Operational Semantics

程序可以分为语法和语义，形式化的语义无歧义地定义了程序应当具有的行为

操作语义定义了程序的执行：作为抽象机器的状态转换

Syntax

一个基本的命令式语言的语法包括

- 整数表达式：常量，变量，加减乘除
- 布尔表达式：常量，变量，关系运算
- 语句：赋值，分支，循环……

$$
\begin{array}{rrcl}
(\text{IntExp}) & e & ::= & \mathbf{n} \\
 & & \mid & x \\
 & & \mid & e + e \mid e - e \mid \ldots \\
 \\
(\text{BoolExp}) & b & ::= & \textbf{true} \mid \textbf{false} \\
 & & \mid & e = e \mid e < e \mid e > e \\
 & & \mid & \neg b \mid b \wedge b \mid b \vee b \mid \ldots \\
 \\
(\text{Comm}) & c & ::= & \textbf{skip} \\
 & & \mid & x := e \\
 & & \mid & c\, ; c \\
 & & \mid & \textbf{if } b \textbf{ then } c \textbf{ else } c \\
 & & \mid & \textbf{while } b \textbf{ do } c
\end{array}
$$

语法中的数字（numeral）不同于自然数（nature number），前者是在语法中对后者的描述（一个是语法层面一个是语义层面，一般记为 $\lfloor \mathbf{n} \rfloor = n$）

State

作为对程序运行过程的定义，最基本的就是要知道状态

定义状态为变量到值的映射

$$\sigma \in \text{Var} \rightarrow \text{Value}$$

一般会将状态 $\sigma_1 = \{(x,2),(y,3),(a,10)\}$ 写作 $\{x \rightsquigarrow 2, y \rightsquigarrow 3, a \rightsquigarrow 10\}$

同理可以改变状态

$$f\{x \rightsquigarrow n\} = \lambda z. \begin{cases} fz & z \neq x \\ n & z = x \end{cases}$$

操作语义的 configuration 即是操作和状态的结合：$(e,\sigma),(b,\sigma),(c,\sigma)$

Small step

Basis

SOS (Structural Operational Semantics)：语法是递归定义的，故语义也可以递归定义（根据语法的递归结构，由各部分的语义组成整体的语义）

如加法表达式就可以定义为

$$\frac{(e_1,\sigma) \longrightarrow (e_1',\sigma)}{(e_1+e_2,\sigma) \longrightarrow (e_1'+e_2,\sigma)} \qquad \frac{(e_2,\sigma) \longrightarrow (e_2',\sigma)}{(\mathbf{n}+e_2,\sigma) \longrightarrow (\mathbf{n}+e_2',\sigma)}$$

$$\frac{\lfloor \mathbf{n}_1 \rfloor \ \lfloor + \rfloor \ \lfloor \mathbf{n}_2 \rfloor \ = \ \lfloor \mathbf{n} \rfloor}{(\mathbf{n}_1+\mathbf{n}_2,\sigma) \longrightarrow (\mathbf{n},\sigma)}$$

减法同理，而变量的操作语义可以定义为

$$\frac{\sigma(x) = \lfloor \mathbf{n} \rfloor}{(x,\sigma) \rightarrow (\mathbf{n},\sigma)}$$

小步语义的定义就确定了操作符的结合性，如上述对加法的定义就确定加法为左结合

可以定义 $\rightarrow^*$ 表示零步或多步转换，关于 $\rightarrow$ 有如下两个性质

determinism：对于任意 $c,\sigma,c',\sigma',c'',\sigma''$，如果有 $(c,\sigma) \rightarrow (c',\sigma')$ 且 $(c,\sigma) \rightarrow (c'',\sigma'')$，则能推出有 $(c',\sigma') = (c'',\sigma'')$

confluence：对于任意 $c,\sigma,c',\sigma',c'',\sigma''$，如果有 $(c,\sigma) \rightarrow^* (c',\sigma')$ 且 $(c,\sigma) \rightarrow^* (c'',\sigma'')$，则存在 c''',σ''' 满足 $(c',\sigma') \rightarrow^* (c''',\sigma''')$ 且 $(c'',\sigma'') \rightarrow^* (c''',\sigma''')$

normalization：没有能无限推导的序列，执行路径最终都会停留在 normal form

- 对于算数表达式，为 $(\lfloor \mathbf{n} \rfloor, \sigma)$
- 对于布尔表达式，为 $(\lfloor \mathbf{true} \rfloor, \sigma), (\lfloor \mathbf{false} \rfloor, \sigma)$

