Axiomatic Semantics

- Operational semantics describes the meaning of programs in terms of the execution steps taken by an abstract machine
- Denotational semantics describes the meaning of programs with formal mathematical objects
- Axiomatic semantics describes the meaning of programs in terms of properties (axioms) about them
  - Captures dynamic behavior of individual programs
  - Vehicle for reasoning about program correctness
States

- Specification language is first-order predicate logic
  - Terms (variables, constants, arithmetic operations)
  - Formulae, e.g.
    - True and false
    - If $t_1$ and $t_2$ are terms then $t_1 = t_2$ and $t_1 < t_2$ are formulae
    - If $\phi$ and $\psi$ are formulae then, $\phi \land \psi$, $\phi \lor \psi$, $\neg \phi$ and $\phi \Rightarrow \psi$ are formulae.
    - If $\phi(x)$ is a formula (with $x$ possibly free) then, $\forall x. \phi(x)$ and $\exists x. \phi(x)$ are formulae.
Hoare Triples

- **Meaning of construct S can be described in terms of triples** \( \{ P \} S \{ Q \} \)
  - \( P \) and \( Q \) are formulae or assertions
    - \( P \) is a precondition on \( S \)
    - \( Q \) is a postcondition on \( S \)
  - **Asserts a fact** (may be either true or false)
  - The triple is valid if:
    - execution of \( S \) begins in a state satisfying \( P \)
    - \( S \) terminates
    - resulting state satisfies \( Q \)
Satisfiability

- A formula in first-order logic can be used to characterize states
  - The formula \( \{ x = 3 \} \) characterizes all program states in which the value of the location associated with \( x \) is 3
  - Think of formulas as assertions about states
- Define \( \{ \sigma \in \Sigma \mid \sigma \models \phi \} \) where “\( \models \)” is a satisfiability relation
- Think of \( \{ P \} \ c \ \{ Q \} \) for command \( c \) as meaning
  - \( \sigma \models \{ P \} \ c \ \{ Q \} \) iff \( \sigma \models P \Rightarrow C[ c ]\sigma \models Q \)
  - \( \bot \models P \) for any assertion \( P \)
Formal Semantics

- $\sigma \models \text{true always}$
- $\sigma \models e_1 = e_2$ iff $[e_1] \sigma = [e_2] \sigma$
- $\sigma \models e_1 \leq e_2$ iff $[e_1] \sigma \leq [e_2] \sigma$
- $\sigma \models A_1 \land A_2$ iff $\sigma \models A_1$ and $\sigma \models A_2$
- $\sigma \models A_1 \lor A_2$ iff $\sigma \models A_1$ or $\sigma \models A_2$
- $\sigma \models A_1 \Rightarrow A_2$ iff $\sigma \models A_1$ implies $\sigma \models A_2$
- $\sigma \models \forall x. A$ iff $\forall n \in \mathbb{Z} \sigma[x \mapsto n] \models A$
- $\sigma \models \exists x. A$ iff $\exists n \in \mathbb{Z} \sigma[x \mapsto n] \models A$
Validity

- Often not so much interested in assertions for particular states, e.g.
  - \{ i < x \} x := x + 1 \{i < x\}
  - Is valid in all states
    - All values for x
    - All interpretations for i

- Validity
  - \( \forall \sigma. \sigma \models \{P\} c \{Q\} \)
  - \( \models \{P\} c \{Q\} \)
Semantics of Assertions

- Note: $i$ in the previous is integer variable introduced for assertions
  - E.g. assertion to express that a number $k$ is not prime: $\exists \ i, l \geq 2 \ k = i \times l$
  - Can be free or bound (cf. lambda)
    - What binds assertion variables?
- Formal meaning
  - $\sigma \models \{P\} \Rightarrow \{Q\}$:
    - $\forall \ \sigma, \sigma' \in \Sigma \ (\sigma \models P \land <c, \sigma> \downarrow \sigma') \Rightarrow \sigma' \models Q$
Examples

- \( \{ 2 = 2 \} \ x := 2 \ {\ x = 2} \)
  - An assignment operation of \( x \) to 2 results in a state in which \( x \) is 2, assuming equality of integers.

- \( \{ \text{True} \} \) if \( B \) then \( x := 2 \) else \( x := 1 \) \( \{ x = 1 \lor x = 2 \} \)
  - A conditional expression that either assigns \( x \) to 1 or 2 if executed will lead to a state in which \( x \) is either 1 or 2.

