Dynamic Semantics

Dynamic Semantics represents the intended meaning of language and language constructs

Dynamic Semantics describes and helps to understand what happens in a computer/machine when a program/model is executed

Describing Dynamic Semantics

- Implementation - Interpreter
- Translational
- Formal semantics
  - operational semantics
  - denotational semantics
  - axiomatic
  - algebraic
  - game semantics
  - …

Formal Dynamic Semantics

- Modern Systems models
  - complex
  - (possibly) high level of abstraction
- Reasoning about the models using rigorous methods
  - need to find existing ambiguities and inconsistencies
  - need to keep a (modeling) language “clean and simple”
- Formal semantics allows for
  - analysis, validation and verification, but also (correct) implementation
  - model/program comparison, thus optimization, modification
Operational Semantics

- Operational semantics specifies HOW (step-by-step) program/model executes
- Specifies HOW states/configurations are modified during the execution
- All possible executions are generated
- Underlying model is the model of transition systems (a program/model execution is turned into a transition system)

Example language - WHILE

Syntax (abstract) of WHILE

Arithmetic expressions (Aexp)
\[ a ::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \]

Boolean expressions (Bexp)
\[ b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid \neg b \mid b_1 \land b_2 \]

Statements (Stm)
\[ S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \]

Semantics of Expressions

- The meaning of an expression depends on the values assigned to the variables
- State as a basic concept in the operational semantics

States = \text{Var} \rightarrow \text{Int}

Example of a program: \[ z := x; x := y; y := z \]
\[ [x \mapsto 5, y \mapsto 7, z \mapsto 0] \in \text{States} \]
\[ s \ x = 5 \]
Semantics of Expressions

- For a given expression e and given state s
  \[ A: A_{\text{exp}} \rightarrow \text{State \rightarrow Int} \]
  \[ B: B_{\text{exp}} \rightarrow \text{State \rightarrow T} \]

- The semantics of arithmetic expressions
  \( A[n] s = n \)
  \( A[x] s = s \times x \)
  \( A[a1 + a2] s = A[a1] s + A[a2] s \)

Where \( n \) range over integers
\( x, y, z, \ldots \) range over set of variables \( \text{Var} \)
\( T = \{ \text{tt, ff} \} \)

Towards Statement Semantics

- The semantics of expressions DO NOT modify the state
- The semantics of statements DO modify the state
- OS specifies how the execution of a statement modify the state
- Modification are described by transition relation (transitions)
- Configuration \( \langle S, s \rangle \)
- Terminating configuration \( s \) (no more statements to execute)

Operational Semantics

- Two approaches
  - Natural (big-step) operational semantics : relation between the initial state and the final state of execution
    \[ \langle S, s \rangle \Rightarrow s' \]
  - Structural operational semantics (small-step) : each individual step of execution
    \[ \langle S, s \rangle \rightarrow \langle S', s' \rangle \text{ or } \langle S, s \rangle \rightarrow s' \]

Big-step Operational Semantics

- Relation between the initial state and the final state of execution
- Describes how an expression reduces to a value
  \[ \langle S, s \rangle \Rightarrow s' \]

Operational rules (general form)

\[ \langle S1, s1 \rangle \Rightarrow s'1, \langle S2, s2 \rangle \Rightarrow s'2, \ldots, \langle Sk, sk \rangle \Rightarrow s'k \]
\[ \langle S, s \rangle \Rightarrow s' \]

where \( S1, \ldots, Sk \) are direct composite statement of \( S \)
Big-step Semantics of WHILE

\[ \langle x := a, s \rangle \Rightarrow s [x \mapsto a] s \]

\[ \langle \text{skip}, s \rangle \Rightarrow s \]

\[ \langle S_1, s \rangle \Rightarrow s', \langle S_2, s' \rangle \Rightarrow s'' \]

\[ \langle S_1 ; S_2, s \rangle \Rightarrow s'' \]

\[ \frac{\langle S_1, s \rangle \Rightarrow s'}{\text{if } b \text{ then } S_1 \text{ else } S_2} \]

\[ \frac{\langle S_2, s \rangle \Rightarrow s'}{\text{if } b \text{ then } S_1 \text{ else } S_2} \]

