Hoare logic

Lecture 2: Formalising the semantics of Hoare logic

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2017/18

Semantics of Hoare logic

Semantics of Hoare logic

Recall: to define a Hoare logic, we need four main components:

- the programming language that we want to reason about: its syntax and dynamic semantics;
- an assertion language for defining state predicates: its syntax and an interpretation;
- an interpretation of Hoare triples;
- a (sound) syntactic proof system for deriving Hoare triples.

This lecture defines a formal semantics of Hoare logic, and introduces properties of Hoare logic (soundness & completeness).

Dynamic semantics of WHILE

Dynamic semantics of WHILE

The dynamic semantics of WHILE will be given in the form of a "big-step" operational semantics.

The reduction relation, written $\langle C, s\rangle \Downarrow s'$, expresses that the command C reduces to the terminal state s' when executed from initial state s.

Dynamic semantics of WHILE

More precisely, these “states” are stacks, which are functions from variables to integers:

$$s \in \mathit{Stack} \overset{\mathit{def}}{=} \mathit{Var} \rightarrow \mathbb{Z}$$

These are **total** functions, and define the current value of every program variable and auxiliary variable.

This models WHILE with arbitrary precision integer arithmetic. A more realistic model might use 32-bit integers and require reasoning about overflow, etc.

Dynamic semantics of WHILE

The reduction relation is defined inductively by a set of inference rule schemas.

To reduce an assignment, we first evaluate the expression E using the current stack, and update the stack with the value of E:

$$\frac{\mathcal{E}[\![E]\!](s) = N}{\langle V := E, s\rangle \Downarrow s[V \mapsto N]}$$

We use functions $\mathcal{E}[\![E]\!](s)$ and $\mathcal{B}[\![B]\!](s)$ to evaluate arithmetic expressions and boolean expressions in a given stack s, respectively.

For example, if $s(X) = 3$, then $\mathcal{E}[\![X + 2]\!](s) = 5$,
so $\langle Y := X + 2, s\rangle \Downarrow s[Y \mapsto 5]$.

Semantics of expressions

$\mathcal{E}[\![E]\!](s)$ evaluates arithmetic expression E to an integer in stack s:

$$\mathcal{E}[\![-]\!](=) : \mathit{Exp} \times \mathit{Stack} \to \mathbb{Z}$$

$$\begin{aligned}
\mathcal{E}[\![N]\!](s) &\overset{\mathit{def}}{=} N \\
\mathcal{E}[\![V]\!](s) &\overset{\mathit{def}}{=} s(V) \\
\mathcal{E}[\![E_1 + E_2]\!](s) &\overset{\mathit{def}}{=} \mathcal{E}[\![E_1]\!](s) + \mathcal{E}[\![E_2]\!](s) \\
&\vdots
\end{aligned}$$

This semantics is too simple to handle operations such as division, which fails to evaluate to an integer on some inputs.

For example, if $s(X) = 3$ and $s(Y) = 0$, then
$\mathcal{E}[\![X + 2]\!](s) = \mathcal{E}[\![X]\!](s) + \mathcal{E}[\![2]\!](s) = 3 + 2 = 5$, and
$\mathcal{E}[\![Y + 4]\!](s) = \mathcal{E}[\![Y]\!](s) + \mathcal{E}[\![4]\!](s) = 0 + 4 = 4$.

Semantics of boolean expressions

$\mathcal{B}[\![B]\!](s)$ evaluates boolean expression B to a boolean in stack s:

$$\mathcal{B}[\![-]\!](=) : \mathit{BExp} \times \mathit{Stack} \to \mathbb{B}$$

$$\mathcal{B}[\![\mathbf{T}]\!](s) \overset{\mathit{def}}{=} \top$$

$$\mathcal{B}[\![\mathbf{F}]\!](s) \overset{\mathit{def}}{=} \bot$$

$$\mathcal{B}[\![E_1 \leq E_2]\!](s) \overset{\mathit{def}}{=} \begin{cases} \top & \text{if } \mathcal{E}[\![E_1]\!](s) \leq \mathcal{E}[\![E_2]\!](s) \\ \bot & \text{otherwise} \end{cases}$$

$$\vdots$$

For example, if $s(X) = 3$ and $s(Y) = 0$, then
$\mathcal{B}[\![X + 2 \geq Y + 4]\!](s) = \mathcal{E}[\![X + 2]\!](s) \geq \mathcal{E}[\![Y + 4]\!](s) = 5 \geq 4 = \top$.

