50.054 Dynamic Semantics

ISTD, SUTD

Learning Outcomes

1. Explain the small step operational semantics of a programming language.
2. Explain the big step operational semantics of a programming language.
3. Formalize the run-time behavior of a programming language using small step operational semantics.
4. Formalize the run-time behavior of a programming language using big step operational semantics.

Dynamic Semantics

A formal specification of a program to define

- Run-time behavior of the program
- Returned result of the program

Recall

```
x = input;
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

PC	Memory
1	{input:1, x:1}
2	{input:1, x:1, s:0}
3	{input:1, x:1, s:0, c:0}
4	{input:1, x:1, s:0, c:0}
5	{input:1, x:1, s:0, c:0}
6	{input:1, x:1, s:0, c:1}
4	{input:1, x:1, s:0, c:1}
8	{input:1, x:1, s:0, c:1}

Different Types of Dynamic Semantics

- Operational Semantics - term rewriting
  - Small Step
  - Big Step
- Denotatoinal Semantics - mapping to math object
- Translational Semantics - translating to another language

Small Step Operational Semantics

- Term rewriting
- Rule $t_1 \longrightarrow t_2$
- Executing a terminating program

$$t_1 \longrightarrow_{rule_1} t_2 \longrightarrow \ldots \longrightarrow_{rule_n} t_n$$

- Executing a non-terminating program

$$t_1 \longrightarrow_{rule_1} t_2 \longrightarrow \ldots \longrightarrow_{rule_n} t_n \longrightarrow \ldots$$

Small Step Operational Semantics for Lambda Calculus

(NOR) $$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}$$

(β reduction) $$(\lambda x.t_1)\ t_2 \longrightarrow [t_2/x]t_1$$

(OpI1) $$\frac{t_1 \longrightarrow t_1'}{t_1\ op\ t_2 \longrightarrow t_1'\ op\ t_2}$$

(OpI2) $$\frac{t_2 \longrightarrow t_2'}{c_1\ op\ t_2 \longrightarrow c_1\ op\ t_2'}$$

(OpC) $$\frac{invoke\ low\ level\ call\ op(c_1, c_2) = c_3}{c_1\ op\ c_2 \longrightarrow c_3}$$

(ifI) $$\frac{t_1 \longrightarrow t_1'}{if\ t_1\ then\ t_2\ else\ t_3 \longrightarrow if\ t_1'\ then\ t_2\ else\ t_3}$$

(ifT) $$if\ true\ then\ t_2\ else\ t_3 \longrightarrow t_2$$

(ifF) $$if\ false\ then\ t_2\ else\ t_3 \longrightarrow t_3$$

(Let) $$let\ x = t_1\ in\ t_2 \longrightarrow [t_1/x]t_2$$

(Fix1) $$\frac{t \longrightarrow t'}{fix\ t \longrightarrow fix\ t'}$$

(Fix2) $$fix\ \lambda f.t \longrightarrow [(fix\ \lambda f.t)/f]t$$

SIMP

Syntax

$$
\begin{array}{rrcl}
\texttt{(Statement)} & S & ::= & X = E; \mid \mathit{return}\ X; \mid \mathit{nop}; \mid \mathit{if}\ E\ \{\overline{S}\}\ \mathit{else}\ \{\overline{S}\} \mid \mathit{while}\ E\ \{\overline{S}\} \\
\texttt{(Expression)} & E & ::= & E\ OP\ E \mid X \mid C \mid (E) \\
\texttt{(Statements)} & \overline{S} & ::= & S \mid S\ \overline{S} \\
\texttt{(Operator)} & OP & ::= & + \mid - \mid * \mid < \mid == \\
\texttt{(Constant)} & C & ::= & 0 \mid 1 \mid 2 \mid \ldots \mid \mathit{true} \mid \mathit{false} \\
