Agenda

• Meta-Language
• Definition: Formal Semantics
• Denotational Semantics
• A Toy Language
• Sample Semantics
• Axiomatic Semantics
• Rules of Program Reasoning
• Operational Semantics
Meta-Language

- **Meta-Language:** is a language used to define (or reason about) languages. E.g. English can be an informal meta-language to reason about Java. Or BNF can be used as a meta-language to specify rules for Java.

- Reminder of BNF: Backus-Naur Form (AKA BNF, AKA Backus-Normal Form) is a tool to specify syntax rules for a language. It is a quadruple \( \{ s, N, T, P \} \), with \( s \) being a start symbol, \( N \) a set of non-terminals, \( T \) a set of terminal symbols, and \( P \) a set of productions. Meta-Symbols typically are \( [ ] \), \( | \), and \( ::= \) or \( \rightarrow \).

- Other meta-languages can specify the meaning of a language whose syntax is understood.
Definition Formal Semantics

**Semantics:** the study of meaning of language, including programming languages. A common approach defining semantics is the informal use of “natural language”, e.g. English, in which case the natural language is the meta-language to specify some language. Or the approach to specify meaning is formal, using “concrete implementation” of a computer language. Formal semantics should define meaning precisely, completely, and concisely.

- **Denotational Semantics:** Describe meaning of a programming languages by constructing mathematical objects (called *denotations*) that in turn defined the meanings of expressions in the language.

- **Axiomatic Semantics:** Define the meaning of a command in a program by describing its effect on assertions about the program state. The assertions are logical statements, i.e. predicates with variables, where the variables define program state. See Toni Hoare’s correctness proofs for programming languages.

- **Operational Semantics:** Giving semantics in terms of the actual steps executed by some simple, precisely-defined abstract computer. See lambda calculus.
Denotational Semantics

Example: Define semantics for binary numbers, BNF as meta-language

Grammar: \[ B \rightarrow 0 \mid 1 \mid B\, 0 \mid B\, 1 \]  // defines syntax

Semantic Domain: \( N = \{ 0, 1, 2, \ldots \} \)

Semantic Function: \[ F : B = N \]

\[ F[0] = 0 \]
\[ F[1] = 1 \]
\[ F[B0] = 2 \times F[B] \]  // \( \times \) defines multiply
\[ F[B1] = 2 \times F[B] + 1 \]  // \( + \) defines addition

For a concreted number 1101, we have
\[ F[1101] = 2 \times F[110] + 1 \]
\[ = 2 \times (2 \times F[11] ) + 1 \]
\[ = 2 \times (2 \times (2 \times F[1] + 1 ) ) + 1 \]
\[ = 2 \times (2 \times (2 \times 1 + 1 ) ) + 1 \]
\[ = 13 \]
A Toy Language

- Below we use BNF to specify syntax, and common sense as meta-language to grasp semantics
- Meta-symbols are | and →

\[
\begin{align*}
P & \rightarrow \text{read } x; \ S; \ \text{write } E & & \text{// start symbol is } P \\
S & \rightarrow S; \ S \\
& \ | \ x := E \ | \ \text{to } E \ \text{do } S \\
& \ | \ (S) & & \text{// non-terminals: } P \ S \ E \\
E & \rightarrow 0 \ | \ x \ | \ y \ | \ \text{succ } E & & \text{// meta-symbol } \rightarrow \text{produce} \\
\end{align*}
\]

A Sample Program:

\[
\begin{align*}
\text{read } x; \\
y & := x; \\
\text{to } x \ \text{do } y & := \text{succ } y; \\
\text{write } y
\end{align*}
\]
Sample: Semantics

$$E \rightarrow 0 \mid 1 \mid -E \mid \text{not } E$$

$$\mid E + E \mid E = E \mid (E) \mid \text{id}$$

$$\mid \text{num} \mid \text{procedure } S$$

$$S \rightarrow \text{null} \mid \text{id := E} \mid \text{call } E \mid S; S \mid \text{begin } S \text{ end}$$

$$\mid \text{if } E \text{ then } S \text{ else } S \mid \text{while } E \text{ do } S$$

$$P \rightarrow \text{program } (\text{id}); S.$$
Axiomatic Semantics

Observation:
All properties of a program and all consequences of executing it in any environment can in principle, be derived from the source program text

Idea:
Reasoning about programs is based on axioms and rules of inference.

Notation: \{P\} S \{Q\}
“If the assertion P (precondition) is true before initiation of a program S, then the assertion Q (post-condition) will be true on its completion.”
If there is no precondition, assume=postulate: \{true\} S \{Q\}
If the claim can be proven in the formal system: \vdash \{P\} S \{Q\}
Axioms of Computer Arithmetic

**Basic Axioms:**

A1: \( x + y = y + x \) // commutativity for +
A2: \( x \times y = y \times x \) // commutativity for *
A3: \((x + y) + z = x + (y + z)\) // associativity
A4: \((x \times y) \times z = x \times (y \times z)\)
A5: \(x \times (y + z) = x \times y + x \times z\)
A6: \(y \leq x \Rightarrow (x - y) + y = x\)
A7: \(x + 0 = x\) // null element
A8: \(x \times 0 = 0\)
A9: \(x \times 1 = x\)

**Supplementary Axioms:** (System Dependent!)

A10\(_I\): \( \neg \exists x \forall y (y \leq x) \) // infinite arithmetic
A10\(_F\): \( \forall x (x \leq \text{max}) \) // finite arithmetic
A11\(_S\): \( \neg \exists x (x = \text{max} + 1) \) // strict interpretation
A11\(_B\): \( \text{max} + 1 = \text{max} \) // firm boundary
A11\(_M\): \( \text{max} + 1 = 0 \) // modulo arithmetic
Sample: Axiomatic Theorem

A theorem is a statement that can be derived from axioms.