Big-step operational semantics of WHILE

$$\frac{\mathcal{E}[\![E]\!](s) = N}{\langle V := E, s\rangle \Downarrow s[V \mapsto N]} \qquad \frac{\langle C_1, s\rangle \Downarrow s' \qquad \langle C_2, s'\rangle \Downarrow s''}{\langle C_1; C_2, s\rangle \Downarrow s''}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \top \qquad \langle C_1, s\rangle \Downarrow s'}{\langle \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2, s\rangle \Downarrow s'} \qquad \frac{\mathcal{B}[\![B]\!](s) = \bot \qquad \langle C_2, s\rangle \Downarrow s'}{\langle \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2, s\rangle \Downarrow s'}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \top \qquad \langle C, s\rangle \Downarrow s' \qquad \langle \textbf{while } B \textbf{ do } C, s'\rangle \Downarrow s''}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow s''}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \bot}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow s} \qquad \frac{}{\langle \textbf{skip}, s\rangle \Downarrow s}$$

Example reduction in WHILE

For example, if $s(X) = 3$ and $s(Y) = 0$, then we have the following reduction derivation:

$$\dfrac{\mathcal{B}[\![X + 2 \geq Y + 4]\!](s) = \top \qquad \dfrac{\dfrac{}{\langle Y := 2 + X, s\rangle \Downarrow s[Y \mapsto 5]} \qquad \dfrac{}{\langle Y := Y + 1, s[Y \mapsto 5]\rangle \Downarrow s[Y \mapsto 6]}}{\langle Y := 2 + X; Y := Y + 1, s\rangle \Downarrow s[Y \mapsto 6]}}{\langle \textbf{if } X + 2 \geq Y + 4 \textbf{ then } (Y := 2 + X; Y := Y + 1) \textbf{ else } Y := 3, s\rangle \Downarrow s[Y \mapsto 6]}$$

Properties of WHILE

Determinacy

The dynamic semantics of WHILE is deterministic:

$$\langle C, s\rangle \Downarrow s' \land \langle C, s\rangle \Downarrow s'' \Rightarrow s' = s''$$

We have already implicitly used this in the definition of total correctness triples: without this property, we would have to specify whether all reductions or just some reductions satisfy the postcondition.

Substitution

We use $E_1[E_2/V]$ to denote E_1 with E_2 substituted for every occurrence of program variable V:

$$
\begin{aligned}
-[= / \equiv] &: \mathit{Expr} \times \mathit{Expr} \times \mathit{Var} \to \mathit{Expr} \\
N[E_2/V] &\overset{\mathit{def}}{=} N \\
V'[E_2/V] &\overset{\mathit{def}}{=} \begin{cases} \text{if } V' = V & E_2 \\ \text{if } V' \neq V & V' \end{cases} \\
(E_a + E_b)[E_2/V] &\overset{\mathit{def}}{=} (E_a[E_2/V]) + (E_b[E_2/V]) \\
&\vdots
\end{aligned}
$$

For example, $(X + (Y \times 2))[3 + Z/Y] = X + ((3 + Z) \times 2)$.

Substitution property for expressions

We will use the following expression substitution property later:

$$\boxed{\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])}$$

The expression substitution property follows by induction on E_1.

Case $E_1 \equiv N$:

$$\mathcal{E}[\![N[E_2/V]]\!](s) = \mathcal{E}[\![N]\!](s) = N = \mathcal{E}[\![N]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])$$

Proof of substitution property: variable case

$$\boxed{\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])}$$

Case $E_1 \equiv V'$:

$$\begin{aligned}
&\mathcal{E}[\![V'[E_2/V]]\!](s) \\
&= \begin{cases} \text{if } V' = V & \mathcal{E}[\![V[E_2/V]]\!](s) = \mathcal{E}[\![E_2]\!](s) = \mathcal{E}[\![V]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) \\ \text{if } V' \neq V & \mathcal{E}[\![V']\!](s) = s(V') = \mathcal{E}[\![V']\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) \end{cases} \\
&= \mathcal{E}[\![V']\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])
\end{aligned}$$

