Today’s Lecture

- Formalize specifications as Hoare triples
- Talk about two types of correctness: partial and total
- Introduce simpler proof technique than using semantics directly
Recall *contracts* from before

<table>
<thead>
<tr>
<th></th>
<th>method Abs(x: int) returns (y: int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ensures 0 &lt;= y</td>
</tr>
<tr>
<td>2</td>
<td>ensures 0 &lt;= x ==&gt; y == x</td>
</tr>
<tr>
<td>3</td>
<td>ensures x &lt; 0 ==&gt; y == -x</td>
</tr>
</tbody>
</table>

Hoare triples formalize contracts

Ideally, program specifications should be:

- **Declarative**: help to understand the program’s behavior without describing how it works
- **Informative**: detailed documentation describing expectations and facilities the code
- **Refutable**: when a specification doesn’t match the code, we should always be able to tell
Recall the language Imp

\[ a \in \text{AEexp} ::= n \in \mathbb{Z} \mid x \in \text{Var} \mid a_1 + a_2 \mid a_1 \times a_2 \]

\[ b \in \text{Bexp} ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \land b_2 \]

\[ c \in \text{Com} ::= \text{skip} \mid x := a \mid c_1 ; c_2 \]
\[ \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \]
\[ \mid \text{while } b \text{ do } c \]

We’ll continue assuming that this is our language today
Partial correctness refers to a program’s terminating behavior. We specify partial correctness using Hoare triples:

$$\{P\} \ c \ {Q}\$$

- $c$ is a program
- $P$ and $Q$ are assertions in a first-order theory
- Free variables in $P, Q$ can range over program variables
- $P$ is the precondition and $Q$ is the postcondition
Hoare Triples: Meaning

The meaning of \( \{P\} \ c \ \{Q\} \) is as follows:

- If we begin executing \( c \) in an **environment satisfying** \( P \),
- and if \( c \) **terminates**, 
- then its final environment will satisfy \( Q \)

The specification says nothing about:

- Executions that do not terminate (i.e., **diverge**)
- Executions that do not begin in \( P \)

**Notice:** \( \{P\} \ c \ \{Q\} \) is a predicate

Goal of verification: prove that it holds, i.e., is a **valid** Hoare triple
Partial correctness doesn’t require termination

**Total correctness** is a stronger statement, written:

\[ [P] \ c \ [Q] \]

The meaning of \([P] \ c \ [Q]\) is:

- If we begin executing \(c\) in an **environment satisfying** \(P\),
- then \(c\) **terminates**,
- and its final environment will satisfy \(Q\)

Total correctness introduces another obligation for verification
What do the following Hoare triples mean?

1. $\{true\} \ c \ {Q} \ If \ c \ terminates, \ then \ Q \ holds$
2. $[true] \ c \ [Q] \ c \ terminates, \ and \ Q \ always \ holds \ after$
3. $[P] \ c \ [true] \ If \ c \ starts \ in \ P, \ then \ c \ terminates$
4. $[true] \ c \ [true] \ c \ terminates$
5. $\{0 < x\} \ while \ 0 < x \ do \ x := x + 1 \ {false} \ c \ does \ not \ terminate \ when \ starting \ in \ 0 < x$
6. $\{true\} \ c \ {false} \ c \ does \ not \ terminate$
Is the following a valid Hoare triple?

1. \{x = 2\} while 0 < x do x := x - 1 \{x = 0\} Yes
2. \{x = 0 \land y = 1\} x := x + 1 \{x = 1 \land y = 2\} No
3. \{true\} while 0 < x do x := x - 1 \{x \leq 0\} Yes
4. \{0 < x\} while 0 < x do x := x + 1 \{x > 0\} Yes
5. [true] while 0 < x do x := x + 1 [x \leq 0] No
6. [x \leq 0] while 0 < x do x := x + 1 [x \leq 0] Yes
Examples: Writing specifications

When \( c \) terminates, \( x \) contains the maximum value of \( b[0 : (n - 1)] \)

\[
\{ \text{true} \} \ c \ \{ \forall i.0 \leq i < n \rightarrow b[i] \leq x \}
\]

