Methodologies for Software Processes

Lecture 7- Hoare Logic

(The lecture slides and the notes are taken from Prof. Mike Gordon from Cambridge University)

Dijkstra's weakest preconditions

- Weakest preconditions is a theory of refinement
  - idea is to calculate a program to achieve a postcondition
  - not a theory of post hoc verification
- Non-determinism a key idea in Dijksta's presentation
  - start with a non-deterministic high level pseudo-code
  - refine to deterministic and efficient code
- Weakest preconditions (wp) are for total correctness
- Weakest *liberal* preconditions (wlp) for partial correctness
- If C is a command and Q a predicate, then informally:
  - $\texttt{wlp}(C, Q)$ = '*The weakest predicate* P *such that* $\{P\}\ C\ \{Q\}$'
  - $\texttt{wp}(C, Q)$ = '*The weakest predicate* P *such that* $[P]\ C\ [Q]$'
- If P and Q are predicates then $Q \Rightarrow P$ means P is 'weaker' than Q

Rules for weakest preconditions

- Relation with Hoare specifications:

$$\begin{array}{lcl} \{P\}\ C\ \{Q\} & \Leftrightarrow & P \Rightarrow \texttt{wlp}(C,Q) \\ [P]\ C\ [Q] & \Leftrightarrow & P \Rightarrow \texttt{wp}(C,Q) \end{array}$$

- Dijkstra gives rules for computing weakest preconditions:

$$\begin{array}{lcl} \texttt{wp}(V\texttt{:=}E, Q) & = & Q[E/V] \\ \texttt{wp}(C_1\texttt{;}C_2,\ Q) & = & \texttt{wp}(C_1, \texttt{wp}(C_2,\ Q)) \\ \texttt{wp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) & = & (S \Rightarrow \texttt{wp}(C_1,Q)) \wedge (\neg S \Rightarrow \texttt{wp}(C_2,Q)) \end{array}$$

  for deterministic loop-free code the same equations hold for `wlp`

- Rule for `WHILE`-commands doesn't give a first order result

- Weakest preconditions closely related to verification conditions

- VCs for $\{P\}\ C\ \{Q\}$ are related to $P \Rightarrow \texttt{wlp}(C,Q)$

  - VCs use annotations to ensure first order formulas can be generated

Sequencing example

- Swapping variables:

  $$\begin{aligned}
  &\texttt{wlp(R:=X; X:=Y; Y:=R}, (\texttt{Y} = x \wedge \texttt{X} = y))\\
  &\quad = \texttt{wlp(R:=X}, \ \texttt{wlp(X:=Y}, \ \texttt{wlp(Y:=R}, \ (\texttt{Y} = x \wedge \texttt{X} = y)))\\
  &\quad = \texttt{wlp(R:=X}, \ \texttt{wlp(X:=Y}, \ (\texttt{Y} = x \wedge \texttt{X} = y)\texttt{[R/Y]}))\\
  &\quad = \texttt{wlp(R:=X}, \ \texttt{wlp(X:=Y}, \ (\texttt{R} = x \wedge \texttt{X} = y)))\\
  &\quad = \texttt{wlp(R:=X}, \ (\texttt{R} = x \wedge \texttt{Y} = y))\\
  &\quad = (\texttt{X} = x \wedge \texttt{Y} = y)
  \end{aligned}$$

- So since $\{P\}\ C\ \{Q\} \ \Leftrightarrow \ P \ \Rightarrow \ \texttt{wlp}(C, Q)$

  to prove

  $\{\texttt{X} = x \wedge \texttt{Y} = y\}\ \texttt{R:=X; X:=Y; Y:=R}\ \{\texttt{Y} = x \wedge \texttt{X} = y\}$

  just need to prove:

  $(\texttt{X} = x \wedge \texttt{Y} = y) \Rightarrow (\texttt{X} = x \wedge \texttt{Y} = y)$

  which is clearly true (instance of $S \Rightarrow S$)