Proof of substitution property: addition case

$$\boxed{\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])}$$

Case $E_1 \equiv E_a + E_b$:

$$\begin{aligned}
&\mathcal{E}[\![(E_a + E_b)[E_2/V]]\!](s) \\
&\quad = \mathcal{E}[\![(E_a[E_2/V]) + (E_b[E_2/V])]\!](s) \\
&\quad = \mathcal{E}[\![E_a[E_2/V]]\!](s) + \mathcal{E}[\![E_b[E_2/V]]\!](s) \\
&\quad = \mathcal{E}[\![E_a]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) + \mathcal{E}[\![E_b]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) \\
&\quad = \mathcal{E}[\![E_a + E_b]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])
\end{aligned}$$

Semantics of assertions

The language of assertions

Now, we have formally defined the dynamic semantics of the WHILE language that we wish to reason about.

The next step is to formalise the assertion language that we will use to reason about states of WHILE programs.

We take the language of assertions to be an instance of (single-sorted) first-order logic with equality.

Knowledge of first-order logic is assumed. We will review some basic concepts now.

Review of first-order logic

Recall that in first-order logic there are two syntactic classes:

- terms, which denote values, and
- assertions, which denote properties that may be true or false.

Since we are reasoning about WHILE states, our values will be integers, and our assertions will describe properties of WHILE states.

Review of first-order logic: signature

In general, first-order logic is parameterised over a signature that defines function symbols ($+, -, \times$, ...) and predicate symbols (*ODD*, *PRIME*, etc.).

We will be using a particular instance with a signature that includes the usual functions and predicates on integers.

Review of first-order logic: terms

Terms may contain variables like x, X, y, Y, z, Z etc.

Terms, like 1 and $4 + 5$, that do not contain any free variables are called ground terms.

We use conventional notation, e.g. here are some terms:

$$\begin{array}{ccc} X, & y, & Z, \\ 1, & 2, & 325, \\ -X, & -(X+1), & (x \times y) + Z, \\ \sqrt{(1+x^2)}, & X!, & Kolmogorov(x) \end{array}$$

Otherwise, we would have to write $X + 1$ as $+(X, 1)$.

Review of first-order logic: atomic assertions

Examples of atomic assertions are:

$$\bot, \qquad \top, \qquad X = 1, \qquad r < Y, \qquad X = r + (Y \times Q)$$

$\bot$ and $\top$ are atomic assertions that are always (respectively) false and true.

Other atomic assertions are built from terms using predicate symbols and equality. Again, we use conventional notation:

$$X = 1, \qquad (X+1)^2 \geq x^2, \qquad \mathit{PRIME}(3), \qquad \mathit{halts}(x)$$

Here $\geq$, *PRIME*, and *halts* are examples of predicates, and X, 1, $X+1$, $(X+1)^2$ and x^2 are examples of terms.

Otherwise, we would have to write $(X+1)^2 \geq x^2$ as $\geq({}^2(+(X,1)), {}^2(x))$.

Review of first-order logic: compound assertions

Compound assertions are built up from atomic assertions using the usual logical connectives:

$$\wedge\ (\textit{conjunction}), \vee\ (\textit{disjunction}), \Rightarrow\ (\textit{implication})$$

and quantification:

$$\forall\ (\textit{universal}), \exists\ (\textit{existential})$$

Negation, $\neg P$, is a shorthand for $P \Rightarrow \perp$.

The assertion language

The formal syntax of the assertion language is given below:

$$\begin{array}{rcllr}
\nu & ::= & V \mid v & & \textit{variables} \\
t & ::= & \nu \mid f(t_1, \ldots, t_n) & n \geq 0 & \textit{terms} \\
P, Q & ::= & \bot \mid \top \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q & & \textit{assertions} \\
 & \mid & \forall v.\, P \mid \exists v.\, P \mid t_1 = t_2 \mid p(t_1, \ldots, t_n) & n \geq 0 & \\
\neg P & \overset{\textit{def}}{=} & P \Rightarrow \bot & &
\end{array}$$

Quantifiers quantify over terms, and only bind logical variables.

Here f and p range over an unspecified set of function symbols and predicate symbols, respectively, that includes (symbols for) the usual mathematical functions and predicates on integers.
In particular, we assume that they contain symbols that allows us to embed arithmetic expressions E as terms, and boolean expressions B as assertions.

