Notes from Semantics and verification of programs

Jacek Olczyk

October 2018

Part I
Notes from tutorials by Lorenzo Clemente

1 Small step semantics - continuation

1.1 Recap

- Global environments $\rho \vdash e \to e'$
- $\frac{\rho[x \to n] \vdash e \to e'}{\rho \vdash \text{ let } x = \underline{n} \text{ in } e \to \text{ let } x = \underline{n} \text{ in } e'}$

1.2 Local environments

- How do we define the semantics for 'let $x = e$ in f' expressions using local environments? More precisely, we need e to have its own environment, so that its evaluation doesn't affect the environment of f, as is the case with global environments.
- We are given the following 2 rules:
- $\frac{}{(\rho, x) \to (\rho, \rho(x))}$
- $\frac{}{(\rho, \text{ let } x = \underline{n} \text{ in } e) \to (\rho[x \to n], e)}$
- Now we need to give a rule for evaluating let expressions where a non-numeric expression is assigned to x.
- $\frac{(\rho, e) \to (\rho', e')}{(\rho, \text{ let } x = e \text{ in } f) \to ((\rho? \text{ or maybe } \rho'?), \text{ let } x = e' \text{ in } f)}$
- ρ doesn't work, because then a nested let in expression can't change the value of their variables.

- Neither does ρ', because then we don't get our original environment back at the end.
- Solution: new construct
- e then $x = n$
- Now we have:
- $\frac{(\rho,e)\to(\rho',e')}{(\rho,e \text{ then } x=\underline{n})\to(\rho',e' \text{ then } x=\underline{n})}$
- $\frac{}{(\rho,\underline{m} \text{ then } x=\underline{n})\to(\rho[x\to\underline{n}],\underline{m})}$
- $\frac{}{(\rho, \text{let } x=\underline{n} \text{ in } e)\to(\rho[x\to\underline{n}],e \text{ then } x=\rho(x))}$

2 Imperative language

- Syntax
- $C ::=$ Skip $|X := e|C;C|$if b then c else $c|$while b do c
- $e ::= n|x|e+e$
- $b :: true|false|e \leq e|\neg b|b \wedge b$
- $E[[e]]_s \in \mathbb{Q}$, $B[[b]]_s \in \{true, false\}$
- $s \in State = Var \to \mathbb{Q}$
- Configurations
- $(c, s) \in C$
- $s \in C$ (final)
- Small step rules for C - expressions
- $\frac{}{(Skip,s)\to s}$
- $\frac{}{(x:=e,s)\to s[x\to E[[e]]_s}$
- $\frac{(c,s)\to s'}{(c;d,s)\to(d,s')}$
- $\frac{(c,s)\to(c',s')}{(c;d,s)\to(c';d,s')}$
- $\frac{B[[b]]_s=true}{(\text{if } b \text{ then } c \text{ else } d,s)\to(c,s)}$
- $\frac{B[[b]]_s=false}{(\text{if } b \text{ then } c \text{ else } d,s)\to(d,s)}$
- $\frac{B[[b]]_s=true}{(\text{while } b \text{ do } c,s)\to(c;\text{while } b \text{ do } c,s)}$

- $\frac{B[[b]]_s = false}{(\text{while } b \text{ do } c, s) \to s}$
- Adding "Repeat c until b"
- $\frac{}{(\text{Repeat } c \text{ until } b, s) \to (c; \text{if } b \text{ then } Skip \text{ else Repeat } c \text{ until } b, s)}$

3 Numbers as strings of bits

- Evaluate:
- $n ::= \$0 | \$1 | n0 | n1 | n + n$
- final configurations: numbers without "+", e.g. \$100101
- $n \to n'$
- $\frac{n \to n'}{n0 \to n'0}$
- $\frac{n \to n'}{n1 \to n'1}$
- $\frac{m \to m'}{m + n \to m' + n}$
- $\frac{n \to n'}{m + n \to m + n'}$
- $\frac{}{m0 + n0 \to (m + n)0}$
- $\frac{}{m0 + n1 \to (m + n)1}$
- $\frac{}{m1 + n0 \to (m + n)1}$
- $\frac{}{m1 + n1 \to (m + n + \$1)0}$
- Fill in the last 4
- I think we should add a rule to merge two doll