Conditional example

- Compute `wlp` of the maximum program:

$$\texttt{wlp(IF X<Y THEN MAX:=Y ELSE MAX:=X}, (\texttt{MAX} = max(x,y))$$

$$= (\texttt{X<Y} \;\Rightarrow\; \texttt{wlp(MAX:=Y}, \; (\texttt{MAX} = max(x,y))))$$
$$\wedge$$
$$(\neg(\texttt{X<Y}) \;\Rightarrow\; \texttt{wlp(MAX:=X}, \; (\texttt{MAX} = max(x,y))))$$

$$= (\texttt{X<Y} \;\Rightarrow\; \texttt{Y} = max(x,y)) \;\wedge\; (\neg(\texttt{X<Y}) \;\Rightarrow\; \texttt{X} = max(x,y))$$

$$= \mathit{if}\ \texttt{X<Y}\ \mathit{then}\ \texttt{Y} = max(x,y)\ \mathit{else}\ \texttt{X} = max(x,y)$$

- So to prove

$$\{\texttt{X} = x \wedge \texttt{Y} = y\}\ \texttt{IF X<Y THEN MAX:=X ELSE MAX:=Y}\ \{\texttt{MAX} = max(x,y)\}$$

just prove:

$$(\texttt{X} = x \wedge \texttt{Y} = y) \Rightarrow (\texttt{X<Y} \;\Rightarrow\; \texttt{Y} = max(x,y)) \;\wedge\; (\neg(\texttt{X<Y}) \;\Rightarrow\; \texttt{X} = max(x,y))$$

which follows from the defining property of max

$$\vdash \;\; \forall x\ y.\ (x \geq y \Rightarrow x = max(x,y)) \;\wedge\; (\neg(x \geq y) \Rightarrow y = max(x,y))$$

Using wlp to improve verification condition method

- If C is loop-free then VC for $\{P\}\ C\ \{Q\}$ is $P \Rightarrow \texttt{wlp}(C, Q)$
  - no annotations needed in sequences!

- Cannot in general compute a finite formula for $\texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q)$

- The following holds

  $\texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q)\ =\ \mathit{if}\ S\ \mathit{then}\ \texttt{wlp}(C,\ \texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q))\ \mathit{else}\ Q$

- Above doesn't define $\texttt{wlp}(C, Q)$ as a finite statement

- Could use a hybrid VC and wlp method

wlp-based verification condition method

- We define $\texttt{awp}(C, Q)$ and $\texttt{wvc}(C, Q)$
  - $\texttt{awp}(C, Q)$ is a statement sort of approximating $\texttt{wlp}(C, Q)$
  - $\texttt{wvc}(C, Q)$ is a set of verification conditions
- If C is loop-free then
  - $\texttt{awp}(C, Q) \;=\; \texttt{wlp}(C, Q)$
  - $\texttt{wvc}(C, Q) \;=\; \{\}$
- Denote by $\bigwedge\mathcal{S}$ the conjunction of all the statements in $\mathcal{S}$
  - $\bigwedge\{\} = \texttt{T}$
  - $\bigwedge(\mathcal{S}_1 \cup \mathcal{S}_2) = (\bigwedge\mathcal{S}_1) \wedge (\bigwedge\mathcal{S}_2)$
- It will follow that $\bigwedge\texttt{wvc}(C, Q) \Rightarrow \{\texttt{awp}(C, Q)\}\ C\ \{Q\}$
- Hence to prove $\{P\}C\{Q\}$ it is sufficient to prove all the statements in $\texttt{wvc}(C, Q)$ and $P \Rightarrow \texttt{awp}(C, Q)$

Definition of awp

- Assume all `WHILE`-commands are annotated: $\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C$

- Define awp recursively by:

$$
\begin{array}{ll}
\texttt{awp}(V\ \texttt{:=}\ E,\ Q) & = Q[E/V] \\
\texttt{awp}(C_1\ \texttt{;}\ C_2,\ Q) & = \texttt{awp}(C_1,\ \texttt{awp}(C_2,\ Q)) \\
\texttt{awp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) & = (S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q)) \\
\texttt{awp}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,\ Q) & = R
\end{array}
$$

- Note:
$(S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q)\ =\ \mathit{if}\ S\ \mathit{then}\ \texttt{awp}(C_1,\ Q)\ \mathit{else}\ \texttt{awp}(C_2,\ Q)$

Definition of wvc

- Assume all WHILE-commands are annotated: WHILE S DO $\{R\}$ C

- Define wvc recursively by:

$$
\begin{aligned}
\texttt{wvc}(V\ \texttt{:=}\ E,\ Q) &= \{\} \\
\texttt{wvc}(C_1\ ;\ C_2,\ Q) &= \texttt{wvc}(C_1, \texttt{awp}(C_2, Q)) \cup \texttt{wvc}(C_2, Q) \\
\texttt{wvc}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) &= \texttt{wvc}(C_1,\ Q) \cup \texttt{wvc}(C_2,\ Q) \\
\texttt{wvc}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,\ Q) &= \{R \land \neg S \Rightarrow Q,\ R \land S \Rightarrow \texttt{awp}(C, R)\} \\
&\quad \cup \texttt{wvc}(C, R)
\end{aligned}
$$

