A Logic Laws

Name	Description
Simplify	$p \wedge q \Rightarrow p, q$
Modus Ponens	$(p \rightarrow q), p \Rightarrow q$
Conjunction	$p, q \Rightarrow p \wedge q$
Disjunction	$p \Rightarrow p \vee q, q \vee p$
Definition of Conditional	$p \rightarrow q \Leftrightarrow \neg p \vee q$
Definition of Biconditional	$p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$
Law of the Excluded Middle (LEM)	$p \vee \neg p \Leftrightarrow T$
Double Negation Elimination (DNE)	$p \Leftrightarrow \neg\neg p$
Contradiction	$p \wedge \neg p \Leftrightarrow F$
Identity	$p \wedge T \Rightarrow p$, $p \vee F \Rightarrow p$
DeMorgan's Laws	$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$ $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ $\neg(\forall x.p(x)) \Leftrightarrow \exists x.\neg p(x)$ $\neg(\exists x.p(x)) \Leftrightarrow \forall x.\neg p(x)$
Distributivity	$(p \wedge q) \vee r \Leftrightarrow (p \vee r) \wedge (q \vee r)$ $(p \vee q) \wedge r \Leftrightarrow (p \wedge r) \vee (q \wedge r)$
Commutativity	$p \wedge q \Leftrightarrow q \wedge p$, $p \vee q \Leftrightarrow q \vee p$
Associativity	$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$, $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
Idempotency	$p \wedge p \Leftrightarrow p$, $p \vee p \Leftrightarrow p$
Domination	$p \vee T \Leftrightarrow T$, $p \wedge F \Leftrightarrow F$

B Language Syntax and Semantics

Expression and Statement Syntax

$$\begin{array}{rcl}
e & ::= & \overline{n} \mid \mathsf{true} \mid \mathsf{false} \mid x \mid a[e] \mid e\ op\ e \mid e\,?\,e : e \mid size(a) \\
s & ::= & \mathsf{skip} \mid s; s \mid x := e \mid a[e] := e \mid \mathsf{if}\ e\ \mathsf{then}\ \{s\}\ \mathsf{else}\ \{s\} \mid \mathsf{while}\ e\ \{s\} \\
 & & \mid \mathsf{havoc}\ x \mid \mathsf{branch}\ \{e \rightarrow s \,\square \cdots \square\, e \rightarrow s\} \mid \mathsf{while}\ \{e \rightarrow s \,\square \cdots \square\, e \rightarrow s\} \\
 & & \mid\ [s \| \ldots \| s] \mid < s > \mid \mathsf{await}\ e\ \mathsf{then}\ \{s\}
\end{array}$$

Expression Semantics

$$\begin{array}{rcll}
\sigma(\overline{n}) & = & n & \\
\sigma(\mathsf{true}) & = & T & \\
\sigma(\mathsf{false}) & = & F & \\
\sigma(x) & = & \sigma(x) & \\
\sigma(a[e]) & = & (\sigma(a))[\sigma(e)] & \sigma(e) \neq \bot_e \wedge 0 \leq \sigma(e) < |\sigma(a)| \\
\sigma(a[e]) & = & \bot_e & \text{otherwise} \\
\sigma(e_1\ op\ e_2) & = & \sigma(e_1)\ op\ \sigma(e_2) & \sigma(e_1) \neq \bot_e \neq \sigma(e_2) \\
\sigma(e_1\ op\ e_2) & = & \bot_e & \sigma(e_1) = \bot_e \vee \sigma(e_2) = \bot_e \\
\sigma(e_1\,?\,e_2 : e_3) & = & \sigma(e_2) & \sigma(e_1) = T \\
\sigma(e_1\,?\,e_2 : e_3) & = & \sigma(e_3) & \sigma(e_1) = F \\
\sigma(e_1\,?\,e_2 : e_3) & = & \bot_e & \sigma(e_1) = \bot_e \\
\sigma(size(a)) & = & |\sigma(a)| &
\end{array}$$