\texttt{(Variable)} & X & ::= & a \mid b \mid c \mid d \mid \ldots
\end{array}
$$

Small step operational semantics

- Rule $(\Delta, S) \longrightarrow (\Delta', S')$
- Rule $\Delta \vdash E \longrightarrow E'$
- Δ is a mapping from variables to constant values.

Small Step Operational Semantics for SIMP

$$\text{(sVar)} \qquad \Delta \vdash X \longrightarrow \Delta(X)$$

$$\text{(sOp1)} \qquad \frac{\Delta \vdash E_1 \longrightarrow E_1'}{\Delta \vdash E_1 \ OP \ E_2 \longrightarrow E_1' \ OP \ E_2}$$

$$\text{(sOp2)} \qquad \frac{\Delta \vdash E_2 \longrightarrow E_2'}{\Delta \vdash C_1 \ OP \ E_2 \longrightarrow C_1 \ OP \ E_2'}$$

$$\text{(sOp3)} \qquad \frac{C_3 = C_1 \ OP \ C_2}{\Delta \vdash C_1 \ OP \ C_2 \longrightarrow C_3}$$

$$\text{(sParen1)} \qquad \frac{\Delta \vdash E \longrightarrow E'}{\Delta \vdash (E) \longrightarrow (E')}$$

$$\text{(sParen2)} \qquad \Delta \vdash (c) \longrightarrow c$$

Small Step Operational Semantics for SIMP

$$(\texttt{sAssign1}) \quad \frac{\Delta \vdash E \longrightarrow E'}{(\Delta, X = E;) \longrightarrow (\Delta, X = E';)}$$

$$(\texttt{sAssign2}) \quad \frac{\Delta' = \Delta \oplus (X, C)}{(\Delta, X = C;) \longrightarrow (\Delta', \mathit{nop})}$$

Small Step Operational Semantics for SIMP

$$\texttt{(sIf1)} \quad \frac{\Delta \vdash E \longrightarrow E'}{(\Delta, \mathit{if}\ \ E\ \{\overline{S_1}\}\ \mathit{else}\ \{\overline{S_2}\}) \longrightarrow (\Delta, \mathit{if}\ \ E'\ \{\overline{S_1}\}\ \mathit{else}\ \{\overline{S_2}\})}$$

$$\texttt{(sIf2)} \quad (\Delta, \mathit{if}\ \ \mathit{true}\ \{\overline{S_1}\}\ \mathit{else}\ \{\overline{S_2}\}) \longrightarrow (\Delta, \overline{S_1})$$

$$\texttt{(sIf3)} \quad (\Delta, \mathit{if}\ \ \mathit{false}\ \{\overline{S_1}\}\ \mathit{else}\ \{\overline{S_2}\}) \longrightarrow (\Delta, \overline{S_2})$$

Small Step Operational Semantics for SIMP

$$(\texttt{sWhile}) \quad (\Delta, \mathit{while}\ E\ \{\overline{S}\}) \longrightarrow (\Delta, \mathit{if}\ E\ \{\overline{S}; \mathit{while}\ E\ \{\overline{S}\}\}\ \mathit{else}\ \{\mathit{nop}\})$$

How about?

$$(\texttt{sWhile1}) \quad \frac{\Delta \vdash E \longrightarrow E'}{(\Delta, \mathit{while}\ E\ \{\overline{S}\}) \longrightarrow (\Delta, \mathit{while}\ E'\ \{\overline{S}\})}$$

$$(\texttt{sWhile2}) \quad (\Delta, \mathit{while}\ \mathit{true}\ \{\overline{S}\}) \longrightarrow (\Delta, \overline{S}; \mathit{while}\ ???\ \{\overline{S}\})$$

$$(\texttt{sWhile3}) \quad (\Delta, \mathit{while}\ \mathit{false}\ \{\overline{S}\}) \longrightarrow (\Delta, \mathit{nop})$$

Small Step Operational Semantics for SIMP

$$(\texttt{sNopSeq}) \qquad (\Delta, \mathit{nop}; \overline{S}) \longrightarrow (\Delta, \overline{S})$$

$$(\texttt{sSeq}) \qquad \frac{S \neq \mathit{nop} \quad (\Delta, S) \longrightarrow (\Delta', S')}{(\Delta, S\overline{S}) \longrightarrow (\Delta', S'\overline{S})}$$

Example 1