Correctness of wlp-based verification conditions

- Theorem: $\bigwedge\texttt{wvc}(C,Q) \Rightarrow \{\texttt{awp}(C,Q)\}\ C\ \{Q\}$. Proof by Induction on C

  - $\bigwedge\texttt{wvc}(V\texttt{:=}E,Q) \Rightarrow \{\texttt{awp}(C,Q)\}\ C\ \{Q\}$ is $\texttt{T} \Rightarrow \{Q[E/V]\}\ V\ \texttt{:=}\ E\ \{Q\}$

  - $\bigwedge\texttt{wvc}(C_1\texttt{;}C_2,Q) \Rightarrow \{\texttt{awp}(C_1\texttt{;}C_2,Q)\}\ C_1\texttt{;}C_2\ \{Q\}$ is
    $\bigwedge(\texttt{wvc}(C_1,\texttt{awp}(C_2,Q)) \cup \texttt{wvc}(C_2,Q)) \Rightarrow \{\texttt{awp}(C_1,\ \texttt{awp}(C_2,Q))\}\ C_1\texttt{;}C_2\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C_2,Q) \Rightarrow \{\texttt{awp}(C_2,Q)\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_1,\texttt{awp}(C_2,Q)) \Rightarrow \{\texttt{awp}(C_1,\texttt{awp}(C_2,Q))\}\ C_2\ \{\texttt{awp}(C_2,Q)\}$,
    hence result by the Sequencing Rule.

  - $\bigwedge\texttt{wvc}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,Q)$
    $\Rightarrow \{\texttt{awp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,Q)\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$
    is $\bigwedge(\texttt{wvc}(C_1,\ Q) \cup \texttt{wvc}(C_2,\ Q))$
    $\Rightarrow \{(S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q)\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C_1,Q) \Rightarrow \{\texttt{awp}(C_1,Q)\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2,Q) \Rightarrow \{\texttt{awp}(C_2,Q)\}\ C_2\ \{Q\}$. Strengthening preconditions
    gives $\bigwedge\texttt{wvc}(C_1,Q) \Rightarrow \{\texttt{awp}(C_1,Q) \wedge S\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2,Q) \Rightarrow \{\texttt{awp}(C_2,Q) \wedge \neg S\}\ C_2\ \{Q\}$, hence
    $\bigwedge\texttt{wvc}(C_1,Q) \Rightarrow \{((S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q))) \wedge S\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2,Q) \Rightarrow \{((S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q))) \wedge \neg S\}\ C_2\ \{Q\}$,
    hence result by the Conditional Rule.

  - $\bigwedge\texttt{wvc}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,Q) \Rightarrow \{\texttt{awp}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,Q)\}\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C\ \{Q\}$
    is $\bigwedge(\{R \wedge \neg S \Rightarrow Q,\ R \wedge S \Rightarrow \texttt{awp}(C,R)\} \cup \texttt{wvc}(C,R)) \Rightarrow \{R\}\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C,R) \Rightarrow \{\texttt{awp}(C,R)\}\ C\ \{R\}$, hence result by WHILE-Rule.

Strongest postconditions

- Define $\mathtt{sp}(C, P)$ to be 'strongest' Q such that $\{P\}\ C\ \{Q\}$
  - partial correctness: $\{P\}\ C\ \{\mathtt{sp}(C, P)\}$
  - strongest means if $\{P\}\ C\ \{Q\}$ then $\mathtt{sp}(C, P) \Rightarrow Q$
- Note that `wlp` goes 'backwards', but `sp` goes 'forwards'
  - verification condition for $\{P\}\ C\ \{Q\}$ is: $\mathtt{sp}(C, P) \Rightarrow Q$
- By 'strongest' and Hoare logic postcondition weakening
  - $\{P\}\ C\ \{Q\}$ **if and only if** $\mathtt{sp}(C, P) \Rightarrow Q$