Statement Semantics - Big-step

$$\begin{array}{rcll}
M(\mathsf{skip}, \sigma) & = & \{\sigma\} & \\
M(s_1; s_2, \sigma) & = & \bigcup_{\sigma' \in M(s_1,\sigma)} M(s_2, \sigma') & \\
M(x := e, \sigma) & = & \{\sigma[x \mapsto \sigma(e)]\} & \sigma(e) \neq \bot_e \\
M(x := e, \sigma) & = & \{\bot_e\} & \sigma(e) = \bot_e \\
M(a[e_1] := e_2, \sigma) & = & \{\sigma[a[\sigma(e_1)] \mapsto \sigma(e_2)]\} & \sigma(e_1) \neq \bot_e \wedge \sigma(e_2) \neq \bot_e \\
 & & & \wedge\ 0 \leq \sigma(e_1) < |\sigma(a)| \\
M(a[e_1] := e_2, \sigma) & = & \{\bot_e\} & \text{otherwise} \\
M(\mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma) & = & M(s_1, \sigma) & \sigma(e) = T \\
M(\mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma) & = & M(s_2, \sigma) & \sigma(e) = F \\
M(\mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma) & = & \{\bot_e\} & \sigma(e) = \bot_e \\
M(\mathsf{branch}\ \{e_1 \rightarrow s_1 \,\square \cdots \square\, e_n \rightarrow s_n\}, \sigma) & = & \{\bot_e\} & \sigma(e_i) = \bot_e \\
M(\mathsf{branch}\ \{e_1 \rightarrow s_1 \,\square \cdots \square\, e_n \rightarrow s_n\}, \sigma) & = & \{\bot_e\} & \forall i. \sigma(e_i) = F \\
M(\mathsf{branch}\ \{e_1 \rightarrow s_1 \,\square \cdots \square\, e_n \rightarrow s_n\}, \sigma) & = & \bigcup_{i \in [1,n], \sigma(e_i) = T} M(s_i, \sigma) & \\
M(\mathsf{havoc}\ x, \sigma) & = & \{\sigma[x \mapsto n] \mid n \in \mathbb{Z}\} & \\
M(\mathsf{while}\ e\ \{s\}, \sigma) & = & \Sigma_k & \Sigma_k \text{ is the lowest } k \text{ such that if} \\
 & & & \sigma \in \Sigma_k \text{ , then } \sigma(e) = F \\
M(\mathsf{while}\ e\ \{s\}, \sigma) & = & \{\bot_d\} & \text{no such } k \text{ exists}
\end{array}$$

where

$$\begin{array}{rcl}
\Sigma_0 & = & \{\sigma\} \\
\Sigma_k + 1 & = & \bigcup_{\sigma \in \Sigma_k} M(s, \sigma)
\end{array}$$

Statement Semantics - Small-step

$$\frac{\langle s_1, \sigma\rangle \to \langle s_1', \sigma\rangle}{\langle s_1; s_2, \sigma\rangle \to \langle s_1'; s_2, \sigma\rangle} \qquad \frac{\langle s_1, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}{\langle s_1; s_2, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle} \qquad \frac{}{\langle \mathsf{skip}; s, \sigma\rangle \to \langle s, \sigma\rangle}$$

$$\frac{\sigma(e) \neq \bot_e}{\langle x := e, \sigma\rangle \to \langle \mathsf{skip}, \sigma[x \mapsto \sigma(e)]\rangle} \qquad \frac{\sigma(e) = \bot_e}{\langle x := e, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}$$

$$\frac{\sigma(e_1) \neq \bot_e \qquad \sigma(e_2) \neq \bot_e \qquad 0 \leq \sigma(e_1) < |\sigma(a)|}{\langle a[e_1] := e_2, \sigma\rangle \to \langle \mathsf{skip}, \sigma[a[\sigma(e_1)] \mapsto \sigma(e_2)]\rangle} \qquad \frac{\sigma(e_1) = \bot_e \vee \sigma(e_2) = \bot_e}{\langle a[e_1] := e_2, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}$$