但对于语句是不成立的，因为有无限循环（while true do skip）

可以对小步语义做一些变动

$$\frac{[\![e]\!]_{intexp}\,\sigma = n}{(x := e, \sigma) \longrightarrow (\mathbf{skip}, \sigma\{x \rightsquigarrow n\})}$$

$$\frac{(c_0,\, \sigma) \longrightarrow (c_0',\, \sigma')}{(c_0\,;c_1,\, \sigma) \longrightarrow (c_0'\,;c_1,\, \sigma')} \qquad \frac{}{(\mathbf{skip}\,;c_1,\, \sigma) \longrightarrow (c_1,\, \sigma)}$$

$$\frac{[\![b]\!]_{boolexp}\,\sigma = true}{(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\, \sigma) \longrightarrow (c_0,\, \sigma)}$$

$$\frac{[\![b]\!]_{boolexp}\,\sigma = false}{(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\, \sigma) \longrightarrow (c_1,\, \sigma)}$$

$$\frac{[\![b]\!]_{boolexp}\,\sigma = true}{(\mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma) \longrightarrow (c\,;\mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma)}$$

$$\frac{[\![b]\!]_{boolexp}\,\sigma = false}{(\mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma) \longrightarrow (\mathbf{skip},\, \sigma)}$$

此时删去原本对 skip 的规则，而是将 $(\mathbf{skip}, \sigma)$ 作为终止状态

可以在此变动的基础上扩展更多功能

Going wrong

引入新的 configuration：$\mathbf{abort}$，表示运行时的错误，如除零，访问不存在的数据

Assignment:

$$\frac{[\![e]\!]_{intexp}\,\sigma = n}{(x := e, \sigma) \longrightarrow (\textbf{skip}, \sigma\{x \rightsquigarrow n\})} \qquad \frac{[\![e]\!]_{intexp}\,\sigma = \bot}{(x := e, \sigma) \longrightarrow \textbf{abort}}$$

Here

$$\begin{array}{lll} [\![e]\!]_{intexp}\,\sigma = n & \text{iff} & (e, \sigma) \longrightarrow^{*} (\mathbf{n}, \sigma) \text{ and } n = \lfloor \mathbf{n} \rfloor \\ [\![e]\!]_{intexp}\,\sigma = \bot & \text{iff} & (e, \sigma) \longrightarrow^{*} \textbf{abort} \end{array}$$

这里要区分 going wrong 和 getting stuck，前者是错误，而后者是不能继续执行，如在变动后的语义上 skip 就属于 getting stuck

Local variable declaration

添加新的语句 $\textbf{newvar}\ x := e\ \textbf{in}\ c$，表示声明局部变量

因为要考虑局部变量的可见性，故给出的语义较为复杂

Solution (due to Eugene Fink):

$$\frac{n = [\![e]\!]_{intexp}\,\sigma \qquad (c, \sigma\{x \rightsquigarrow n\}) \longrightarrow (c', \sigma') \qquad \sigma'\,x = \lfloor \mathbf{n}' \rfloor}{(\textbf{newvar}\ x := e\ \textbf{in}\ c,\ \sigma) \longrightarrow (\textbf{newvar}\ x := \mathbf{n}'\ \textbf{in}\ c',\ \sigma'\{x \rightsquigarrow \sigma\,x\})}$$

$$\frac{}{(\textbf{newvar}\ x := e\ \textbf{in}\ \textbf{skip},\ \sigma) \longrightarrow (\textbf{skip},\ \sigma)}$$

$$\frac{[\![e]\!]_{intexp}\,\sigma = \bot}{(\textbf{newvar}\ x := e\ \textbf{in}\ c,\ \sigma) \longrightarrow \textbf{abort}}$$

$$\frac{n = [\![e]\!]_{intexp}\,\sigma \qquad (c, \sigma\{x \rightsquigarrow n\}) \longrightarrow \textbf{abort}}{(\textbf{newvar}\ x := e\ \textbf{in}\ c,\ \sigma) \longrightarrow \textbf{abort}}$$

Heap

对于动态分配的数据，可以将状态分为 store 和 heap，其中 store 是变量到值的映射，值包括常量和地址，而 heap 便是地址到值的一个 partial mapping（i.e. 有些地址没有对应的值）

可以在语句中添加新的动态操作

$$
\begin{array}{lcll}
c & ::= & \ldots & \\
 & | & x := \textbf{alloc}(e) & \text{allocation} \\
 & | & y := [x] & \text{lookup} \\
 & | & [x] := e & \text{mutation} \\
 & | & \textbf{free}(x) & \text{deallocation}
\end{array}
$$

contextual semantic

表达式的执行可以提取出一个模板

$$\frac{(r, \sigma) \to (e', \sigma)}{(\mathcal{E}[r], \sigma) \to (\mathcal{E}[e'], \sigma)}$$