Strongest postconditions for loop-free code

- Only consider loop-free code
- $\mathtt{sp}(V\ \mathtt{:=}\ E,\ P)\ =\ \exists v.\ V = E\mathtt{[}v/V\mathtt{]} \wedge P\mathtt{[}v/V\mathtt{]}$
- $\mathtt{sp}(C_1\ \mathtt{;}\ C_2,\ P)\ =\ \mathtt{sp}(C_2,\ \mathtt{sp}(C_1,\ P))$
- $\mathtt{sp}(\mathtt{IF}\ S\ \mathtt{THEN}\ C_1\ \mathtt{ELSE}\ C_2,\ P)\ =\ \mathtt{sp}(C_1,\ P \wedge S)\ \vee\ \mathtt{sp}(C_2,\ P \wedge \neg S)$

- $\mathtt{sp}(V\mathtt{:=}E,\ P)$ corresponds to Floyd assignment axiom
- Can *dynamically prune* conditionals because $\mathtt{sp}(C, \mathtt{F}) = \mathtt{F}$
- Computer strongest postconditions is *symbolic execution*

Sequencing example

- $$\begin{aligned}
&\mathtt{sp}(\mathtt{R:=X;\ \ X:=Y;\ \ Y:=R},\ \mathtt{X} = x \wedge \mathtt{Y} = y)\\
&= \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ \mathtt{sp}(\mathtt{R:=X},\ \mathtt{X} = x \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\exists v.\ \mathtt{R} = \mathtt{X}[v/\mathtt{R}] \wedge (\mathtt{X} = x \wedge \mathtt{Y} = y)[v/\mathtt{R}])))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\exists v.\ \mathtt{R} = \mathtt{X} \wedge (\mathtt{X} = x \wedge \mathtt{Y} = y))))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\mathtt{R} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y}[v/\mathtt{X}] \wedge (\mathtt{R} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y)[v/\mathtt{X}]))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = v \wedge v = x \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge v = x \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge (\exists v.\ v = x) \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge \mathtt{T} \wedge \mathtt{Y} = y)))\\
&= \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge \mathtt{R} = x \wedge \mathtt{Y} = y))\\
&= \exists v.\ \mathtt{Y} = \mathtt{R}[v/\mathtt{Y}] \wedge (\mathtt{X} = \mathtt{Y} \wedge \mathtt{R} = x \wedge \mathtt{Y} = y)[v/\mathtt{Y}]\\
&= \exists v.\ \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = v \wedge \mathtt{R} = x \wedge v = y)\\
&= \exists v.\ \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge v = y)\\
&= \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge (\exists v.\ v = y))\\
&= \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge \mathtt{T})\\
&= \mathtt{Y} = \mathtt{R} \wedge \mathtt{X} = y \wedge \mathtt{R} = x\\
&= \mathtt{Y} = x \wedge \mathtt{X} = y \wedge \mathtt{R} = x
\end{aligned}$$

- So to prove $\{\mathtt{X} = x \wedge \mathtt{Y} = y\}$ `R:=X; X:=Y; Y:=R` $\{\mathtt{Y} = x \wedge \mathtt{X} = y\}$
  just prove: $(\mathtt{Y} = x \wedge \mathtt{X} = y \wedge \mathtt{R} = x) \Rightarrow \mathtt{Y} = x \wedge \mathtt{X} = y$

Conditional example

- Compute sp of the maximum program:

$$
\begin{aligned}
& \mathtt{sp}(\mathtt{IF\ X<Y\ THEN\ MAX:=Y\ ELSE\ MAX:=X},\ (\mathtt{X} = x \wedge \mathtt{Y} = y)) \\
= \ & \mathtt{sp}(\mathtt{MAX:=Y},\ ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y})) \\
& \vee \\
& \mathtt{sp}(\mathtt{MAX:=X},\ ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \neg(\mathtt{X} < \mathtt{Y}))) \\
= \ & \exists v.\ \mathtt{MAX} = \mathtt{Y}[v/\mathtt{MAX}] \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y})[v/\mathtt{MAX}] \\
& \vee \\
& \exists v.\ \mathtt{MAX} = \mathtt{X}[v/\mathtt{MAX}] \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \neg(\mathtt{X} < \mathtt{Y}))[v/\mathtt{MAX}] \\
= \ & \exists v.\ \mathtt{MAX} = \mathtt{Y} \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y}) \\
& \vee \\
& \exists v.\ \mathtt{MAX} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(\mathtt{X} < \mathtt{Y})) \\
= \ & (\mathtt{MAX} = \mathtt{Y} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \mathtt{X} < \mathtt{Y}) \vee (\mathtt{MAX} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(\mathtt{X} < \mathtt{Y}) \\
= \ & (\mathtt{MAX} = y \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge x < y) \vee (\mathtt{MAX} = x \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(x < y) \\
= \ & \mathit{if}\ x < y\ \mathit{then}\ (\mathtt{MAX} = y \wedge \mathtt{X} = x \wedge \mathtt{Y} = y)\ \mathit{else}\ (\mathtt{MAX} = x \wedge \mathtt{X} = x \wedge \mathtt{Y} = y) \\
= \ & \mathtt{MAX} = (\mathit{if}\ x < y\ \mathit{then}\ y\ \mathit{else}\ x) \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \\
= \ & \mathtt{MAX} = \mathit{max}(x, y) \wedge \mathtt{X} = x \wedge \mathtt{Y} = y
\end{aligned}
$$