$$\frac{\sigma(e_1) \geq |\sigma(a)| \vee \sigma(e_1) < 0}{\langle a[e_1] := e_2, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle} \qquad \frac{\sigma(e) = T}{\langle \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma\rangle \to \langle s_1, \sigma\rangle}$$

$$\frac{\sigma(e) = F}{\langle \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma\rangle \to \langle s_2, \sigma\rangle} \qquad \frac{\sigma(e) = \bot_e}{\langle \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}$$

$$\frac{}{\langle \mathsf{while}\ e\ \{s\}, \sigma\rangle \to \langle \mathsf{if}\ e\ \mathsf{then}\ \{s; \mathsf{while}\ e\ \{s\}\}\ \mathsf{else}\ \{\mathsf{skip}\}, \sigma\rangle}$$

$$\frac{\exists i \in [1,n].\sigma(e_i) = \bot_e}{\langle \mathsf{branch}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}$$

$$\frac{\sigma(e_1) = \cdots = \sigma(e_n) = F}{\langle \mathsf{branch}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle}$$

$$\frac{\sigma(e_i) = T}{\langle \mathsf{branch}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle s_i, \sigma\rangle} \qquad \frac{n \in \mathbb{Z}}{\langle \mathsf{havoc}\ x, \sigma\rangle \to \langle \mathsf{skip}, \sigma[x \mapsto n]\rangle}$$

$$\frac{\exists i \in [1,n].\sigma(e_i) = \bot_e}{\langle \mathsf{while}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle \mathsf{skip}, \bot_e\rangle} \qquad \frac{\sigma(e_1) = \cdots = \sigma(e_n) = F}{\langle \mathsf{while}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle \mathsf{skip}, \sigma\rangle}$$

$$\frac{\sigma(e_i) = T}{\langle \mathsf{while}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle \to \langle s_i; \mathsf{while}\ \{e_1 \to s_1 \,\Box \cdots \Box\, e_n \to s_n\}, \sigma\rangle}$$

$$\frac{\langle s_i, \sigma\rangle \to \langle s_i', \sigma'\rangle}{\langle [s_1 \| \ldots \| s_i \| \ldots s_n], \sigma\rangle \to \langle [s_1 \| \ldots \| s_i' \| \ldots s_n], \sigma'\rangle} \qquad \frac{\langle s, \sigma\rangle \to^* \langle \mathsf{skip}, \sigma'\rangle}{\langle < s >, \sigma\rangle \to \langle \mathsf{skip}, \sigma'\rangle}$$

$$\frac{\sigma(e) = T \qquad \langle s, \sigma\rangle \to^* \langle \mathsf{skip}, \sigma'\rangle}{\langle \mathsf{await}\ e\ \mathsf{then}\ \{s\}, \sigma\rangle \to \langle \mathsf{skip}, \sigma'\rangle}$$

C Hoare Triple Inference Rules

$$\frac{}{\vdash \{p\}\ \mathsf{skip}\ \{p\}}\ (\textsc{Skip}) \qquad \frac{}{\vdash \{[e/x]p\}\ x := e\ \{p\}}\ (\textsc{AssignBwd})$$

$$\frac{}{\vdash \{[e/a[i]]p\}\ a[i] := e\ \{p\}}\ (\textsc{ArrAssignBwd})$$

$$\frac{x_0\ \mathsf{fresh}}{\vdash \{p\}\ x := e\ \{[x_0/x]p \land x = [x_0/x]e\}}\ (\textsc{AssignFwd}) \qquad \frac{\vdash \{p\}\ s_1\ \{q'\} \qquad \vdash \{q'\}\ s_2\ \{q\}}{\vdash \{p\}\ s_1; s_2\ \{q\}}\ (\textsc{Seq})$$

$$\frac{\vdash \{p \land e\}\ s_1\ \{q_1\} \qquad \vdash \{p \land \neg e\}\ s_2\ \{q_2\}}{\vdash \{p\}\ \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}\ \{q_1 \lor q_2\}}\ (\textsc{If}) \qquad \frac{\vdash \{p \land e\}\ s_1\ \{q\} \qquad \vdash \{p \land \neg e\}\ s_2\ \{q\}}{\vdash \{p\}\ \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}\ \{q\}}\ (\textsc{If'})$$