- \( \{ 2=2 \} \ x := 2 \ {\ y = 1} \)

- \( \{ \text{True} \} \) if \( B \) then \( x := 2 \) else \( x := 1 \) \( \{ x = 1 \land x = 2 \} \)

- The above two triples are invalid. Why?
Partial and total correctness

- The validity of a Hoare triple depends upon statement S terminating
  - Partial correctness assertions
  - E.g., \( \{ 0 \leq a \land 0 \leq b \} S \{ z = a \times b \} \)
    - If executed in a state in which \( 0 \leq a \) and \( 0 \leq b \), and
    - S terminates,
    - then \( z = a \times b \)

- Alternative: total correctness assertions
  - \( [P] c [Q] \): execution of c from state satisfying P terminates in a state which satisfies Q
  - Formal semantics has additional clause
  - \( \sigma \models [P] c [Q] \):
    - \( \forall \sigma, \sigma' \in \Sigma (\sigma \models P \land <c, \sigma> \Downarrow \sigma') \Rightarrow \sigma' \models Q \)
    - \( \land \forall \sigma \in \Sigma \sigma \models P \Rightarrow \exists \sigma' \in \Sigma <c, \sigma> \Downarrow \sigma' \)
A Theory

- Wanted: theory for proving soundness of programs
  - Mathematical framework for proving properties about a certain object domain
  - Such properties are called theorems
- Components of a theory
  - Grammar (e.g. BNF) defines well-formed formulae (WFF)
  - Axioms: formulae asserted to be theorems
  - Inference rules: ways to prove new theorems from previously obtained theorems
Hoare Logic

- Proof system based on Hoare rules
  - derivations are proofs
  - conclusions are theorems
- Write $\vdash \{P\} c \{Q\}$ if $\{P\} c \{Q\}$ is a theorem
- If $\vdash \{P\} c \{Q\}$ then $\models \{P\} c \{Q\}$
  - Any derivable assertion is sound wrt to the underlying semantics
Proof Rules

- **Skip:**
  \[ \vdash \{ P \} \text{skip} \{ P \} \]

- **Assignment:**
  \[ \vdash \{ P[t/x] \} x := t \{ P \} \]
  Example: Suppose \( t = x + 1 \) then, \( \{ x+1 = 2 \} x := x + 1 \{ x = 2 \} \)

- **("Forward" axiom):**
  \[ \vdash \{ P \} x := t \{ \exists x_0 [x_0/x] \top \land x = [x_0/x] t \} \]
  Example: \( \{ y=x \} x := x + 1 \{ y=x-1 \}; x_0 \text{ "stores" the old value of } x \)

- **Sequencing:**
  \[ \vdash \{ P \} c_0 \{ Q \}
  \vdash \{ Q \} c_1 \{ R \} \]
  \[ \vdash \{ P \} c_0 ; c_1 \{ R \} \]
Proof Rules (cont)

- **Conditionals:**
  \[ \vdash \{ P \land b \} c_0 \{ Q \} \quad \vdash \{ P \land \neg b \} c_1 \{ Q \} \]
  \[ \vdash \{ P \} \text{ if } b \text{ then } c_0 \text{ else } c_1 \{ Q \} \]

- **Loops:**
  \[ \vdash \{ P \land b \} \ c \ \{ P \} \]
  \[ \vdash \{ P \} \text{ while } b \text{ do } c \ \{ P \land \neg b \} \]

- **Consequence:**
  \[ \vdash (P \Rightarrow P') \quad \vdash \{ P' \} \ c \ \{ Q' \} \quad \vdash (Q' \Rightarrow Q) \]
  \[ \vdash \{ P \} \ c \ \{ Q \} \]

- If \( \vdash P \Rightarrow P' \) then all states \( \sigma \) which satisfy \( P \) also satisfy \( P' \). Rule allows strengthening of \( P' \) to \( P \) and weakening of \( Q' \) to \( Q \)
Example

- \( \{ x > 0 \} \ y = x - 1 \{ y \geq 0 \} \) implies
- \( \{ x > 10 \} \ y = x - 1 \{ y \geq -5 \} \)

- \( \{ x > 0 \} \ y = x - 1 \{ y \geq 0 \} \) and
- \( \{ y \geq 0 \} \ x = y \{ x \geq 0 \} \) implies
- \( \{ x > 0 \} \ y = x - 1; \ x = y \{ x \geq 0 \} \)

- Rule of consequence allows us to arrive at a precondition of true and postcondition of false
Example

- Prove the program:
  - \( z := 0; \)
  - \( n := y; \)
  - \( \text{while } n > 0 \text{ do} \)
  - \( z := z + x; \)
  - \( n := n - 1 \)
  - computes the product of \( x \) and \( y \) (assuming \( y \) is not negative).
Example

- Want to show the following:
  \[ \{ y \geq 0 \} \text{ <program> } \{ z = x \times y \} \]
  is valid.

