Hoare Logic

Specifications of the form $\{P\}C\{Q\}$, $[P]C[Q]$

Rule	Partial Hoare Logic
Assignment	$\vdash \{P[E/V]\}V := E\{P\}$
Precondition Strengthening	$\dfrac{\vdash P \Rightarrow P', \quad \vdash \{P'\}C\{Q\}}{\vdash \{P\}C\{Q\}}$
Postcondition Weakening	$\dfrac{\vdash \{P\}C\{Q\}, \quad \vdash Q' \Rightarrow Q}{\vdash \{P\}C\{Q\}}$
Conjunction	$\dfrac{\vdash \{P_1\}C\{Q_1\}, \quad \vdash \{P_2\}C\{Q_2\}}{\vdash \{P_1 \wedge P_2\}C\{Q_1 \wedge Q_2\}}$
Disjunction	$\dfrac{\vdash \{P_1\}C\{Q_1\}, \quad \vdash \{P_2\}C\{Q_2\}}{\vdash \{P_1 \vee P_2\}C\{Q_1 \vee Q_2\}}$
Sequencing	$\dfrac{\vdash \{P\}C_1\{Q\}, \quad \vdash \{Q\}C_2\{R\}}{\vdash \{P\}C_1; C_2\{R\}}$
Blocks	$\dfrac{\vdash \{P\}C\{Q\}}{\vdash \{P\}\texttt{BEGIN VAR } V_1\texttt{; } \ldots\texttt{; VAR } V_n\texttt{; } C \texttt{ END}\{Q\}}$
Single If	$\dfrac{\vdash \{P \wedge S\}C\{Q\}, \quad \vdash P \wedge \neg S \Rightarrow Q}{\vdash \texttt{IF } S \texttt{ THEN } C\{Q\}}$
Double If	$\dfrac{\vdash \{P \wedge S\}C_1\{Q\}, \quad \{P \wedge \neg S\}C_2\{Q\}}{\vdash \{P\}\texttt{IF } S \texttt{ THEN } C_1 \texttt{ ELSE } C_2\{Q\}}$
While	$\dfrac{\vdash \{P \wedge S\}C\{P\}}{\vdash \{P\}\texttt{WHILE } S \texttt{ DO } C\{P \wedge \neg S\}}$
For Rule	$\dfrac{\vdash \{P \wedge (E_1 \leq V) \wedge (V \leq E_2)\}C\{P[V+1/V]\}}{\vdash \{P[E_1/V] \wedge (E_1 \leq E_2)\}\texttt{FOR } V := E_1 \texttt{ UNTIL } E_2 \texttt{ DO } C\{P[E_2+1/V]\}}$
For Axiom	$\vdash \{P \wedge (E_2 < E_1)\}\texttt{FOR } V := E_1 \texttt{ UNTIL } E_2 \texttt{ DO } C\{P\}$
Array Assignment	$\vdash \{P[A\{E_1 \leftarrow E_2\}/A]\}A(E_1) := E_2\{P\}$

Reasoning About Arrays

$\vdash A\{E_1 \leftarrow E_2\}(E_1) = E_2$

$E_1 \neq E_3 \Rightarrow \vdash A\{E_1 \leftarrow E_2\}(E_3) = A(E_3)$

Termination

If we assume that all functions in expressions terminate then total correctness is as partial correctness except for the `WHILE` rule. In this case:

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

Where E is an integer-valued expression and n is an auxilliary variable not occuring in P, C, S and E.

Verification Conditions

We require annotation in the following places:

1. Before each command C_i in a sequence $C_1; C_2; \ldots; C_n$ which is not an assigment command
2. After the word `DO` in `WHILE` and `FOR` commands

Rule	VCs
Assignment	$P \Rightarrow Q[E/V]$
Sequencing	$[\{P\}C_1; \ldots ; C_{n-1}\{R\}]_{VC}$, $[\{R\}C_n\{Q\}]_{VC}$ (for assignments, $R = Q[E/V]$)
Blocks	$[\{P\}C\{Q\}]_{VC}$ (note block and P, Q variables disjoint)
Single If	$(P \wedge S) \Rightarrow Q$, $[\{P \wedge S\}C\{Q\}]_{VC}$
Double If	$[\{P \wedge S\}C_1\{Q\}]_{VC}$, $[\{P \wedge \neg S\}C_2\{Q\}]_{VC}$
While	$P \Rightarrow R$, $R \wedge \neg S \Rightarrow Q$,$[\{R \wedge S\}C\{R\}]_{VC}$
For Rule	$P \Rightarrow R[E_1/V]$ $R[E_2 + 1/V] \Rightarrow Q$ $[\{R \wedge E_1 \leq V \wedge V \leq E_2\}C\{R[V + 1/V]\}]_{VC}$, (with usual conditions on loop variables)
For Axiom	$P \wedge E_2 < E_1 \Rightarrow Q$
Array Assignment	$P \Rightarrow Q[A(E_1 \leftarrow E_2)/A]$