$$\frac{\vdash \{p \land e\}\ s\ \{p\}}{\vdash \{p\}\ \mathsf{while}\ e\ \{s\}\ \{p \land \neg e\}}\ (\textsc{While}) \qquad \frac{\vdash [\ p \land e \land t = t_0\ ]\ s\ [\ p \land t < t_0\ ]}{\vdash [\ p\ ]\ \{\mathbf{dec}\ t\}\ \mathsf{while}\ e\ \{s\}\ [\ p \land \neg e\ ]}\ (\textsc{WhileBound})$$

$$\frac{\vdash \{p \land e_i\}\ s_i\ \{q\}}{\vdash \{p\}\ \mathsf{branch}\ \{e_1 \rightarrow s_1 \,\Box \cdots \Box\, e_n \rightarrow s_n\}\ \{q\}}\ (\textsc{NDBranch})$$

$$\frac{}{\vdash \{\forall x_0 \in \mathbb{Z}.[x_0/x]q\}\ \mathsf{havoc}\ x\ \{q\}}\ (\textsc{HavocBwd}) \qquad \frac{x_0\ \mathsf{fresh}}{\vdash \{p\}\ \mathsf{havoc}\ x\ \{[x_0/x]p\}}\ (\textsc{HavocFwd})$$

$$\frac{\vdash \{p\}\ s\ \{q\} \qquad p' \Rightarrow p \qquad q \Rightarrow q'}{\vdash \{p'\}\ s\ \{q'\}}\ (\textsc{Weaken})$$

$$\frac{\vdash \{p\}\ s_1; \ldots; s_n\ \{q\} \qquad s_1, \ldots, s_n \text{ are pairwise disjoint programs}}{\vdash \{p\}\ [s_1 \| \ldots \| s_n]\ \{q\}}\ (\textsc{Sequentialize})$$

$$\frac{\vdash \{p_i\}\ s_i\ \{q_i\} \qquad s_1, \ldots, s_n \text{ pairwise disjoint w/ pairwise disjoint conditions}}{\vdash \{p_1 \land \cdots \land p_n\}\ [s_1 \| \ldots \| s_n]\ \{q_1 \land \ldots q_n\}}\ (\textsc{Par})$$

$$\frac{\vdash \{p_i\}\ s_i\ \{q_i\} \qquad \text{The } \{p_i\}\ s_i^*\ \{q_i\} \text{ are pairwise interference-free}}{\vdash \{p_1 \land \cdots \land p_n\}\ [s_1 \| \ldots \| s_n]\ \{q_1 \land \ldots q_n\}}\ (\textsc{Par-OG})$$

D Simplifying Conditional Expressions

$$\begin{array}{rcll} T\,?\,e_1 : e_2 & \Rrightarrow & e_1 & \text{Always} \\ e_0\,?\,e : e & \Rrightarrow & e & \text{Always} \\ e_0\,?\,e_1 : e_2 & \Rrightarrow & e_2 & \text{If } e_0 \Rightarrow e_1 = e_2 \end{array} \qquad \begin{array}{rcll} F\,?\,e_1 : e_2 & \Rrightarrow & e_2 & \text{Always} \\ & & & \\ e_0\,?\,e_1 : e_2 & \Rrightarrow & e_1 & \text{If } \neg e_0 \Rightarrow e_1 = e_2 \end{array}$$

Let Θ be a unary operator, $\oplus$ be a binary operator or relation and f be any function.

$$\begin{array}{rcl} \Theta(e\,?\,e_1 : e_2) & \Rrightarrow & e\,?\,\Theta(e_1) : \Theta(e_2) \\ a[e\,?\,e_1 : e_2] & \Rrightarrow & e\,?\,a[e_1] : a[e_2] \end{array} \qquad \begin{array}{rcl} (e\,?\,e_1 : e_2) \oplus e_3 & \Rrightarrow & e\,?\,e_1 \oplus e_3 : e_2 \oplus e_3 \\ f(e\,?\,e_1 : e_2) & \Rrightarrow & e\,?\,f(e_1) : f(e_2) \end{array}$$