Computing `sp` versus `wlp`

- Computing `sp` is like execution
  - can simplify as one goes along with the 'current state'
  - may be able to resolve branches, so can avoid executing them
  - Floyd assignment rule complicated in general
  - `sp` used for symbolically exploring 'reachable states'
    (related to *model checking*)
- Computing `wlp` is like backwards proof
  - don't have 'current state', so can't simplify using it
  - can't determine conditional tests, so get big `if-then-else` trees
  - Hoare assignment rule simpler for arbitrary formulae
  - `wlp` used for improved verification conditions

Using sp to generate verification conditions

- If C is loop-free then VC for $\{P\}\ C\ \{Q\}$ is $\mathtt{sp}(C,\ P) \Rightarrow Q$

- Cannot in general compute a finite formula for $\mathtt{sp}(\mathtt{WHILE}\ S\ \mathtt{DO}\ C,\ P)$

- The following holds

  $\mathtt{sp}(\mathtt{WHILE}\ S\ \mathtt{DO}\ C,\ P)\ = \mathtt{sp}(\mathtt{WHILE}\ S\ \mathtt{DO}\ C,\ \mathtt{sp}(C,\ (P \wedge S)))\ \vee\ (P \wedge \neg S)$

- Above doesn't define $\mathtt{sp}(C, P)$ to be a finite statement

- As with `wlp`, can use a hybrid VC and `sp` method

sp-based verification conditions

- Define $\texttt{asp}(C, P)$ to be an approximate strongest postcondition
- Define $\texttt{svc}(C, P)$ to be a set of verification conditions
- Idea is that if $\bigwedge \texttt{svc}(C, P) \Rightarrow \{P\}\ C\ \{\texttt{asp}(C, P)\}$
- If C is loop-free then
  - $\texttt{asp}(C, P) = \texttt{sp}(C, P)$
  - $\texttt{svc}(C, P) = \{\}$

Definition of asp

- Define asp recursively by:

$$
\begin{aligned}
\mathtt{asp}(P,\ V\ \mathtt{:=}\ E) &= \exists v.\ V = E[v/V] \wedge P[v/V] \\
\mathtt{asp}(P,\ C_1\ \mathtt{;}\ C_2) &= \mathtt{asp}(\mathtt{asp}(P, C_1), C_2) \\
\mathtt{asp}(P,\ \mathtt{IF}\ S\ \mathtt{THEN}\ C_1\ \mathtt{ELSE}\ C_2) &= \mathtt{asp}(P \wedge S,\ C_1)\ \vee\ \mathtt{asp}(P \wedge \neg S,\ C_2) \\
\mathtt{asp}(P,\ \mathtt{WHILE}\ S\ \mathtt{DO}\ \{R\}\ C) &= R \wedge \neg S
\end{aligned}
$$

Definition of svc

- Define svc recursively by:

$$
\begin{array}{ll}
\texttt{svc}(P,\ V\ \texttt{:=}\ E) & = \{\} \\
\texttt{svc}(P,\ C_1\ ;\ C_2) & = \texttt{svc}(P, C_1) \cup \texttt{svc}(\texttt{svc}_1(P, C_1), C_2) \\
\texttt{svc}(P,\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2) & = \texttt{svc}(P \wedge S,\ C_1)\ \cup\ \texttt{svc}(P \wedge \neg S,\ C_2) \\
\texttt{svc}(P,\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C) & = \{P \Rightarrow R,\ \texttt{asp}(R \wedge S,\ C) \Rightarrow R\} \\
 & \quad \cup\ \texttt{svc}(R \wedge S,\ C)
\end{array}
$$