Termination

Most rules are unchanged, but the `WHILE` rule has these other verification conditions:

- $P \Rightarrow R$
- $R \wedge \neg S \Rightarrow Q$
- $R \wedge S \Rightarrow E \geq 0$
- $[[R \wedge S \wedge (E = n)]C[R \wedge (E < n)]]_{VC}$

Program Refinement

$[P, Q] = \{C \mid \vdash [P]C[Q]\}$

This is added to the syntax of a programming language to give a *wide spectrum* language. However, such programs are not directly executable. Note that in such a language strictly every command should be represented as a single element set.

Rule	Refinement
Skip	$[P, P] \supseteq \{SKIP\}$
Assignment	$[P[E/V], P] \supseteq \{V := E\}$
Precondition Weakening	$[P, Q] \supseteq [R, Q]$ if $\vdash P \Rightarrow R$
Postcondition Strengthening	$[P, Q] \supseteq [P, R]$ if $\vdash R \Rightarrow Q$
Sequencing	$[P, Q] \supseteq [P, R]; [R, Q]$
Blocks	$[P, Q] \supseteq$ `BEGIN VAR V;` $[P, Q]$ `END`
Single If	$[P, Q] \supseteq$ `IF` S `THEN` $[P \wedge S, Q]$ if $\vdash P \wedge \neg S \Rightarrow Q$
Double If	$[P, Q] \supseteq$ `IF` S `THEN` $[P \wedge S, Q]$ `ELSE` $[P \wedge \neg S, Q]$
While	$[P, P \wedge \neg S] \supseteq$ `WHILE` S `DO` $[P \wedge S \wedge (E = n), P \wedge (E < n)]$ if $\vdash P \wedge S \Rightarrow E \geq 0$

Higher Order Logic

Types are expressions that denote sets of values. Terms of HOL must be *well-typed* in that a type assignment to subterms exists. All binding is done via λ abstractions only, but syntactic sugar is provided for the common cases. Tuples are iterated pairs and may contain hetrogenous types.

Conditionals are represented as $t \rightarrow (t_1|t_2)$. Hilberts ϵ-operator lets you refer to values you know exist but aren't able to write down. It is defined by $\vdash \forall P\, x.P\, x \Rightarrow P(\epsilon\, P)$. Note that if the variable f does not occur in t then $\lambda x.t$ is equivalent to $\epsilon f.\forall x.f(x) = t$. This can be used to construct pattern matching functions and recursive functions.

Peano's Axioms

1. There is a number 0
2. There is a function Suc called the successor function such that if n is a number then $Suc\, n$ is a number
3. 0 is not the successor of any number
4. If two numbers have the same successor then they are equal
5. If a property holds of 0 and if whenever it holds of a number then it also holds of the successor of the number, then the property holds of all numbers

Primitive Recursion

$\vdash \forall x : \alpha . \forall f : a \to num \to \alpha . \exists fun : num \to \alpha . (fun\, 0 = x) \wedge (\forall m . fun(Suc\, m) = f\,(fun\, m)\, m)$

For example: $\vdash + = PrimRec\,(\lambda x_1 . x_1)(\lambda f\, m\, x_1 . Suc(f\, x_1))$

Primitive recursion can be extended to other structures such as lists.

Semantic Embedding

With *deep embedding*, we define the semantics of a term structure by building a function in the host logic which pattern matches on it and assigns some meaning function. This allows theorems to be proved about the embedded terms very simply. With *shallow embedding*, notational conventions are set up for translating term structures into logic terms in a syntactic manner: however, only theorems in the embedded language are provable (i.e. we cannot quantify over program terms).

We can embed our programming language in HOL using the techniques I will go on to describe. Define the type $state = string \to num$, so we can say, e.g. $[\![X + 1]\!] = \lambda s . s' X' + 1$.