Semantics of terms

$[\![t]\!](s)$ defines the semantics of a term t in a stack s:

$$[\![-]\!](=) : \mathit{Term} \times \mathit{Stack} \to \mathbb{Z}$$

$$[\![\nu]\!](s) \overset{\mathit{def}}{=} s(\nu)$$
$$[\![f(t_1, ..., t_n)]\!](s) \overset{\mathit{def}}{=} [\![f]\!]([\![t_1]\!](s), ..., [\![t_n]\!](s))$$

We assume that the appropriate function $[\![f]\!]$ associated to each function symbol f is provided along with the implicit signature.

In particular, we have $[\![E]\!](s) = \mathcal{E}[\![E]\!](s)$.

Semantics of assertions

$[\![P]\!]$ defines the set of stacks that satisfy the assertion P:

$$[\![-]\!] : \mathit{Assertion} \to \mathcal{P}(\mathit{Stack})$$

$$
\begin{aligned}
[\![\bot]\!] &\overset{\mathit{def}}{=} \{s \in \mathit{Stack} \mid \bot\} = \emptyset \\
[\![\top]\!] &\overset{\mathit{def}}{=} \{s \in \mathit{Stack} \mid \top\} = \mathit{Stack} \\
[\![P \vee Q]\!] &\overset{\mathit{def}}{=} \{s \in \mathit{Stack} \mid s \in [\![P]\!] \vee s \in [\![Q]\!]\} = [\![P]\!] \cup [\![Q]\!] \\
[\![P \wedge Q]\!] &\overset{\mathit{def}}{=} \{s \in \mathit{Stack} \mid s \in [\![P]\!] \wedge s \in [\![Q]\!]\} = [\![P]\!] \cap [\![Q]\!] \\
[\![P \Rightarrow Q]\!] &\overset{\mathit{def}}{=} \{s \in \mathit{Stack} \mid s \in [\![P]\!] \Rightarrow s \in [\![Q]\!]\}
\end{aligned}
$$

(continued)

Semantics of assertions (continued)

$$\llbracket t_1 = t_2 \rrbracket \overset{\text{def}}{=} \{s \in \mathit{Stack} \mid \llbracket t_1 \rrbracket(s) = \llbracket t_2 \rrbracket(s)\}$$

$$\llbracket p(t_1, ..., t_n) \rrbracket \overset{\text{def}}{=} \{s \in \mathit{Stack} \mid \llbracket p \rrbracket(\llbracket t_1 \rrbracket(s), ..., \llbracket t_n \rrbracket(s))\}$$

$$\llbracket \forall v.\, P \rrbracket \overset{\text{def}}{=} \{s \in \mathit{Stack} \mid \forall N.\, s[v \mapsto N] \in \llbracket P \rrbracket\}$$

$$\llbracket \exists v.\, P \rrbracket \overset{\text{def}}{=} \{s \in \mathit{Stack} \mid \exists N.\, s[v \mapsto N] \in \llbracket P \rrbracket\}$$

We assume that the appropriate predicate $\llbracket p \rrbracket$ associated to each predicate symbol p is provided along with the implicit signature.

In particular, we have $\llbracket B \rrbracket = \{s \mid \mathcal{B}\llbracket B \rrbracket(s) = \top\}$.

This interpretation is related to the forcing relation you used in Part IB "Logic and Proof": $s \in \llbracket P \rrbracket \Leftrightarrow s \models P$.

Substitutions

We use $t[E/V]$ and $P[E/V]$ to denote t and P with E substituted for every occurrence of program variable V, respectively.

Since our quantifiers bind logical variables, and all free variables in E are program variables, there is no issue with variable capture:

$$(\forall v.\, P)[E/V] \overset{\mathit{def}}{=} \forall v.\, (P[E/V])$$

$$\vdots$$

Substitution property

The term and assertion semantics satisfy a similar substitution property to the expression semantics:

- $[\![t[E/V]]\!](s) = [\![t]\!](s[V \mapsto \mathcal{E}[\![E]\!](s)])$
- $s \in [\![P[E/V]]\!] \Leftrightarrow s[V \mapsto \mathcal{E}[\![E]\!](s)] \in [\![P]\!]$

They are easily provable by induction on t and P, respectively: the former by using the substitution property for expressions, and the latter by using the former. (Exercise)

The latter property will be useful in the proof of soundness of the syntactic assignment rule.