The program \( t := x; x := y; y := t \) swaps \( x \) and \( y \)

\[
\{ x = x' \land y = y' \} \ t := x; x := y; y := t \ \{ x = y' \land y = x' \}
\]
Examples: Writing specifications

\(c\) \textbf{returns a sorted array} \(b\)

\[
\{true\} \ c \ \{\forall i. 0 \leq i < n - 1 \rightarrow b[i] \leq b[i + 1]\}
\]

\(c\) \textbf{sorts the array} \(b\)

\[
\{b = b'\} \ c \ \{\text{perm}(b, b') \wedge \forall i. 0 \leq i < n - 1 \rightarrow b[i] \leq b[i + 1]\}
\]

where

\[
\text{perm}(a, b) \leftrightarrow \forall e. \text{count}(e, a) = \text{count}(e, b)
\]

\[
\text{count}(e, a) \overset{\text{def}}{=} \begin{cases} 
0 & \text{if } n = 0 \\
1 + \text{count}(e, a[1 : (n - 1)]) & \text{if } 0 < n \land a[0] = e \\
\text{count}(e, a[1 : (n - 1)]) & \text{if } 0 < n \land a[0] \neq e
\end{cases}
\]

\textbf{Note:} This is why we have functions and predicates in Dafny
Assertion Language

Given a triple

\{P\} c \{Q\}

the formulas \(P\) and \(Q\) are assertions in some first-order language \(\Sigma_T\)

For \(\text{Imp}\), a useful theory is \(T_Z \cup T_A\)

The variables appearing in \(P\) and \(Q\) can contain program variables

\[pvars(Q) = \text{vars}(c) \cap \text{vars}(Q)\]

As well as additional **logical variables**

\[lvars(Q) = \text{free}(Q) - \text{vars}(c)\]
Semantics of Assertions

Given an assertion $Q$, we need to define “$P$ holds in environment $\sigma$”

Because $Q$ may have logical variables, we need more than $\sigma$

We’ll define satisfiability and validity in this context:

$$\sigma \models_\alpha Q$$  \hspace{1cm} $\sigma$ satisfies $Q$ under $\alpha$

$$\sigma \models Q$$  \hspace{1cm} $Q$ is valid in $\sigma$

**Note:** $\alpha$ only maps $lvars(Q)$, and $\sigma$ only $pvars(Q)$

The way we defined them, these two sets are disjoint
Now we want to define the relations $\models_\alpha$ and $\models$

We can do this by extending the assignment $\alpha$ with $\sigma$:

$$(\alpha \cup \sigma)(x) \overset{\text{def}}{=} \begin{cases} 
\alpha(x) & \text{if } x \in \alpha \\
\sigma(x) & \text{if } x \in \sigma
\end{cases}$$

Then we have **satisfiability**: $\sigma \models_\alpha Q$ iff $(\sigma \cup \alpha) \models_T Q$

And **assertion validity**: $\sigma \models Q$ iff for all $\alpha$, $(\sigma \cup \alpha) \models_T Q$

Finally, we will simply write $\models Q$ if $\sigma \models Q$ for all $\sigma$

**Summary**: when we say “$Q$ holds in $\sigma$”, we generally mean $\sigma \models Q$
Now that we have assertion semantics, we lift them to specifications

\{P\} c \{Q\} \textit{is valid in } \sigma \textit{ under } \alpha \textit{, written } \sigma \models_\alpha \{P\} c \{Q\}:

\forall \sigma'.(\sigma \models_\alpha P \land \langle c, \sigma \rangle \downarrow \sigma') \rightarrow \sigma' \models_\alpha Q

1. Whenever \sigma \textit{satisfies } P \textit{ under } \alpha,
2. and executing \( c \) \textit{in } \sigma' \textit{yields } \sigma',
3. then \sigma' \textit{satisfies } Q \textit{ under } \alpha

Then we say that \{P\} c \{Q\} \textit{is valid, written } \models \{P\} c \{Q\}, \textit{if:}

\forall \sigma, \alpha.\sigma \models_\alpha \{P\} c \{Q\}

\textit{l.e., for all initial states an } lvars \textit{ assignments, } \sigma \models_\alpha \{P\} c \{Q\} \textit{ holds}
We can use the operational semantics to prove triples.