- Key insight is picking the invariant for the while loop:

  \[ P = \{ z = x \times (y - n) \land n \geq 0 \} \]
Example

\[
\{ z = x \times (y - n) \land n \geq 0 \}\] \hspace{1cm} (13)

while \( n > 0 \) do \( z := z + x; \ n := n -1 \)

\{ z = x \times y \}

\[
z = x \times (y - n) \land n \geq 0 \land \neg (n > 0) \Rightarrow z = x \times y \] \hspace{1cm} (12)

(by definition of while: \{ P \} while \( b \) do \( c \) \{ P \land \neg b \})

\[
\{ z = x \times (y - n) \land n \geq 0 \}\]

while \( n > 0 \) do \( z := z + x; \ n := n -1 \)

\{ z = x \times (y - n) \land n \geq 0 \land \neg n > 0 \} \hspace{1cm} (11)

\[
\{ z = x \times (y - n) \land n \geq 0 \land n > 0 \}\]

\[
z := z + x; \ n := n -1 \]

\{ z = x \times (y - n) \land n \geq 0 \} \hspace{1cm} (10)

(by definition of while: premise is \{ P \land b \} c \{ P \})
Example

- \[ z = x \cdot (y - n) \land n \geq 0 \land n > 0 \Rightarrow \]
- \[(z + x) = x \cdot (y - (n - 1)) \land (n - 1) \geq 0 \quad (9)\]
- (rewrite; prepare replacing \( z+x \) by \( z \) and \( n-1 \) by \( n \))

\[
\{(z + x) = x \cdot (y - (n - 1)) \land (n - 1) \geq 0\}
\]
- \[ z := z + x; \; n := n - 1 \]
- \[ \{z = x \cdot (y - n) \land n \geq 0\} \quad (8) \]

\[
\{(z + x ) = x \cdot (y - (n - 1)) \land (n - 1) \geq 0\}
\]
- \[ z := z + x; \]
- \[ \{z = x \cdot (y - (n - 1)) \land (n - 1) \geq 0\}
\]
- \[ n := n - 1; \]
- \[ \{z = x \cdot (y - n) \land n \geq 0\} \quad (7) \]
Example

We have shown $A = \{ z = x \times (y - n) \land n \geq 0 \}$ while $n > 0$ do $z := z + x; n := n - 1 \{ z = x \times y \}$

Remains

\{ y \geq 0 \} z := 0; n := y \quad A = \{ z = x \times (y - n) \land n \geq 0 \} \quad (5)

Holds because $y \geq 0 \Rightarrow 0 = x \times (y - y) \land y \geq 0$:

\{ 0 = x \times (y - y) \land y \geq 0 \}
\{ z := 0; n := y \}
\{ z = x \times (y - n) \land n \geq 0 \} \quad (3)

\{ z = x \times (y - y) \land y \geq 0 \}
\{ n := y \}
\{ z = x \times (y - n) \land n \geq 0 \} \quad (2)

\{ 0 = x \times (y - y) \land y \geq 0 \}
\{ z := 0 \}
\{ z = x \times (y - y) \land y \geq 0 \} \quad (1)
Soundness and Completeness

- **Soundness**
  - Whenever $\vdash \{P\} \propto \{Q\}$ we do have $\models \{P\} \propto \{Q\}$
  - Hoare rules are sound

- **But is it true that whenever $\vdash \{P\} \propto \{Q\}$ we can also derive $\vdash \{P\} \propto \{Q\}$?**
  - If it isn’t that means there are valid properties of programs we cannot verify
  - We would like to automatically generate and verify all proofs
Gödel’s Incompleteness

- Take consequence rule
  - Allows for strengthening of preconditions and weakening of postconditions
    - How strong? How weak? How to prove?
- What invariant for while?
- It is logically impossible to have an effective proof system in which one can prove precisely the valid assertions
Weakest Preconditions

- Take sequence \{P\} c_0;c_1 \{Q\}
  - How to find R s.t. \{P\} c_0 \{R\} and \{R\} c_1 \{Q\}

- Idea: weakest precondition
  - “Bottom-up”, iteratively compute minimally necessary preconditions for postconditions under respective commands
  - \(wp[c, Q] = \{\sigma \in \Sigma \upsilon \mid C[c] \sigma \models Q\}\)
  - Thus \(\models \{P\} c \{Q\}\) iff \(P \subseteq wp[c, Q]\)
Weakest Preconditions

- **Skip:** \( \{ P \} \text{skip} \{ P \} \)
- \( wp(\text{skip}, P) = P \)

