbf{ then } S_1 \mathbf{ else } S_2 \mathbf{ fi}, p) \equiv (B \wedge wp(S_1, p)) \vee (\neg B \wedge wp(S_2, p))$
- $wp(\mathbf{skip}, p) \equiv p$

Some examples:

- $wp(x := x+1, x \geq 0)$ is $x+1 \geq 0$
- $wp(y := y+x; x := x+1, x \geq 0)$
 - $\Leftrightarrow wp(y := y+x, wp(x := x+1, x \geq 0))$
 - $\Leftrightarrow wp(y := y+x, x+1 \geq 0)$
 - $\Leftrightarrow x+1 \geq 0$
- $wp(y := y+x; x := x+1, x \geq y)$
 - $\Leftrightarrow wp(y := y+x, wp(x := x+1, x \geq y))$
 - $\Leftrightarrow wp(y := y+x, x+1 \geq y)$
 - $\Leftrightarrow x+1 \geq y+x$ (so $y \leq 1$)
- $wp(\mathbf{if } y \geq 0 \mathbf{ then } x := y \mathbf{ fi}, x \geq 0)$
 - $\Leftrightarrow wp(\mathbf{if } y \geq 0 \mathbf{ then } x := y \mathbf{ else skip fi}, x \geq 0)$
 - $\Leftrightarrow (y \geq 0 \wedge wp(x := y, x \geq 0)) \vee (y \leq 0 \wedge wp(\mathbf{skip}, x \geq 0))$
 - $\Leftrightarrow (y \geq 0 \wedge y \geq 0) \vee (y \leq 0 \wedge x \geq 0)$
 - $\Leftrightarrow (y \leq 0 \rightarrow x \geq 0)$