其中 r 称为 redex，可以是变量或是常数的表达式（n+n，n-n）

$\mathcal{E}$ 称为 evaluation context，$\mathcal{E} ::= [] \mid \mathcal{E} + e \mid \mathcal{E} - e \mid \mathbf{n} + \mathcal{E} \mid \mathbf{n} - \mathcal{E}$

可以用 redex 和 evaluation context 表示小步操作语义，将操作分为两个部分

- 对哪个部分运算（redex）
- 何时能运算（evaluation context）

redex 是能在一步内运算的表达式或者命令

evaluation context 是有“洞”的 term，指明了下一步运算的位置，$\mathcal{E}[r]$ 即是用 r 替换了空洞得到的表达式，这个过程是递归进行的，如 $e_1 + e_2$ 作为 $\mathcal{E}[r]$

- 如果 $e_1 = \mathbf{n}_1, e_2 = \mathbf{n}_2$，则 $r = \mathbf{n}_1 + \mathbf{n}_2, \mathcal{E} = []$
- 如果 $e_1 = \mathbf{n}_1$，则将 e_2 变为 $\mathcal{E}_2[r]$，而 $\mathcal{E} = \mathbf{n}_1 + \mathcal{E}_2$

对于非 skip 的 command 或非 numeral 的 expression，总存在唯一的分解 $\mathcal{E}[r]$，故运算过程可以总结为

- 分解当前 term，得到 redex 和 evaluation context
- 运算 redex
- 将运算结果填回 evaluation context，从而得到新 term

$$\frac{(r, \sigma) \to (t, \sigma')}{(\mathcal{E}[r], \sigma) \to (\mathcal{E}[t], \sigma')}$$

对于布尔运算也是同样的道理

可以将 contextual semantics 中的 hole 看作 pc，但是因为每步都要分解，故实现并不高效

Big step

大步语义是对结果的总结

$$\frac{(e_1,\sigma) \Downarrow n_1 \qquad (e_2,\sigma) \Downarrow n_2}{(e_1 \ \mathbf{op} \ e_2,\sigma) \Downarrow n_1 \lfloor \mathbf{op} \rfloor n_2}$$

$$\frac{(e,\sigma) \Downarrow n}{(x := e,\sigma) \Downarrow \sigma\{x \rightsquigarrow n\}} \qquad \frac{}{(\mathbf{skip},\sigma) \Downarrow \sigma}$$

$$\frac{(c_0,\sigma) \Downarrow \sigma' \qquad (c_1,\sigma') \Downarrow \sigma''}{(c_0 \, ; c_1,\sigma) \Downarrow \sigma''} \qquad \frac{(b,\sigma) \Downarrow \mathit{true} \qquad (c_0,\sigma) \Downarrow \sigma'}{(\mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, \ \sigma) \Downarrow \sigma'}$$

$$\frac{(b,\sigma) \Downarrow \mathit{false} \qquad (c_1,\sigma) \Downarrow \sigma'}{(\mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, \ \sigma) \Downarrow \sigma'} \qquad \frac{(b,\sigma) \Downarrow \mathit{false}}{(\mathbf{while} \ b \ \mathbf{do} \ c, \ \sigma) \Downarrow \sigma}$$

$$\frac{(b,\sigma) \Downarrow \mathit{true} \qquad (c,\sigma) \Downarrow \sigma' \qquad (\mathbf{while} \ b \ \mathbf{do} \ c, \ \sigma') \Downarrow \sigma''}{(\mathbf{while} \ b \ \mathbf{do} \ c, \ \sigma) \Downarrow \sigma''}$$

同样有一些关于 $\Downarrow$ 的性质

determinism：对于任意 e,σ,n,n'，如果 $(e,\sigma) \Downarrow n$ 且 $(e,\sigma) \Downarrow n'$，则 $n = n'$

totality：对于任意 e,σ，存在 n 满足 $(e,\sigma) \Downarrow n$

大步语义和小步语义的联系在于 $(e,\sigma) \Downarrow \lfloor \mathbf{n} \rfloor \iff (e,\sigma) \to^* (\mathbf{n},\sigma)$

布尔表达式的大步语义同理，而对于语句来说，有

$$(c,\sigma) \Downarrow \mathbf{abort} \iff (c,\sigma) \to^* \mathbf{abort}$$
$$(c,\sigma) \Downarrow \sigma' \iff (c,\sigma) \to^* (\mathbf{skip},\sigma')$$

循环不会终结的情况下不能应用 while 的规则

大步语义更接近递归解释器，可以更快地证明