- Theorem: $\bigwedge \texttt{svc}(P, C) \Rightarrow \{P\}\ C\ \{\texttt{asp}(P, C)\}$

- Proof by induction on C (exercise)

Summary

- Annotate then generate VCs is the classical method
  - practical tools:
  - weakest preconditions are alternative explanation of VCs
  - `wlp` needs fewer annotations than VC method described earlier
  - `wlp` also used for refinement

- VCs and `wlp` go backwards, `sp` goes forward
  - `sp` provides verification method based on symbolic simulation
  - widely used for loop-free code
  - current research potential for forwards full proof of correctness
  - probably need mixture of forwards and backwards methods (Hoare's view)

Range of methods for proving $\{P\}C\{Q\}$

- Bounded model checking (BMC)
  - unwind loops a finite number of times
  - then symbolically execute
  - check states reached satisfy decidable properties

- Full proof of correctness
  - add invariants to loops
  - generate verification conditions
  - prove verification conditions with a theorem prover

Total Correctness Specification

- So far our discussion has been concerned with partial correctness
  - what about termination
- A total correctness specification $[P]\ C\ [Q]$ is true if and only if
  - whenever C is executed in a state satisfying P,
    then the execution of C terminates
  - after C terminates Q holds
- Except for the `WHILE`-rule, all the axioms and rules described so far are sound for total correctness as well as partial correctness

Termination of WHILE-Commands

- WHILE-commands are the only commands that might not terminate

- Consider now the following proof

1. $\vdash$ {T} X := X {T} (assignment axiom)
2. $\vdash$ {T $\wedge$ T} X := X {T} (precondition strengthening)
3. $\vdash$ {T} WHILE T DO X := X {T $\wedge$ $\neg$T} (2 and the WHILE-rule)

- If the WHILE-rule worked for total correctness, then this would show:

$\vdash$ [T] WHILE T DO X := X [T $\wedge$ $\neg$T]

- Thus the WHILE-rule is unsound for total correctness

Rules for Non-Looping Commands

- Replace { and } by [and], respectively, in:
  - Assignment axiom (see next slide for discussion)
  - Consequence rules
  - Conditional rule
  - Sequencing rule
- The following is a valid derived rule

$$\frac{\vdash\ \{P\}\ C\ \{Q\}}{\vdash\ [P]\ C\ [Q]}$$

if C contains no `WHILE`-commands

Total Correctness Assignment Axiom

- Assignment axiom for total correctness

$$\vdash \ [P[E/V]] \ V := E \ [P]$$

- Note that the assignment axiom for total correctness states that assignment commands *always* terminate

- So all function applications in expressions must terminate

- This might not be the case if functions could be defined recursively

- Consider $X := \mathit{fact}(-1)$, where $\mathit{fact}(n)$ is defined recursively:

$$\mathit{fact}(n) \ = \ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times \mathit{fact}(n-1)$$

Error Termination

- We assume erroneous expressions like $1/0$ don't cause problems

- Most programming languages will raise an error on division by zero

- In our logic it follows that

$$\vdash\ [\mathtt{T}]\ \mathtt{X} := 1/0\ [\mathtt{X} = 1/0]$$

- The assignment $\mathtt{X} := 1/0$ halts in a state in which $\mathtt{X} = 1/0$ holds

- This assumes that $1/0$ denotes some value that $\mathtt{X}$ can have

Two Possibilities

- There are two possibilities

  (i) $1/0$ denotes some number;

  (ii) $1/0$ denotes some kind of 'error value'.

- It seems at first sight that adopting (ii) is the most natural choice
  - this makes it tricky to see what arithmetical laws should hold
  - is $(1/0) \times 0$ equal to 0 or to some 'error value'?
  - if the latter, then it is no longer the case that $\forall n.\ n \times 0 = 0$ is valid

- It is possible to make everything work with undefined and/or error values, but the resultant theory is a bit messy

Example

- We assume that arithmetic expressions *always* denote numbers

- In some cases exactly what the number is will be not fully specified

  - **for example, we will assume that m/n denotes a number for any m and n**

  - **only assume:** $\neg(n = 0) \quad \Rightarrow \quad (m/n) \times n = m$

  - **it is not possible to deduce anything about $m/0$ from this**

  - **in particular it is not possible to deduce that $(m/0) \times 0 = 0$**

  - **but $(m/0) \times 0 = 0$ does follow from $\forall n.\ n \times 0 = 0$**

- People still argue about this – e.g. advocate "three-valued" logics

WHILE-rule for Total Correctness (i)