4 Next time

Add to the syntax:

- for x:=e to e do c
- do e times c
- do c while e

5 TODO przepisanie z zeszytu

6 Loop, continue and break

$$C ::= \ldots | loop\ c | continue | break$$

Last time we did small steps semantics using $c\ then\ d$ statements. Now we want big steps:

$$\frac{c, s \to \ldots}{loop\ c, s \to} \frac{}{continue, s \to} \frac{}{break, s \to \ldots}$$

We can change the set of configurations by adding to the existing set of final configurations pairs $(state, flag)$ where $flag \in \{CNT, BRK\}$, thus:

$$\frac{c, s \to s', loop\ c, s' \to s''}{loop\ c, s \to s''} \frac{c, s \to (s', CNT), loop\ c, s' \to s''}{loop\ c, s \to s''} \frac{c, s \to (s', BRK)}{loop\ c, s \to s'}$$

$$\frac{c, s \to s', (d, s') \to s''}{c; d, s \to s''}, \hat{s} \in \{s'', (s'', CNT), (s'', BRK)\} \frac{c, s \to (s', f)}{c; d, s \to (s', f)}$$

7 Expressions with side effects

The syntax is as follows:

$$C ::= Skip | x := e | c; c$$

$$e ::= x | n | e + e | c\ resultis\ e$$

Old rules:

$$\frac{}{n, s \to \underline{n}}$$

$$\frac{}{x, s \to s(x)}$$

$$\frac{e, s \to \underline{m} \quad f, s \to \underline{n}}{e + f, s \to \underline{\underline{m + n}}}$$

$$\frac{}{Skip, s \to s}$$

$$\frac{e, s \to \underline{n}}{x := e, s \to s[x \mapsto n]}$$

New rules:

$$\frac{c, s \to s' \quad e, s' \to \underline{n}}{c\ resultis\ e, s \to \underline{n}}$$

Buut, this doesn't propagate the state change from inside the expressions! To fix this, we change the meaning of $\to$ for expressions by making it go to a pair $(number, state)$. Here are the modified old rules for addition and *resultis*:

$$\frac{e, s \to \underline{m}, s' \quad f, s \to \underline{n}, s''}{e + f, s \to \underline{\underline{m + n}}, s''}$$

$$\frac{c, s \to s' \quad e, s' \to \underline{n}, s''}{c\ resultis\ e, s \to \underline{n}, s''}$$

8 Let in expressions with lazy evaluation

Previously we had 'call by value' semantics for let in expressions, now we want 'call by name' semantics, which evaluate the variable assignment only when its value is needed. In CBV, we had $\frac{}{s \vDash x \to s(x)}$. How do we write semantics for *let* in CBN?

$$\frac{s[x \mapsto e] \vDash f \to m}{s \vDash let\ x = e\ in\ f \to m}$$

$$\frac{s \vDash s(x) \to n}{s \vDash x \to \underline{n})}$$

But this is dynamic binding, the environment used is whatever was at the moment of evaluation. To get static binding, we need variables to record state alongside the expressions: $St = Var \to (Expr \times St) \cup \mathbb{Q}$. But this is not a definition, just a recursive equation! Thus, let $St_0 = \emptyset$ and $St_{i+1} = Var \to (E \times St_i \cup \mathbb{Q})$. And the whole state is defined like this: (A set of russian dolls with arbitrary nesting) $St = \bigcup_{i=0}^{\infty} St_i$ Now we can get to the rules with static binding:

$$\frac{s(x) = (e, s') \quad s' \vDash e \to n}{s \vDash x \to \underline{n}}$$

$$\frac{s[x \mapsto e, s] \vDash f \to m}{s \vDash let\ x = e\ in\ f \to m}$$