However, working directly with the definition is clunky:

\[ \forall \sigma, \sigma', \alpha. (\sigma \models_\alpha P \land \langle c, \sigma \rangle \Downarrow \sigma') \rightarrow \sigma' \models_\alpha Q \]

We’ll look at a logic for deriving new triples from existing ones.

This is called the **Hoare logic**.

Set of inference rules for producing judgements of the form:

\[ \vdash \{ P \} \ c \ \{ Q \} \]

One rule for each command in \texttt{Imp}, plus the **rule of consequence**.
As always, reasoning for skip is trivial:

\[
\text{Skip} \quad \frac{}{\{P\} \text{skip} \{P\}}
\]

The rule for assignment is less straightforward:

\[
\text{Asgn} \quad \frac{}{\{Q[a/x]\} \ x := a\{Q\}}
\]

Read \(Q[a/x]\) as “\(Q\) with \(a\) substituted for \(x\)”

- For example, \((5 + x)[(x + 1)/x]\) is \(5 + (x + 1)\)

So, \(Q\) holds after assignment if \(Q'\) holds before, where

- \(Q'\) replaces the left-hand-side with the right-hand-side within \(Q\)
What is the precondition $P$ of:

1. $\vdash \{ P \} \ x := 1 \ \{ x = y \} \ 1 = y$
2. $\vdash \{ P \} \ x := x + 1 \ \{ x = n \} \ x + 1 = n$
3. $\vdash \{ P \} \ x := 1 \ \{ y = 3 \} \ y = 3$
4. $\vdash \{ P \} \ x := 1 \ \{ x = 1 \} \ 1 = 1$

What is the postcondition $Q$ of:

1. $\vdash \{ y = 0 \} \ x := 1 \ \{ Q \} \ y = 0$

Is $\vdash \{ y = 0 \} \ x := 1 \ \{ x = 1 \}$ provable using Asgn?
We couldn’t prove something that seemed simple (and indeed valid):

\[ \vdash \{ y = 0 \} \ x := 1 \ \{ x = 1 \} \]

But we could prove \[ \vdash \{ 1 = 1 \} \ x := 1 \ \{ x = 1 \} \]

If we can prove \( x = 1 \) assuming \( 1 = 1 \) (which is equiv. to \textit{true}), we ought to be able to prove it even if we do make assumptions.

This is the idea behind \textbf{precondition strengthening}:

\[
\begin{align*}
\text{Pre} & \quad \vdash \{ P' \} \ c \ \{ Q \} \quad P \Rightarrow P' \\
\{ P \} & \quad c \ \{ Q \}
\end{align*}
\]
Applying this to our previous example, we wanted to prove:

$$\{true\} \ x := a \ \{x = 1\}$$

\[
\begin{array}{c}
\text{Asgn} \\
\text{Pre}
\end{array}
\quad
\begin{array}{c}
\vdash \{1 = 1\} \ x := 1 \ \{x = 1\}
\quad
\text{true} \Rightarrow 1 = 1
\end{array}
\quad
\begin{array}{c}
\{true\} \ x := a \ \{x = 1\}
\end{array}
\]
Hoare Logic: Weakening

Similarly, we have a rule for weakening the postcondition:

\[
\text{Post} \quad \frac{\vdash \{P\} c \{Q'\} \quad Q' \Rightarrow Q}{\{P\} c \{Q\}}
\]

Intuitively, if we can prove a **strong** statement that covers many facts

We can also prove a simpler (**weak**) statement that covers less

The rule of **consequence** combines both of these rules:

\[
\text{Conseq} \quad \frac{P \Rightarrow P' \quad \vdash \{P'\} c \{Q'\} \quad Q' \Rightarrow Q}{\{P\} c \{Q\}}
\]
Given triples for $c_1$ and $c_2$, this gives us one for $c_1; c_2$

