Lecture 2

Multistep evaluation relation

This lecture closely followed *Software foundations*, Vol. 2, on Small Step Operational Semantics.

Given a binary relation $\rightarrow$ we define its reflexive transitive closure by the rules

$$\frac{}{e \rightarrow^* e} \qquad \frac{e \rightarrow e_1 \quad e_1 \rightarrow^* e_2}{e \rightarrow^* e_2}$$

Theorem 0.1 *$\rightarrow^*$ is reflexive, transitive and contains $\rightarrow$. It is the least reflexive transitive relation which contains $\rightarrow$.*

The next result was already in Frege's 1879 book (which introduced quantifiers and proof system for higher order logic).

Theorem 0.2 *If $\rightarrow$ is deterministic and $e \rightarrow^* e_1$ and $e \rightarrow^* e_2$ then $e_1 \rightarrow^* e_2 \vee e_2 \rightarrow^* e_1$*

We define $NF(e, e_1)$ to mean $e \rightarrow^* e_1$ and $\neg\exists e' \ (e_1 \rightarrow e')$

Theorem 0.3 *If $\rightarrow$ is deterministic and $NF(e, e_1)$ and $NF(e, e_2)$ then $e_1 = e_2$*

Intuitively $NF(e, e_1)$ means that e_1 is the result of the computation of e. The Theorem states that if the one step eveluation relation is deterministic then the result of the computation is uniquely determined (if it exists).

In the particular case of arithmetic expressions

$$e \ ::= \ v \mid \mathsf{add} \ e \ e \qquad\qquad v \ ::= \ \mathsf{const} \ n$$

where

$$n \ ::= \ 0 \mid \mathsf{succ} \ n$$

We can define the value as a function from expressions to natural numbers

$$[\![\mathsf{const} \ n]\!] = n \qquad [\![\mathsf{add} \ e_0 \ e_1]\!] = [\![e_0]\!] + [\![e_1]\!]$$

We have described leftmost evaluation by the rules

$$\frac{}{\mathsf{add} \ (\mathsf{const} \ n_0) \ (\mathsf{const} \ n1) \rightarrow \mathsf{const} \ (n_0 + n_1)} (C)$$

$$\frac{e_0 \rightarrow e_0'}{\mathsf{add} \ e_0 \ e_1 \rightarrow \mathsf{add} \ e_0' \ e_1} (A_0) \qquad \frac{e_1 \rightarrow e_1'}{\mathsf{add} \ (\mathsf{const} \ n) \ e_1 \rightarrow \mathsf{add} \ (\mathsf{const} \ n) \ e_1'} (A_1)$$

We have seen that this is a deterministic evaluation relation. In this case, the evaluation of any expression terminates.

Theorem 0.4 *We have* $\forall e \exists e_1 \; NF(e, e_1)$. *Actually* $\forall e \; NF(e, \mathsf{const}([\![e]\!]))$.

The proof is by induction on e, using the following.

Lemma 0.5 *If* $e \to^* e'$ *then* $\mathsf{add}\ e\ e_1 \to^* \mathsf{add}\ e'\ e_1$ *and* $\mathsf{add}\ v\ e \to^* \mathsf{add}\ v\ e'$.

We can define in a similar way Boolean expressions

$$e \ ::= \ v \mid \mathsf{if}\ e\ e\ e \qquad\qquad v \ ::= \ \mathsf{const}\ b$$

where

$$b \ ::= \ \mathsf{true} \mid \mathsf{false}$$

and the evaluation rules are

$$\frac{}{\mathsf{if}\ (\mathsf{const\ true})\ e_0\ e_1 \to e_0} \qquad \frac{}{\mathsf{if}\ (\mathsf{const\ false})\ e_0\ e_1 \to e_1}$$

$$\frac{e \to e'}{\mathsf{if}\ e\ e_0\ e_1 \to \mathsf{if}\ e'\ e_0\ e_1}$$

Theorem 0.6 *The relation* $\to$ *is deterministic and we have* $\forall e \exists e' \; NF(e, e')$.

A simple abstract machine and compiler correctness proof

What we present is a simplified version of the fundamental paper of McCarthy and Painter on correctness of a compiler for arithmetic expressions (1967).

We define the instruction list (code) as

$$cd \ ::= \ \mathsf{LOAD}\ n\ cd \mid \mathsf{ADD}\ cd \mid \mathsf{HALT}$$

and the compilation function is

$$comp\ (\mathsf{const}\ n)\ cd = \mathsf{LOAD}\ n\ cd \qquad comp\ (\mathsf{add}\ e_0\ e_1)\ cd = comp\ e_1\ (comp\ e_0\ (\mathsf{ADD}\ cd))$$

The machine has then for state a pair cd, S where cd is a code and S is a stack of numbers. The small step semantics for this machine is

$$\frac{}{\mathsf{ADD}\ cd,\ n_1 : n_0 : S \ \to \ \ cd, (n_1 + n_0) : S} \qquad \frac{}{\mathsf{LOAD}\ n\ cd, S \ \to \ \ cd, n : S}$$

We can now state, and prove by induction on e

Theorem 0.7 *For all expression* e *we have* $\forall cd\ \forall S \qquad comp\ e\ cd, S \ \to^* \ \ cd, [\![e]\!] : S$