- **Assignment:** \( \{ P[t/x] \} x := t \{ P \} \)
- \( wp(x:=t, P) = \{ P[t/x] \} \)
Weakest Preconditions (cont)

- **Sequencing:** \[
\{P\} c_0 \{Q\} \quad \{Q\} c_1 \{R\}
\]

- \[
\{P\} c_0 ; c_1 \{R\}
\]

- \[
wp(c_0 ; c_1, R) = wp(c_0, wp(c_1, R))
\]

- **Conditionals:**

- \[
\{P\} c_0 \{Q\} \quad \{R\} c_1 \{Q\}
\]

- \[
\{b \Rightarrow P \land \neg b \Rightarrow R\} \text{ if } b \text{ then } c_0 \text{ else } c_1 \{Q\}
\]

- \[
wp(\text{if } b \text{ then } c_0 \text{ else } c_1) = b \Rightarrow wp(c_0, Q) \land \neg b \Rightarrow wp(c_1, Q)
\]
While Loops Again

- How about \{P\} while b do c \{Q\}? P?
- Remember
  - while b do c = if b then c; while b do c else skip
- Let w = while b do c and W = wp(w, Q)
- We have W= b \Rightarrow wp(c, W) \land \neg b \Rightarrow Q
  - Recursion
Using Domain Theory

- What assertion contains least information? True
- What is an appropriate information ordering? \( P \sqsubseteq P' \iff \models P' \Rightarrow P \)
- Is this partial order complete?
  - Take a chain \( P_0 \sqsubseteq P_1 \sqsubseteq \ldots \)
  - Let \( \bigwedge P_i \) be the infinite conjunction of \( P_i \)
  - \( \sigma \models \bigwedge P_i \) iff for all \( i \) we have that \( \sigma \models P_i \)
Fixed Point Theorem

- Use fixed point theorem for
  - \( F(P) = b \Rightarrow wp(c, P) \land \neg b \Rightarrow Q \)
  - Verify that \( F \) is continuous

- Least fixed point is
  - \( wp(w, Q) = \land F^n(true) \)

- Define family of \( wp \)'s
  - \( wp_n(while \ b \ do \ c, Q) = \) weakest precondition on which if loop terminates in \( n \) or fewer iterations, terminates in \( Q \)
    - \( wp_0 = \neg b \Rightarrow Q \)
    - \( wp_1 = b \Rightarrow wp(c, wp_0) \land \neg b \Rightarrow Q \)
  - \( wp(c, Q) = \land n \geq 0 wp_n = \Box \{wp_n | n \geq 0\} \)
With a Little Help...

- With a little human guidance assertions can be established a lot better
  - wp can in fact not always be computed
- Define the annotated commands of IMP as:
  - $c ::= \text{skip}$
  - $| x := e$
  - $| c_0; (x := e)$
  - $| c_0; \{R\} c_1$
  - $| \text{if } b \text{ then } c_0 \text{ else } c_1$
  - $| \text{while } b \text{ do } \{R\} c$
- An annotated partial correctness assertion is of the form $\{P\} c \{Q\}$ with $c$ defined as above
Check

- while loop means
  - \{P\} while b do \{R\} c \{Q\} with R invariant
  - Thus \{R \land b\} c \{R\}
  - \text{P }\Rightarrow \text{ R and } R \land \neg b \Rightarrow Q
  - Verifiable with Hoare consequence and while rules

- Verification conditions
  - Verify annotated partial correctness assertions
Verification Conditions

- $vc(\{P\} \text{ skip } \{Q\}) = \{P \Rightarrow Q\}$
- $vc(\{P\} X:=a \{Q\}) = \{P \Rightarrow Q[a/X]\}$
- $vc(\{P\} c_0; X:=a \{Q\}) = vc(\{P\} c_0 \{Q[a/X]\})$
- $vc(\{P\} c_0; \{R\} c_1 \{Q\}) = vc(\{P\} c_0 \{R\}) \cup vc(\{R\} c_1 \{Q\})$ (c_0 \text{ no assignm.})
- $vc(\{P\} \text{ if } b \text{ then } c_0 \text{ else } c_1 \{Q\}) =
  \begin{align*}
  &\quad vc(\{P \land b\} c_0 \{Q\}) \cup vc(\{P \land \neg b\} c_1 \{Q\})
  \end{align*}$
- $vc(\{P\} \text{ while } b \text{ do } \{D\} c \{Q\}) =
  \begin{align*}
  &\quad vc(\{P \land b\} c \{Q\}) \cup \{P \Rightarrow D\} \cup \{D \land \neg b \Rightarrow Q\}
  \end{align*}$