The semantics of a programming language is concerned with the meaning of programs, that is, how programs behave when executed on computers.

The semantics of a programming language assigns a precise meaning to every sentence of the language that can be formed using the given syntax definition. There are three approaches to define the semantics of a programming language:
- Axiomatic semantics,
- Operational semantics,
- Denotational semantics.

The Hilbert-style Proof System
- A Hilbert-style proof system consists of axioms and proof rules.
  - An axiom of a proof system is a formula that is provable by definition.
  - An inference rule asserts that if some list of formulas is provable, then so is another formula.
  - A proof is a structured object built from formulas according to constraints established by a set of axioms and inference rules.
- The rule format:
  - Conclusion
    - Premise
    - Premise
    - Premise
  - Conclusion
  - Conclusion
  - Conclusion

Axiomatic Semantics
- The axiomatic semantics is a formal (proof) system for deriving equations between expressions.
- The basic idea of the axiomatic method is to define the meaning of language elements indirectly using logical assertions. For example, we can write \( (E_1) \implies (E_2) \), called a Hoare triple, to state that if the boolean expression \( E_1 \) holds prior the computation of \( C \), and if \( C \) terminates, then the boolean expression \( E_2 \) must hold as well.
- Examples:
  - \( \{ a > 0 \} \ x := a + 1 \ { a > 1 } \)
  - \( (E_1) \ C_1 \ (E_2) \)
  - \( (E_1) \ C_2 \ (E_2) \)
  - \( (E_1) \ C_3 \ C_3 \ (E_2) \)
### Example Rules

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Proof Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C =_{\text{def}} ) C.Target := C.Source</td>
<td>( {Q; C, \text{Source} \mid C, \text{Target} } ; L {Q } )</td>
</tr>
<tr>
<td>( C =_{\text{def}} ) if C.Test then C.Then else C.Else</td>
<td>( (C, \text{Test} \land P) ; C, \text{Then} ) ( {Q } ) ( (\neg C, \text{Test} \land P) ; C, \text{Else} ) ( {Q } )</td>
</tr>
</tbody>
</table>

### Rule of Consequence

\[
\begin{array}{c}
P \Rightarrow \{P \} \mid C \{Q' \} \quad Q' \Rightarrow Q \\
\end{array}
\]

### Example

Using axiomatic semantics, we need to prove the validity of a given Hoare triple:

Example:

\[
C =_{\text{def}} \begin{cases} 
\text{if } (a \geq b) \\
\text{then } m = a; \\
\text{else } m = b; \\
\{m = \max(a, b)\}
\end{cases}
\]

### Proof

Premise I:

\[
\begin{align*}
a \geq b & \Rightarrow a = \max(a, b) \\
\text{true} & \Rightarrow \{a = \max(a, b) ; m = a; \{m = \max(a, b)\} \} \\
(a \geq b) & \Rightarrow \{m = a; \{m = \max(a, b)\}\}
\end{align*}
\]

Premise II:

\[
\begin{align*}
a < b & \Rightarrow b = \max(a, b) \\
\text{true} & \Rightarrow \{b = \max(a, b) ; m = b; \{m = \max(a, b)\} \} \\
(a < b) & \Rightarrow \{m = b; \{m = \max(a, b)\}\}
\end{align*}
\]

\[
\begin{align*}
\{a \geq b \land \text{true}\} & \Rightarrow \{m = a; \{m = \max(a, b)\} \}
\end{align*}
\]

\[
\begin{align*}
\{a < b \land \text{true}\} & \Rightarrow \{m = b; \{m = \max(a, b)\} \}
\end{align*}
\]

\[
\begin{align*}
\text{true} & \Rightarrow \{m = \max(a, b)\}
\end{align*}
\]
Operational Semantics

- The operational semantics is based on a directed form of equational reasoning called "reduction". Reduction may be regarded as a form of symbolic evaluation.
- The basic idea of the operational method is to define the meaning of the language elements by means of a (labeled) transition system.
- The operational semantics definition provides means to display the computation steps undertaken when a program is evaluated to its output.
- Some forms of operational semantics are interpreted-based, with instruction counters, data structures, and the like, and others are inference rule-based, with proof trees that show control flows and data dependencies.

Example Transition Rules

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Transition Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>C =def C.Target := C.Source</td>
<td>σ(C.Source) → C.Target := C.Source → σ σ(C.Target, id)</td>
</tr>
<tr>
<td>C =def if C.Test then C.Then else C.Else</td>
<td>(\sigma(C.\text{Test} = \text{true}) \rightarrow \sigma(C.\text{Then}) \rightarrow \sigma') (\text{if } C.\text{Test} \text{ then } C.\text{Then} \text{ else } C.\text{Else} \rightarrow \sigma) (\sigma(C.\text{Test} = \text{false}) \rightarrow \sigma(C.\text{Else}) \rightarrow \sigma') (\text{if } C.\text{Test} \text{ then } C.\text{Then} \text{ else } C.\text{Else} \rightarrow \sigma)</td>
</tr>
</tbody>
</table>

Denotational Semantics

- The denotational semantics, or model theory, is defined in the spirit of equational logic or first-order logic. A denotational semantics definition (model) consists of a family of sets, one for each type, with the property that each well-typed expression may be interpreted as a specific element of the appropriate set.
- The denotational semantics is a recursive definition that maps well-typed derivation trees to their mathematical meanings. For example, the set Bool consists of two meanings: Bool = \{true, false\} and an operation \(\text{not} : \text{Bool} \rightarrow \text{Bool with } \text{not(false)} = \text{true}, \text{not(true)} = \text{false}\).
- The denotational method does not maintain states, but the meaning of a program is given as a function that interprets all language elements of a given program as elements of a corresponding set of values.
Example Meaning Functions

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Meaning Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C = \text{if } C.\text{Test} \text{ then } C.\text{Then} \text{ else } C.\text{Else}$</td>
<td>$f((C.\text{Target} \rightarrow C.\text{Source}, \sigma) = \begin{cases} f((C.\text{Target}, \sigma) - \sigma) &amp; \text{then } \sigma = f((C.\text{Target}, \sigma), f((C.\text{Source}, \sigma))) \ \text{else ERROR} &amp; \end{cases}$</td>
</tr>
<tr>
<td>$C = \text{if } C.\text{Test} \text{ then } C.\text{Then} \text{ else } C.\text{Else}$</td>
<td>$f((if C.\text{Test} \text{ then } C.\text{Then} \text{ else } C.\text{Else}, \sigma) = \begin{cases} f(C.\text{Test}, \sigma) &amp; \text{then } f(C.\text{Then}, \sigma) \ \text{else } f(C.\text{Else}, \sigma) &amp; \end{cases}$</td>
</tr>
</tbody>
</table>