Semantics of Hoare logic

Semantics of partial correctness triples

Now that we have formally defined the dynamic semantics of WHILE and our assertion language, we can define the formal meaning of our triples.

A partial correctness triple asserts that if the given command terminates when executed from an initial state that satisfies the precondition, then the terminal state must satisfy the postcondition:

$$\models \{P\}\ C\ \{Q\} \quad \overset{def}{=} \quad \forall s, s'.\, s \in [\![P]\!] \land \langle C, s\rangle \Downarrow s' \Rightarrow s' \in [\![Q]\!]$$

Semantics of total correctness triples

A total correctness triple asserts that when the given command is executed from an initial state that satisfies the precondition, then it must terminate in a terminal state that satisfies the postcondition:

$$\models [P]\ C\ [Q] \quad \overset{def}{=} \quad \forall s.\, s \in [\![P]\!] \Rightarrow \exists s'.\, \langle C, s\rangle \Downarrow s' \wedge s' \in [\![Q]\!]$$

Since WHILE is deterministic, if one terminating execution satisfies the postcondition, then all terminating executions satisfy the postcondition.

There is a blind spot here: we do not even have a way of saying that there are no other, non-terminating executions.

Properties of Hoare logic

Properties of Hoare logic

Now, we have a syntactic proof system for deriving Hoare triples, $\vdash \{P\}\ C\ \{Q\}$, and a formal definition of the meaning of our Hoare triples, $\models \{P\}\ C\ \{Q\}$.

How are these related?

We might hope that any triple that can be derived syntactically holds semantically (soundness), and that any triple that holds semantically is syntactically derivable (completeness).

Hoare logic is sound but **not** complete.

Soundness of Hoare logic

Soundness of Hoare logic

Theorem (Soundness)
If $\vdash \{P\}\ C\ \{Q\}$ *then* $\models \{P\}\ C\ \{Q\}$.

Soundness expresses that any triple derivable using the syntactic proof system holds semantically.

Soundness can be proved by induction on the $\vdash \{P\}\ C\ \{Q\}$ derivation:

- it suffices to show, for each inference rule, that if each hypothesis holds semantically (that is what our induction hypothesis gives us), then the conclusion holds semantically.

Soundness of the assignment rule

$$\boxed{\models \{P[E/V]\}\ V := E\ \{P\}}$$

Assume $s \in [\![P[E/V]]\!]$ and $\langle V := E, s\rangle \Downarrow s'$.

From the substitution property, it follows that $s[V \mapsto \mathcal{E}[\![E]\!](s)] \in [\![P]\!]$.

From inversion on the reduction, there exists an N such that $\mathcal{E}[\![E]\!](s) = N$ and $s' = s[V \mapsto N]$, so $s' = s[V \mapsto \mathcal{E}[\![E]\!](s)]$.

Hence, $s' \in [\![P]\!]$.

Soundness of the loop rule

$$\boxed{\text{If } \models \{P \land B\}\ C\ \{P\} \text{ then } \models \{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P \land \neg B\}}$$

How can we get past the fact that the loop reduction rules define the reduction of a loop in terms of itself?

We will prove $\models \{P\}$ **while** B **do** C $\{P \land \neg B\}$ by proving a modified version of the property for a modified but equivalent reduction relation.

Instrumented big-step operational semantics of WHILE

We can write an instrumented version of our big-step operational semantics of WHILE that counts how many times the body of the top-level loop is executed:

$$\frac{\mathcal{B}[\![B]\!](s) = \top \qquad \langle C, s\rangle \Downarrow s' \qquad \langle \textbf{while } B \textbf{ do } C, s'\rangle \Downarrow^{n} s''}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^{n+1} s''}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \bot}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^{0} s}$$

that is equivalent to the original dynamic semantics in the following sense:

$$\big(\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow s'\big) \Leftrightarrow \big(\exists n.\, \langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^{n} s'\big)$$

Soundness of the loop rule: base case

If (IH) $\forall s, s'.\, s \in [\![P \wedge B]\!] \wedge \langle C, s\rangle \Downarrow s' \Rightarrow s' \in [\![P]\!]$, then
$\forall n.\, \forall s, s'.\, s \in [\![P]\!] \wedge \langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^n s' \Rightarrow s' \in [\![P \wedge \neg B]\!]$

We can prove this by a (nested) induction on n:

Case 0: assume $s \in [\![P]\!]$ and $\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^0 s'$.