- WHILE-commands are the only commands in our little language that can cause non-termination
  - they are thus the only kind of command with a non-trivial termination rule
- The idea behind the WHILE-rule for total correctness is
  - to prove WHILE S DO C terminates
  - show that some non-negative quantity decreases on each iteration of C
  - this decreasing quantity is called a variant

WHILE-Rule for Total Correctness (ii)

- In the rule below, the variant is E, and the fact that it decreases is specified with an auxiliary variable n

- The hypothesis $\vdash\ P \wedge S \Rightarrow E \geq 0$ ensures the variant is non-negative

WHILE-rule for total correctness

$$\frac{\vdash\ [P \wedge S \wedge (E = n)]\ C\ [P \wedge (E < n)], \quad \vdash\ P \wedge S \Rightarrow E \geq 0}{\vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ [P \wedge \neg S]}$$

where E is an integer-valued expression
and n is an identifier not occurring in P, C, S or E.

Example

- We show

$$\vdash\ [\texttt{Y} > 0]\ \texttt{WHILE Y}{\leq}\texttt{R DO (R:=R-Y; Q:=Q+1)}\ [\texttt{T}]$$

- Take

$$\begin{array}{lcl} P & = & \texttt{Y} > \texttt{0} \\ S & = & \texttt{Y} \leq \texttt{R} \\ E & = & \texttt{R} \\ C & = & \texttt{(R:=R-Y; Q:=Q+1)} \end{array}$$

- We want to show $\vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ [\texttt{T}]$

- By the WHILE-rule for total correctness it is sufficient to show

$$\begin{array}{rl} \text{(i)} & \vdash\ [P \land S \land (E = \texttt{n})]\ C\ [P \land (E < \texttt{n})] \\ \text{(ii)} & \vdash\ P \land S \Rightarrow E \geq 0 \end{array}$$

Example Continued (1)

- From previous slide:

$$\begin{aligned} P &= \mathtt{Y} > \mathtt{0} \\ S &= \mathtt{Y} \leq \mathtt{R} \\ E &= \mathtt{R} \\ C &= \mathtt{(R:=R-Y;\ Q:=Q+1)} \end{aligned}$$

- We want to show

$$\begin{aligned} \text{(i)} &\vdash [P \wedge S \wedge (E = \mathtt{n})]\ C\ [P \wedge (E < \mathtt{n})] \\ \text{(ii)} &\vdash P \wedge S \Rightarrow E \geq 0 \end{aligned}$$

- The first of these, (i), can be proved by establishing

$$\vdash \{P \wedge S \wedge (E = \mathtt{n})\}\ C\ \{P \wedge (E < \mathtt{n})\}$$

- Then using the total correctness rule for non-looping commands

Example Continued (2)

- From previous slide:

$$
\begin{aligned}
P &= \texttt{Y} > \texttt{0} \\
S &= \texttt{Y} \leq \texttt{R} \\
E &= \texttt{R} \\
C &= \texttt{R:=R-Y; Q:=Q+1)}
\end{aligned}
$$

- The verification condition for $\{P \wedge S \wedge (E = \mathtt{n})\}\ C\ \{P \wedge (E < \mathtt{n})\}$ is:

$$
\begin{aligned}
&\mathtt{Y} > 0 \ \wedge\ \mathtt{Y} \leq \mathtt{R} \ \wedge\ \mathtt{R} = \mathtt{n} \quad \Rightarrow \\
&\quad (\mathtt{Y} > 0 \ \wedge\ \mathtt{R} < \mathtt{n})[\mathtt{Q}{+}1/\mathtt{Q}][\mathtt{R}{-}\mathtt{Y}/\mathtt{R}]
\end{aligned}
$$

  i.e. $\mathtt{Y} > 0 \ \wedge\ \mathtt{Y} \leq \mathtt{R} \ \wedge\ \mathtt{R} = \mathtt{n} \quad \Rightarrow \quad \mathtt{Y} > 0 \ \wedge\ \mathtt{R}{-}\mathtt{Y} < \mathtt{n}$

  which follows from the laws of arithmetic

- The second subgoal, (ii), is just $\vdash\ \mathtt{Y} > 0 \ \wedge\ \mathtt{Y} \leq \mathtt{R} \Rightarrow \mathtt{R} \geq 0$