Derivation tree for a statement S

Example:

S is \((z := x; x := y); y := z)\)

s0 is \([x \mapsto 5, y \mapsto 7, z \mapsto 0]\)

We can derive that \(\langle S, s0 \rangle \Rightarrow [x \mapsto 7, y \mapsto 5, z \mapsto 5]\)

Question: Deterministic semantics is

if \(\langle S, s \rangle \Rightarrow s'\) and \(\langle S, s \rangle \Rightarrow s''\) then \(s' = s''\).

Is the operational semantics of WHILE deterministic?
Structural Operational Semantics

- Two approaches
  - Natural (big-step) operational semantics: relation between the initial state and the final state of execution
    \[ \langle S, s \rangle \Rightarrow s' \]
  - Structural operational semantics (small-step): each individual steps of execution
    \[ \langle S, s \rangle \rightarrow \langle S', s' \rangle \text{ or } \langle S, s \rangle \rightarrow s' \]

Each assignment and test induce a state change

- Transition
  \[ \langle S, s \rangle \rightarrow c \]
  expresses the first step of the execution of statement \( S \) in state \( s \)
  - \( c \) can be a non-terminating configuration \( \langle S', s' \rangle \) or
  - \( c \) can be a terminating configuration \( s' \)

Operational rules (general form)

\[ \langle S_1, s_1 \rangle \rightarrow c_1, \langle S_2, s_2 \rangle \rightarrow c_2, \ldots, \langle S_k, s_k \rangle \rightarrow c_k \]
\[ \langle S, s \rangle \rightarrow c \]

where \( S_1, \ldots, S_k \) are direct composite statement of \( S \)

Structural Oper. Semantics of WHILE

- Derivation sequence for a statement \( S \)

Example: \( S \) is \( (z:=x;x:=y);y:=z \)
\( s_0 \) is \( [x\rightarrow 5, y\rightarrow 7, z\rightarrow 0] \)

- We can derive the (finite) sequence
  \[ \langle \langle z:=x;x:=y,y:=z, s_0 \rangle \rangle \rightarrow \langle \langle x:=y,y:=z, s_0[z\rightarrow 5] \rangle \rangle \rightarrow \langle \langle y:=z, s_0[z\rightarrow 5][x\rightarrow 7] \rangle \rangle \rightarrow \langle \langle ((s_0[z\rightarrow 5][x\rightarrow 7])[y\rightarrow 5] \rangle \rangle \]
- Each of these steps has a derivation tree (valid step)
Two Semantics of WHILE Statements

The meaning of statements is defined by a (partial) function

- Big-step (natural) operational semantics: \( (S, s) \Rightarrow s' \)
  \[
  \delta_b[S]s = \begin{cases} 
  s', & \text{if } (S, s) \Rightarrow s' \\
  \text{undef}, & \text{otherwise}
  \end{cases}
  \]

- Structural operational semantics (small-step): \( (S, s) \Rightarrow \ldots \Rightarrow s' \)
  \[
  \delta_s[S]s = \begin{cases} 
  s', & \text{if } (S, s) \Rightarrow \ldots \Rightarrow s' \\
  \text{undef}, & \text{otherwise}
  \end{cases}
  \]

The two semantics are equivalent: for every WHILE statement \( S \)

\[
\delta_b[S] = \delta_s[S]
\]

Example DSL for supervisory control

- Complex language
- Semantics defined by the implementation
- Different subjective interpretations of the same constructs

DLS in place

Supervisory control
- perform (optimal) allocation of activities
- to resources (subsystems) over time
**Formal Semantics in place**

- (domain) model
- DSL TReCS
- (implementation) engine
- formal semantics
- SOS

**Target model**

- Target formal language
- (verification) engine
- design
- decision improvement

---

**TReCS syntax**

- The smallest identifiable behavior (activity)
- A labeled node represents a task
- Set of labeled tasks: $\mathcal{T}$