If e, e_1, and e_2 are Boolean expressions, then

$$\begin{array}{rcl} (e\,?\,e_1 : F) & \Leftrightarrow & (e \wedge e_1) \\ (e\,?\,e_1 : T) & \Leftrightarrow & (e \to e_1) \;\Leftrightarrow\; (\neg e \vee e_1) \end{array} \qquad \begin{array}{rcl} (e\,?\,F : e_2) & \Leftrightarrow & (\neg e \wedge e_2) \\ (e\,?\,T : e_2) & \Leftrightarrow & (\neg e \to e_2) \;\Leftrightarrow\; (e \vee e_2) \end{array}$$

$$(e\,?\,e_1 : e_2) \Leftrightarrow ((e \to e_1) \wedge (\neg e \to e_2)) \Leftrightarrow ((e \wedge e_1) \vee (\neg e \wedge e_2))$$

E Calculating wp, wlp, sp for loop-free programs

$\oplus$ stands for any binary operator that doesn't itself cause errors, e.g., +, -...

$$\begin{array}{rcl} wlp(\mathsf{skip}, q) & = & q \\ wlp(a[e_1] := e_2, q) & = & [e_2/a[e_1]]q \\ wlp(\mathsf{havoc}\ x, q) & = & \forall x_0.[x_0/x]q \end{array} \qquad \begin{array}{rcl} wlp(x := e, q) & = & [e/x]q \\ wlp(s_1; s_2, q) & = & wlp(s_1, wlp(s_2, q)) \end{array}$$

$$\begin{array}{rcl} wlp(\mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}, q) & = & (e \to wlp(s_1, q)) \wedge (\neg e \to wlp(s_2, q)) \\ wlp(\mathsf{branch}\ \{e_1 \to s_1 \,\square \cdots \square\, e_n \to s_n\}, q) & = & (e_1 \to wlp(s_1, q)) \wedge \cdots \wedge (e_n \to wlp(s_n, q)) \end{array}$$

$$\begin{array}{rcl} sp(p, \mathsf{skip}) & = & p \\ sp(p, s_1; s_2) & = & sp(sp(p, s_1), s_2) \end{array} \qquad \begin{array}{rcl} sp(p, x := e) & = & [x_0/x]p \wedge x = [x_0/x]e \\ sp(p, \mathsf{havoc}\ x) & = & [x_0/x]p \end{array}$$

$$\begin{array}{rcl} sp(p, \mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}) & = & sp(p \wedge e, s_1) \vee sp(p \wedge \neg e, s_2) \\ sp(p, \mathsf{branch}\ \{e_1 \to s_1 \,\square \cdots \square\, e_n \to s_n\}) & = & sp(p \wedge e_1, s_1) \vee \cdots \vee sp(p \wedge e_n, s_n) \end{array}$$

$$\begin{array}{rcl} D(c) & = & T \\ D(a[e]) & = & D(e) \wedge 0 \le e < |a| \\ D(sqrt(e)) & = & D(e) \wedge e \ge 0 \end{array} \qquad \begin{array}{rcl} D(x) & = & T \\ D(e_1/e_2) & = & D(e_1) \wedge D(e_2) \wedge e_2 \ne 0 \\ D(e_1 \oplus e_2) & = & D(e_1) \wedge D(e_2) \end{array}$$

$$D(e_1\,?\,e_2 : e_3) = D(e_1) \wedge (e_1 \to D(e_2)) \wedge (\neg e_1 \to D(e_3))$$

$$\begin{array}{rcl} D(\mathsf{skip}) & = & T \\ D(a[e_1] := e_2) & = & D(a[e_1]) \wedge D(e_2) \\ D(\mathsf{while}\ e\ \{s\}) & = & D(e) \wedge (e \to D(s)) \end{array} \qquad \begin{array}{rcl} D(x := e) & = & D(e) \\ D(s_1; s_2) & = & D(s_1) \wedge wp(s_1, D(s_2)) \\ D(\mathsf{havoc}\ x) & = & T \end{array}$$