```
x = input;
x = x + 1;
return x;
---> # (sSeq)
   {(input,1)}, x = input
   ---> # (sAssign1)
   {(input,1)}, x = 1
   ---> # (sAssign2)
   {(input, 1), (x,1)}, nop
--->
{(input,1), (x,1)},
nop;
x = x + 1;
return x;
```

```
{(input,1), (x,1)},
nop;
x = x + 1;
return x;
---> # (sNopSeq)
{(input,1), (x,1)},
x = x + 1;
return x;
```

Example 1 (Cont'd)

```
{(input,1), (x,1)},
x = x + 1;
return x;
---> # (sSeq)
  {(input,1), (x,1)}, x = x + 1
  ---> # (sAssign1)
    {(input,1), (x,1)} |- x + 1
    ---> # (sOp1)
        {(input,1), (x,1)} |- x
        ---> # (sVar) 1
    {(input,1), (x,1)} |- 1 + 1
    ---> # (sOp3) 2
  {(input,1), (x,2)}, nop
--->
{(input,1), (x,2)},
nop;
return x;
```

```
{(input,1), (x,2)},
nop;
return x;
---> # (sNopSeq)
{(input,1), (x,2)},
return x;
```

Example 2

```
{(input, 1)},
x = input;
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
---> # (sSeq), (sAssign1), (sAssign2), (NopSeq)
{(input,1), (x,1)},
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

```
{(input,1), (x,1)},
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
---> # (sSeq), (sAssign2), (sNoSeq)
{(input,1), (x,1), (s,0)},
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

Example 2 (Cont'd)

```
{(input,1), (x,1), (s,0)},
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
---> # (sSeq), (sAssign2), (sNoSeq)
{(input,1), (x,1), (s,0), (c,0)},
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

sub derivation of the while statement

```
{(input,1), (x,1), (s,0), (c,0)},
while c < x {
    s = c + s;
    c = c + 1;
}
---> # (sWhile)
{(input,1), (x,1), (s,0), (c,0)},
if (c < x) {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s;
        c = c + 1;
    }
} else { nop; }
```

Example 2 Sub derivation of while (Cont'd)

```
{(input,1), (x,1), (s,0), (c,0)},
if (c < x) {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s;
        c = c + 1;
    }
} else { nop; }
---> # (sIf1), (sOp1), (sOp2), (sOp3)
{(input,1), (x,1), (s,0), (c,0)},
if true {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s;
        c = c + 1;
    }
} else { nop; }
```

```
{(input,1), (x,1), (s,0), (c,0)},
if true {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s;
        c = c + 1;
    }
} else { nop; }
---> # (sIf2)
{(input,1), (x,1), (s,0), (c,0)},
s = c + s;
c = c + 1;
while c < x {
    s = c + s;
    c = c + 1;
}
```

Example 2 Sub derivation of while (Cont'd)

```
{(input,1), (x,1), (s,0), (c,0)},
s = c + s;
c = c + 1;
while c < x {
    s = c + s;
    c = c + 1;
}
---> # (sSeq),(sAssign1),(sOp1-3),
     # (sAssign2),(sNopSeq)
{(input,1), (x,1), (s,0), (c,0)},
c = c + 1;
while c < x {
    s = c + s;
    c = c + 1;
}
---> # (sSeq),(sAssign1),(sOp1&3),
     # (sAssign2),(sNopSeq)
{(input,1), (x,1), (s,0), (c,1)},
while c < x {
    s = c + s;
    c = c + 1;
}
```

```
{(input,1), (x,1), (s,0), (c,1)},
while c < x {
    s = c + s;
    c = c + 1;
}
---> # (sWhile)
{(input,1), (x,1), (s,0), (c,1)},
if (c < x) {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s;
        c = c + 1;
    }
} else { nop }
---> # (sIf1), (sOp1-3)
{(input,1), (x,1), (s,0), (c,1)},
if false {
    s = c + s;
    c = c + 1;
    while c < x {
        s = c + s; c = c + 1;
    }
} else { nop } --->  nop;
```

Example 2 (Cont'd)

```
{(input,1), (x,1), (s,0), (c,1)},
nop;
return s;
---> # (sNopSeq)
{(input,1), (x,1), (s,0), (c,1)},
return s;
```

Big Step Operational Semantics

- Assume the program is terminating
- Following the syntactical structure
- Fast-forward the derivation until a normal form is reached.
- More concise

Big Step Operational Semantics for SIMP

- Rules for expression $\Delta \vdash E \Downarrow C$,
- Rules for statement $(\Delta, S) \Downarrow \Delta'$
- Δ maps variables to constant values.