- A unlabeled node represents a skip
TReCS Finish-start precedence relation

Syntax of Finish-start precedence relation:

\[ p \cdot q \]

Impact:

- Refine: \( T = T_\alpha \cup T_\omega \cup \{\tau}\)

\( t_\alpha \in T_\alpha \) : set of starting tasks

\( t_\omega \in T_\omega \) : set of finishing tasks

Syntax for Start-start precedence relation:

\[ p \parallel q \]
**TReCS Choice**

Impact:
- State of model
  \[ \sigma : \text{Var} \rightarrow \text{Doms} \]
- Selection function \( d \):
  \[ d : \Sigma \rightarrow \mathbb{N} \]

Syntax of Choice:
\[ \forall d(p_1, \ldots, p_n) \]

**TReCS Syntax**

- Selection function \( d \):
- Syntax of Choice:
  \[ \forall d(p_1, \ldots, p_n) \]

**TReCS Synchronized Parallelism**

- Resources can be claimed/released by tasks
- Set of resource labels: \( \mathcal{R} \)
- Required/Produced resources for a task:
  \[ \mathcal{R}_{\text{req}} \rightarrow (\mathcal{R} \rightarrow \text{Nat}) \]
  \[ \mathcal{R}_{\text{prod}} \rightarrow (\mathcal{R} \rightarrow \text{Nat}) \]

**TReCS Resources**
Structural Operational Semantics of TReCs

<table>
<thead>
<tr>
<th>Model (Term)</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (vector)</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>Configuration</td>
<td>$\langle p, \sigma \rangle$</td>
</tr>
<tr>
<td>Terminating configuration</td>
<td>$\langle \top, \sigma' \rangle$</td>
</tr>
<tr>
<td>Transition</td>
<td>$\langle p, \sigma \rangle \xrightarrow{\tau, R} \langle p', \sigma' \rangle$</td>
</tr>
<tr>
<td>Terminating transition</td>
<td>$\langle p, \sigma \rangle \xrightarrow{H} \langle \top, \sigma' \rangle$</td>
</tr>
</tbody>
</table>

Transition Rules

Structural Operational Semantics

$\langle \tau, \sigma \rangle \xrightarrow{\tau, \emptyset} \langle \top, \sigma \rangle$

\[ (\text{start-task}) \quad \sigma(R_A) \geq R_Q(t_{\alpha}) \]
\[ (t_{\alpha}^{i_{\alpha}}, \sigma) \xrightarrow{t_{\alpha}^{i_{\alpha}}} \langle t_{\omega}, \sigma[R_A \mapsto \sigma(R_A) - R_Q(t_{\alpha})] \rangle \]

\[ (\text{finish-task}) \quad \sigma(R_A) \geq \sigma(R_A) + R_P(t_{\omega}) \]

Structural Operational Semantics (cont.)

(FS) $\frac{\langle p, \sigma \rangle \xrightarrow{\beta, \rho} \langle p', \sigma' \rangle}{\langle p \cdot q, \sigma \rangle \xrightarrow{\beta, \rho} \langle p' \cdot q, \sigma' \rangle}$

(SS) $\frac{\langle p, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}{\langle p \| q, \sigma \rangle \xrightarrow{\beta, \rho} \langle p' \| q, \sigma' \rangle}$

(C) $\frac{\langle p_{d(\sigma)}, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}{\langle \forall d(p_1, \ldots, p_n, \sigma) \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle \mid d(\sigma) \in [1, n] \rangle}$

(spc) $\frac{\langle p \mid \text{sync}(p) \cap \text{sync}(q) \mid q, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}{\langle p \| q, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}$

(spe1) $\frac{\beta \in C, \langle p, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle, \langle q, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}{\langle p \mid C \mid q, \sigma \rangle \xrightarrow{\beta, \rho} \langle \top, \sigma' \rangle}$
Formal Analysis at reach

- (domain) model
- DSL TReCS
- (implementation engine)
- Target model
- Target formal language
- (verification) engine
- Formal semantics SOS
- Formal semantics SOS

Design decision improvement