$$\begin{array}{rcl} D(\mathsf{if}\ e\ \mathsf{then}\ \{s_1\}\ \mathsf{else}\ \{s_2\}) & = & D(e) \wedge (e \to D(s_1)) \wedge (\neg e \to D(s_2)) \\ D(\mathsf{branch}\ \{e_1 \to s_1 \,\square \cdots \square\, e_n \to s_n\}) & = & D(e_1) \wedge \cdots \wedge D(e_n) \wedge (e_1 \vee \cdots \vee e_n) \\ & & \wedge\ (e_1 \to D(s_1)) \wedge \ldots (e_n \to D(s_n)) \end{array}$$

$$wp(s, q) = wlp(s, q) \wedge D(s)$$

F Algorithm For Expanding Proof Outlines

A. Add a precondition:

(a) Prepend $\{wp(x := e, q)\}$ to $x := e\ \{q\}$.

(b) Prepend $\{q\}$ to $\mathsf{skip}\ \{q\}$.

(c) Prepend some p to s_2 in $s_1; s_2\ \{q\}$ to get $s_1; \{p\}\ s_2\ \{q\}$.

(d) Add preconditions to the branches of an if-else:
Turn $\{p\}$ if e then $\{s_1\}$ else $\{s_2\}$ into $\{p\}$ if e then $\{\{p \wedge e\}\ s_1\}$ else $\{\{p \wedge \neg e\}\ s_2\}$.

(e) Add a precondition to an if-else:
Prepend $\{(e \rightarrow p_1) \wedge (\neg e \rightarrow p_2)\}$ to if e then $\{\{p_1\}\ s_1\}$ else $\{\{p_2\}\ s_2\}$.

B. Or add a postcondition:

(a) Append $\{sp(p, x := e)\}$ to $\{p\}\ x := e$.

(b) Append $\{p\}$ to $\{p\}$ skip.

(c) Append some q to s_1 in $\{p\}\ s_1; s_2$ to get $\{p\}\ s_1\ \{q\}; s_2$.

(d) Add postconditions to the branches of an if-else:
Turn if e then $\{s_1\}$ else $\{s_2\}$ $\{q_1 \vee q_2\}$ into if e then $\{s_1\ \{q_1\}\}$ else $\{s_2\ \{q_2\}\}$ $\{q_1 \vee q_2\}$
or if e then $\{s_1\}$ else $\{s_2\}$ $\{q\}$ into if e then $\{s_1\ \{q\}\}$ else $\{s_2\ \{q\}\}$ $\{q\}$

(e) Add a postcondition to an if-else:
Append $\{q_1 \vee q_2\}$ to if e then $\{s_1\ \{q_1\}\}$ else $\{s_2\ \{q_2\}\}$.

C. Or add loop conditions:

(a) Take a loop and add pre- and post-conditions to the loop body; add a postcondition for the loop:
Turn $\{\mathbf{inv}\ p\}$ while e $\{s\}$ into:
$\{\mathbf{inv}\ p\}$ while e $\{\{p \wedge e\}\ s\ \{p\}\}$ $\{p \wedge \neg e\}$

D. Or strengthen or weaken some condition:

(a) Turn $\{q\}$ into $\{p\} \Rightarrow \{q\}$ for some p where $p \Rightarrow q$.

(b) Turn $\{p\}$ into $\{p\} \Rightarrow \{q\}$ for some q where $p \Rightarrow q$.

G Substitution for Array Elements

If $a \neq b$: $[e/a[e_1]](b[e_2]) = b[e_2']$ where $e_2' = [e/a[e_1]]e_2$.
General case: $[e_0/a[e_1]](a[e_2]) = (e_2' = e_1\ ?\ e_0 : a[e_2'])$ where $e_2' = [e_0/a[e_1]](e_2)$.
Special case where k is constant or variable: $[e/a[e_1]](a[k]) = (k = e_1\ ?\ e : a[k])$.