Termination Specifications

- The relation between partial and total correctness is informally given by the equation

$$\textit{Total correctness} = \textit{Termination} + \textit{Partial correctness}$$

- This informal equation can be represented by the following two rules of inferences

$$\frac{\vdash\ \{P\}\ C\ \{Q\} \qquad \vdash\ [P]\ C\ [\mathrm{T}]}{\vdash\ [P]\ C\ [Q]}$$

$$\frac{\vdash\ [P]\ C\ [Q]}{\vdash\ \{P\}\ C\ \{Q\} \qquad \vdash\ [P]\ C\ [\mathrm{T}]}$$

Derived Rules

- Multiple step rules for total correctness can be derived in the same way as for partial correctness
  - the rules are the same up to the brackets used
  - same derivations with total correctness rules replacing partial correctness ones

The Derived While Rule

- Derived `WHILE`-rule needs to handle the variant

Derived `WHILE`-rule for total correctness

$$\frac{\begin{array}{l}\vdash\ P \Rightarrow R \\ \vdash\ R \wedge S \Rightarrow E \geq 0 \\ \vdash\ R \wedge \neg S\ \Rightarrow Q \\ \vdash\ [R \wedge S \wedge (E = n)]\ C\ [R \wedge (E < n)]\end{array}}{\vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ [Q]}$$

VCs for Termination

- Verification conditions are easily extended to total correctness
- To generate total correctness verification conditions for `WHILE`-commands, it is necessary to add a variant as an annotation in addition to an invariant
- Variant added directly after the invariant, in square brackets
- No other extra annotations are needed for total correctness
- VCs for `WHILE`-free code same as for partial correctness

WHILE Annotation

- A correctly annotated total correctness specification of a WHILE-command thus has the form

$$[P]\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}[E]\ C\ [Q]$$

  where R is the invariant and E the variant

- Note that the variant is intended to be a non-negative expression that decreases each time around the WHILE loop

- The other annotations, which are enclosed in curly brackets, are meant to be conditions that are true whenever control reaches them (as before)

WHILE VCs

- A correctly annotated specification of a WHILE-command has the form

$$[P]\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}[E]\ C\ [Q]$$

WHILE-commands

The verification conditions generated from

$$[P]\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}[E]\ C\ [Q]$$

are

(i) $P \Rightarrow R$

(ii) $R \wedge \neg S \Rightarrow Q$

(iii) $R \wedge S \Rightarrow E \geq 0$

(iv) the verification conditions generated by

$$[R \wedge S \wedge (E = n)]\ C[R \wedge (E < n)]$$

where n is a variable not occurring in P, R, E, C, S or Q.

Example

- The verification conditions for

```
[R=X ∧ Q=0]
 WHILE Y≤R DO {X=R+Y×Q}[R]
   (R:=R-Y; Q=Q+1)
[X = R+(Y×Q) ∧ R<Y]
```

are:

(i) `R=X ∧ Q=0 ⇒ (X = R+(Y×Q))`

(ii) `X = R+Y×Q ∧ ¬(Y≤R) ⇒ (X = R+(Y×Q) ∧ R<Y)`

(iii) `X = R+Y×Q ∧ Y≤R ⇒ R≥0`

together with the verification condition for

```
[X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n)]
 (R:=R-Y; Q:=Q+1)
[X=R+(Y×Q) ∧ (R<n)]
```

Example Continued

- The single verification condition for

```
[X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n)]
 (R:=R-Y; Q:=Q+1)
[X=R+(Y×Q) ∧ (R<n)]
```

  is

```
(iv) X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n) ⇒
     X = (R-Y)+(Y×(Q+1)) ∧ ((R-Y)<n)
```

- But this isn't true
  - take `Y=0`
- To prove `R-Y<n` we need to know `Y>0`
- *Exercise:* Explain why one would not expect to be able to prove the verification conditions of this last example
- *Hint:* Consider the original specification

Summary

- We have given rules for total correctness
- They are similar to those for partial correctness
- The main difference is in the `WHILE`-rule
  - because `WHILE` commands are the only ones that can fail to terminate
- Must prove a non-negative expression is decreased by the loop body
- Derived rules and VC generation rules for partial correctness easily extended to total correctness