Big Step Operational Semantics for SIMP

$$\text{(bConst)} \qquad \Delta \vdash c \Downarrow C$$

$$\text{(bVar)} \qquad \Delta \vdash X \Downarrow \Delta(X)$$

$$\text{(bOp)} \qquad \frac{\Delta \vdash E_1 \Downarrow C_1 \quad \Delta \vdash E_2 \Downarrow C_2 \quad C_1 \ OP \ C_2 = C_3}{\Delta \vdash E_1 \ OP \ E_2 \Downarrow C_3}$$

$$\text{(bParen)} \qquad \frac{\Delta \vdash E \Downarrow C}{\Delta \vdash (E) \Downarrow C}$$

Big Step Operational Semantics for SIMP

$$\text{(bAssign)} \quad \frac{\Delta \vdash E \Downarrow C}{(\Delta, X = E) \Downarrow \Delta \oplus (X, C)}$$

$$\text{(bIf1)} \quad \frac{\Delta \vdash E \Downarrow \mathit{true} \qquad (\Delta, \overline{S_1}) \Downarrow \Delta_1}{(\Delta, \mathit{if}\ \ E\ \{\overline{S_1}\}\ \ \mathit{else}\ \{\overline{S_2}\}) \Downarrow \Delta_1}$$

$$\text{(bIf2)} \quad \frac{\Delta \vdash E \Downarrow \mathit{false} \qquad (\Delta, \overline{S_2}) \Downarrow \Delta_2}{(\Delta, \mathit{if}\ \ E\ \{\overline{S_1}\}\ \ \mathit{else}\ \{\overline{S_2}\}) \Downarrow \Delta_2}$$

$$\text{(bWhile1)} \quad \frac{\Delta \vdash E \Downarrow \mathit{true} \qquad (\Delta, \overline{S}; \mathit{while}\ E\ \{\overline{S}\}) \Downarrow \Delta'}{(\Delta, \mathit{while}\ E\ \{\overline{S}\}) \Downarrow \Delta'}$$

$$\text{(bWhile2)} \quad \frac{\Delta \vdash E \Downarrow \mathit{false}}{(\Delta, \mathit{while}\ E\ \{\overline{S}\}) \Downarrow \Delta}$$

Big Step Operational Semantics for SIMP

$$(\texttt{bNop}) \qquad (\Delta, \mathit{nop}) \Downarrow \Delta$$

$$(\texttt{bSeq}) \qquad \frac{(\Delta, S) \Downarrow \Delta' \qquad (\Delta', \overline{S}) \Downarrow \Delta''}{(\Delta, S\overline{S}) \Downarrow \Delta''}$$

Example 3

```
{(input, 1)},
x = input;
x = x + 1;
return x;
```

Example 4

```
{(input, 1)},
x = input;
s = 0;
c = 0;
while c < x {
    s = c + s;
    c = c + 1;
}
return s;
```

Small vs Big

	Small step operational semantics	Big step operational semantics
mode	one step of change at a time	many steps of changes at a time
derivation	it is linear	it is a tree
cons	it is slow-paced and lengthy, requires more rules	it is a fast-forward version, requires fewer rules
pros	it is expressive, supports non-terminiating programs	it assumes the given program is terminating

Formal Results (Agreement between Small Step and Big Step Operational Semantics of SIMP)

Let $\overline{S}$ be a SIMP program, Δ be a memory environment. Then $\Delta, \overline{S} \Downarrow \Delta'$ iff $(\Delta, \overline{S}) \longrightarrow^* (\Delta', \mathit{return}\ X)$ for some X.

Summary

- Dynamic Semantics
- Operational Semantics
- Small step vs Big step
- Lambda Calculus
- SIMP
- Pseudo Assembly (refer to the notes)