Now we can say that $Spec(p, c, q) = \forall s_1 s_2 . p\, s_1 \wedge c(s_1, s_2) \Rightarrow q\, s_2$ where the semantics of commands are:

Rule	Semantics
Skip	$[\![SKIP]\!](s_1, s_2) = s_1 = s_2$
Assignment	$[\![V := E]\!] = Assign('V', [\![E]\!])$
Sequencing	$[\![C_1; C_2]\!] = Seq([\![C_1]\!], [\![C_2]\!])$
If	$[\![\texttt{IF}\ B\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2]\!] = If([\![B]\!], [\![C_1]\!], [\![C_2]\!])$
While	$[\![\texttt{WHILE}\ B\ \texttt{DO}\ C]\!] = While([\![B]\!], [\![C]\!])$

Where:

$Assign(v, e)(s_1, s_2) = (s_2 = Bnd(e, v, s_1))$

$Bnd(e, v, s) = \lambda x . (x = v \to e\, s | s\, x)$

$Seq(c_1, c_2)(s_1, s_2) = \exists s . c_1(s_1, s) \wedge c_2(s, s_2)$

$If(b, c_1, c_2)(s_1, s_2) = (b\, s_1 \to c_1(s_1, s_2) | c_2(s_1, s_2))$

$While(b, c)(s_1, s_2) = \exists n . Iter(n)(b, c)(s_1, s_2)$

$Iter(0)(b, c)(s_1, s_2) = F$

$Iter(Succ\, n)(b, c)(s_1, s_2) = If(b, Seq(c, Iter(n)(b, c)), Skip)(s_1, s_2)$

Note that using these definitions all the rules of Hoare logic can be turned into logical statements about which $Spec$ terms imply each other (with universally quantified free variables) and vice versa.

Termination

A termination assertion is of the form $Halts(p, c) = \forall s_1 . p\, s_1 \Rightarrow \exists s_2 . c(s_1, s_2)$, This is sufficient for languages without nondeterminism i.e. where $\vdash Det[\![C]\!]$ given $Det\, c = \forall s\, s_1\, s_2 . c(s, s_1) \wedge c(s, s_2) \Rightarrow (s_1 = s_2)$. It is straightforward to derive HOL theorems stating termination of all commands except for `WHILE`, which is shown here (including the variant x):

$\forall b\, c\, x . (\forall n . Spec((\lambda s . p\, s \wedge b\, s \wedge (s\, x = n)), c, (\lambda s . p\, s \wedge s\, x < n))) \wedge Halts((\lambda s . p\, s \wedge b\, s), c) \Rightarrow Halts(p, While(b, c))$

Weakest Preconditions

Define $p \Leftarrow q = \forall s. q\, s \Rightarrow p\, s$ to mean p is weaker than q. Now we have:

$Weakest\, P = \epsilon p. P\, p \wedge \forall p'. P\, p' \Rightarrow (p \Leftarrow p')$

$wlp(c, q) = Weakest(\lambda p. Spec(p, c, q))$

$wp(c, q) = Weakest(\lambda p. TotalSpec(p, c, q))$

In practice we use the facts that:

$\vdash wlp(c, q) = \lambda s. \forall s'. c(s, s') \Rightarrow q\, s'$

$\vdash wp(c, q) = \lambda s. (\exists s'. c(s, s')) \wedge \forall s'. c(s, s') \Rightarrow q\, s'$

The relationship to Hoare logic is that:

$\vdash Spec(p, c, q) = \forall s. p\, s \Rightarrow wlp(c, q)\, s$

$\vdash TotalSpec(p, c, q) = \forall s. p\, s \Rightarrow wp(c, q)\, s$

Rule	WP	WLP
Skip	q	
Assignment	$\lambda s. q(Bnd([\![E]\!]s)\ 'V'\ s)$	
Double If	$\lambda s. ([\![B]\!]s \rightarrow wp([\![C_1]\!], s) \mid wp([\![C_2]\!], s))$	$\lambda s. ([\![B]\!]s \rightarrow wlp([\![C_1]\!], s) \mid wlp([\![C_2]\!], s))$
Sequencing	$\vdash Det\ [\![C_1]\!] \Rightarrow wp([\![C_1]\!], wp([\![C_2]\!], q))$	$\vdash wlp([\![C_1]\!], wlp([\![C_2]\!], q))$
While	$\vdash Det\ C \Rightarrow \exists n. Iter\_wp\ n\ [\![B]\!]\ [\![C]\!]\ q\ s$	$\vdash \forall n. Iter\_wp\ n\ [\![B]\!]\ [\![C]\!]\ q\ s$

Where:

$Iter\_wp\ 0\ b\ c\ q = \neg b \wedge p$

$Iter\_wp\ (n+1)\ b\ c\ q = b \wedge wp(c, Iter\_wp\ n\ b\ c\ p)$