Since the loop reduced in 0 iterations, B must have evaluated to false: $\mathcal{B}[\![B]\!](s) = \bot$ and $s' = s$.

Since $\mathcal{B}[\![B]\!](s) = \bot$, $s \notin [\![B]\!]$, so $s \in [\![B]\!] \Rightarrow s \in [\![\bot]\!]$, so $s \in [\![B \Rightarrow \bot]\!]$, so $s \in [\![\neg B]\!]$. Therefore, $s \in [\![P \wedge \neg B]\!]$.
Hence, $s' = s \in [\![P \wedge \neg B]\!]$.

Soundness of the loop rule: inductive case

If (IH) $\forall s, s'.\, s \in [\![P \wedge B]\!] \wedge \langle C, s\rangle \Downarrow s' \Rightarrow s' \in [\![P]\!]$, then
$\forall n.\, \forall s, s'.\, s \in [\![P]\!] \wedge \langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^n s' \Rightarrow s' \in [\![P \wedge \neg B]\!]$

Case $n+1$: assume $s \in [\![P]\!]$, $\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^{n+1} s'$, and (nIH) $\forall s, s'.\, s \in [\![P]\!] \wedge \langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^n s' \Rightarrow s' \in [\![P \wedge \neg B]\!]$.

Since the loop reduced in one iteration or more, B must have evaluated to true: $\mathcal{B}[\![B]\!](s) = \top$, and there exists an s^* such that $\langle C, s\rangle \Downarrow s^*$ and $\langle \textbf{while } B \textbf{ do } C, s^*\rangle \Downarrow^n s'$.

Since $\mathcal{B}[\![B]\!](s) = \top$, $s \in [\![B]\!]$. Therefore, $s \in [\![P \wedge B]\!]$.

From the outer induction hypothesis IH, it follows that $s^* \in [\![P]\!]$, and so by the inner induction hypothesis nIH, $s' \in [\![P \wedge \neg B]\!]$.

Other properties of Hoare logic

Completeness

Completeness is the converse property of soundness:
If $\models \{P\}\ C\ \{Q\}$ then $\vdash \{P\}\ C\ \{Q\}$.

Our Hoare logic inherits the incompleteness of arithmetic and is therefore **not** complete.

Completeness

To see why, assume that, using our syntactic proof system, we can derive any triple that holds semantically.

Then, for every assertion P that is true in arithmetic, that is, such that $\forall s.\, s \in [\![P]\!]$, and hence such that $\models \{\top\}$ **skip** $\{P\}$, we can derive $\vdash \{\top\}$ **skip** $\{P\}$.

Then, by examining that derivation, we have a derivation of $\vdash \top \Rightarrow P$, and hence a derivation of $\vdash P$.

Since the assertion logic (which includes arithmetic) is **not** complete, this is not the case.

Relative completeness

The previous argument showed that because the assertion logic is not complete, then neither is Hoare logic.

However, Hoare logic is **relatively complete** for our simple language:

- Relative completeness expresses that any failure to derive $\vdash \{P\}\ C\ \{Q\}$ for a statement that holds semantically can be traced back to a failure to prove $\vdash R$ for some valid arithmetic statement R.

In practice, completeness is not that important, and there is more focus on nice, usable rules.

Decidability

Finally, Hoare logic is not decidable.

$\vDash \{\top\}\ C\ \{\bot\}$ holds if and only if C does not terminate.
Moreover, we can encode Turing machines in WHILE.
Hence, since the Halting problem is undecidable, so is Hoare logic.

Summary

We have defined a dynamic semantics for the WHILE language, and a formal semantics for a Hoare logic for WHILE.

We have shown that the syntactic proof system from the last lecture is sound with respect to this semantics, but not complete.

Supplementary reading on soundness and completeness:

- Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. Chapters 6–7.
- Software Foundations, Benjamin C. Pierce et al.

In the next lecture, we will look at examples of proofs in Hoare logic.