Sample Theorem:

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

Proof:

\[
(r - y) + y \times (1 + q) \\
= (r - y) + (y \times 1 + y \times q) \quad // \ A5 \\
= (r - y) + (y + y \times q) \quad // \ A9 \\
= ((r - y) + y) + y \times q \quad // \ A3 \\
= r + y \times q \quad \text{provided } y \leq r \quad // \ A6
\]

Note: An infinite number of theorems can be derived from a finite set of axioms.
Rules of Program Reasoning

• **Axiom of Assignment:**
  D0: \{P(expr)\} x := expr \{P(x)\}
  The precondition \(P(expr)\) is obtained from the post-condition \(P(x)\) by substituting expr for all occurrences of \(x\).

• **Rule of Consequence:**
  D1: If \{P\}S\{Q\} and \(Q \implies R\) then \{P\}S\{R\}.
  D2: If \{P\}S\{Q\} and \(R \implies P\) then \{R\}S\{Q\}.

• **Rule of Composition:**
  D3: If \{P\}S1\{Q\} and \{Q\}S2\{R\} then \{P\}S1; S2\{R\}.

• **Rule of Iteration:**
  D4: If \{P \land B\}S\{P\} then \{P\} while B do S\{P \land \neg B\}.

• **Rule of Selection:**
  D5: If \{P \land B\}S1\{Q\} and \{P \land \neg B\}S2\{Q\}
  then \{P\} if B then S1 else S2\{Q\}.
Sample: Axiomatic Reasoning

Finding the quotient q and remainder r for x/y.
Program: \( r := x; \ q := 0; \ \text{while} \ y \leq r \ \text{do} \ (r := r - y; \ q := 1 + q) \)
Theorem: \{true\} S \{\neg y \leq r \ \land \ x = r + y \times q\}

Proof:
1. \( \text{true} \Rightarrow x = x + y \times 0 \) \hspace{1cm} \text{Lemma1}
2. \( (x = r + y \times q) \ \land \ y \leq r \Rightarrow x = (r - y) + y \times (1 + q) \) \hspace{1cm} \text{Lemma2}
3. \( \{x = x + y \times 0\} \ r := x \ \{x = r + y \times 0\} \) \hspace{1cm} \text{D0}
4. \( \{x = r + y \times 0\} \ q := 0 \ \{x = r + y \times q\} \) \hspace{1cm} \text{D0}
5. \( \{\text{true}\} \ r := x \ \{x = r + y \times 0\} \) \hspace{1cm} \text{D2,1,3}
6. \( \{\text{true}\} \ r := x; \ q := 0 \ \{x = r + y \times q\} \) \hspace{1cm} \text{D3,4,5}
7. \( \{x = (r - y) + y \times (1 + q)\} \ r := r - y \ \{x = r + y \times (1 + q)\} \) \hspace{1cm} \text{D0}
8. \( \{x = r + y \times (1 + q)\} \ q := 1 + q \ \{x = r + y \times q\} \) \hspace{1cm} \text{D0}
9. \( \{x = (r - y) + y \times (1 + q)\} \ r := r - y; \ q := 1 + q \ \{x = r + y \times q\} \) \hspace{1cm} \text{D3,7,8}
10. \( \{(x = r + y \times q) \ \land \ y \leq r\} \ r := r - y; \ q := 1 + q \ \{x = r + y \times q\} \) \hspace{1cm} \text{D2,2,9}
11. \( \{x = r + y \times q\} \ \text{while} \ y \leq r \ \text{do} \ (r := r - y; \ q := 1 + q) \ \{\neg y \leq r \ \land \ x = r + y \times q\} \) \hspace{1cm} \text{D4,10}
12. \( \{\text{true}\} \ r := x; \ q := 0; \ \text{while} \ y \leq r \ \text{do} \ (r := r - y; \ q := 1 + q) \ \{\neg y \leq r \ \land \ x = r + y \times q\} \) \hspace{1cm} \text{D3,6,11}
Operational Semantics

Operational semantics for a programming language describe how any particular valid program in the language is interpreted as a sequence of computational steps. This sequence then is the meaning of the program.

Example:

\[(\text{Fn } x \Rightarrow x + 2) \ (3 + 2 + 5)\]
\[\Rightarrow (\text{Fn } x \Rightarrow x + 2) \ (5 + 5)\]
\[\Rightarrow (\text{Fn } x \Rightarrow x + 2) \ (10)\]
\[\Rightarrow 10 + 2\]
\[\Rightarrow 12\]

Here computation is carried out by transforming (rewriting) the program text. Each \(-\) is an atomic “step” of computation, corresponding a semantic rule precisely defined for the involved operation.