Note that the premises need to “meet in the middle”

Oftentimes, use strengthening and weakening to achieve this
Example

Now let’s use Hoare logic to prove swap:

\[
\{x = x' \land y = y'\} \quad t := x; x := y; y := t \quad \{y = x' \land x = y'\}
\]

1. By Asgn, we have \(\{x = x' \land y = y'\} \quad t := x \quad \{t = x' \land y = y'\}\)
2. By Asgn, we have \(\{t = x' \land y = y'\} \quad x := y \quad \{t = x' \land x = y'\}\)
3. By Asgn, we have \(\{t = x' \land x = y'\} \quad y := t \quad \{y = x' \land x = y'\}\)
4. By 1, 2, Seq, \(\{x = x' \land y = y'\} t := x; x := y \{t = x' \land x = y'\}\)
5. Finally, by 3, 4, and Seq, we have:

\[
\{x = x' \land y = y'\} \quad t := x; x := y; y := t \quad \{y = x' \land x = y'\}
\]
Oftentimes it helps to write the intermediate assertions as you go:

1. \( \{x = x' \land y = y'\} \) (assumption)
2. \( t := x; \)
3. \( \{t = x' \land y = y'\} \) (asgn)
4. \( x := y; \)
5. \( \{t = x' \land x = y'\} \) (asgn)
6. \( y := t \)
7. \( \{y = x' \land x = y'\} \) (asgn)

You can then combine with \( S_{eq} \) at the end

It’s good form to mention which lines apply to \( S_{eq} \) when used
Hoare Logic: Conditional

\[
\text{If} \quad \begin{array}{c}
\{ P \wedge b \} \quad c_1 \quad \{ Q \} \\
\{ P \wedge \neg b \} \quad c_2 \quad \{ Q \}
\end{array}
\]

\[
\{ P \} \quad \text{if } b \text{ then } c_1 \quad \text{else } c_2 \{ Q \}
\]

- At the beginning of the \emph{true} branch, we know that \( b \) holds.
- So, we want to show that \( \{ P \wedge b \} \quad c_1 \quad \{ Q \} \)
- Similarly, in the false branch, \( \neg b \) must hold.
- So, we want to show that \( \{ P \wedge \neg b \} \quad c_2 \quad \{ Q \} \)
Hoare Logic: While Loop

While \( \{P \land b\} \ c \ \{P\} \)
\(\{P\} \ while \ b \ do \ c \ \{P \land \neg b\}\)

- What role does \(P\) play in this rule
- \(P\) is a **loop invariant**
- \(P\) holds before the loop, and is preserved by each iteration
- This is formalized in the premise \(\{P \land b\} \ c \ \{P\}\)
- To use While, prove that \(P\) is invariant wrt. \(c\)
Example

\{true\}
\begin{align*}
  r & := x; \quad q := 0; \\
  \textbf{while} \quad y \leq r \quad \textbf{do} \\
  & \quad r := r - y; \quad q := q + 1 \\
  & \quad \{ r < y \land x = r + (q \times y) \}
\end{align*}

To start out, what should our invariant be?

\[ P : x = r + (q \times y) \]

Now we need to prove that \( P \) is preserved by the loop:

\[ \{ P \land b \} \quad c \quad \{ P \} \]
We’re obligated to show that,
\[
\{ x = r + (q \times y) \land y \leq r \} \quad r := r - y; \quad q := q + 1 \{ x = r + (q \times y) \}
\]

Roughly, we’ll use two applications of Asgn, followed by Seq

Oftentimes, it’s easier to work backwards from the goal

In this case, what \( P' \) do we need to establish:
\[
\{ P' \} \quad q := q + 1 \{ x = r + (q \times y) \}
\]
\[P' : x = r + ((q + 1) \times y)\]

Now, what \( P'' \) do we need to establish:
\[
\{ P'' \} \quad r := r - y \{ x = r + ((q + 1) \times y) \}
\]
\[P'' : x = (r - y) + ((q + 1) \times y)\]
We now have the following:

1. \( \{ x = (r - y) + ((q + 1) \times y) \} \)
2. \( r := r - y; \)
3. \( \{ x = r + ((q + 1) \times y) \} \)
4. \( q := q + 1; \)
5. \( \{ x = r + (q \times y) \} \)

We wanted to show that:

\[ \{ x = r + (q \times y) \land y \leq r \} \quad r := r - y; q := q + 1 \quad \{ x = r + (q \times y) \} \]

We can use strengthening to show that:

\[ (x = r + (q \times y) \land y \leq r) \Rightarrow x = (r - y) + ((q + 1) \times y) \]

This is called a **proof obligation**
Example

We now have the following:

1. \( \{ x = r + (q \times y) \land y \leq r \} \)
2. \( r := r - y; \)
3. \( \{ x = r + ((q + 1) \times y) \} \)
4. \( q := q + 1; \)
5. \( \{ x = r + (q \times y) \} \)

Applying Seq, we arrive at:

\[ \{ x = r + (q \times y) \land y \leq r \} \quad r := r - y; q := q + 1 \quad \{ x = r + (q \times y) \} \]

Now by the While rule, we have:

\[ \{ x = r + (q \times y) \} \]

**while** \( y \leq r \) **do**

\[ r := r - y; q := q + 1 \]

\[ \{ r < y \land x = r + (q \times y) \} \]
Recall that our program was:

\[
\{true\} \\
r := x; q := 0; \\
\textbf{while } y \leq r \textbf{ do} \\
\quad r := r - y; q := q + 1 \\
\{r < y \land x = r + (q \times y)\}
\]

What’s next?

We need to show that:

\[
\{true\} r := x; q := 0 \\{x = r + (q \times y)\}
\]
Example

\{true\} r := x; q := 0 \{x = r + (q \times y)\}

Working backwards again:

1. By Asgn, \{x = r + (0 \times y)\} q := 0 \{x = r + (q \times y)\}
2. By Asgn, \{r = r + (0 \times y)\} r := x \{x = r + (0 \times y)\}
3. Logically, \textit{true} \Rightarrow (r = r + (0 \times y))
4. By Conseq, \{true\} r := x \{x = r + (0 \times y)\}
5. Finally, by Seq: \{true\} r := x; q := 0 \{x = r + (q \times y)\}

We complete the proof with one more application of Seq
Do these rules work for total correctness?

Most of them do, except While

Consider the following proof:

1. $\{true\} x := 0 \{true\}$, by Asgn
2. $\{true \land true\} x := 0 \{true\}$, by Conseq
3. $\{true\}$ while true do $x := 0 \{true \land \neg true\}$, by While

This is fine for partial correctness, because the loop diverges

But if While held for total correctness, then we could derive:

$$\vdash [true] \text{ while true do } x := 0 [false]$$
The proof rules we’ve just covered are sound for partial correctness:

\[
\text{If } \vdash \{P\} \ c \ \{Q\}, \ \text{then } \models \{P\} \ c \ \{Q\}
\]

If we can derive a triple using the rules, then it is valid.

To prove this, we use the operational semantics.

Show equivalence between proof rules and reductions.

Need to use induction on derivations.
Completeness of Hoare Logic

Completeness of Hoare logic is stated as:

If \[ \models \{P\} \ c \ \{Q\}, \] then \[ \vdash \{P\} \ c \ \{Q\} \]

If \[ \{P\} \ c \ \{Q\} \] is valid, then we can derive it using the rules

Is this true?

For strengthening, we need to prove statements of the form:

\[ P \Rightarrow Q \]

This requires proving a universal implication in Peano arithmetic

Recall that \( T_{PA} \) is undecidable!
So, we know there can’t be a proof system that derives all valid triples

The Hoare logic has relative completeness

If we assume an oracle for deciding $P \Rightarrow Q$

Then we can derive any valid Hoare triple for Imp

However, for more complex languages, this isn’t always the case
Next time, we’ll see how to mechanize these proofs further

1. Predicate transformers
2. Verification conditions
3. Basic paths