Lecture 1

Arithmetic expression

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

The expressions are

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

where

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

Expressions form a type exp. We can define the value as a function from expressions to natural numbers

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

But we also can define function that refers to the *syntactic* form of an expression, for instance

$$depth\ (\mathsf{const}\ n) = 0 \qquad depth\ (\mathsf{add}\ e_0\ e_1) = 1 + max\ (depth\ e_0)\ (depth\ e_1)$$

We describe leftmost evaluation by the rules

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

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

We say that $e \to e'$ if it is the conclusion of a *derivation tree* using these primitive inference rules.

This defines a *one step evaluation* relation.

This defines a *binary relation* on expressions.

To simplify, we write simply n instead of $\mathsf{const}\ n$. The relation $e \to e'$ can be chracterised as being the *least* relation $R(e, e')$ satisfying the conditions

- $C_1 = \forall n_0\ n_1\ R(\mathsf{add}\ n_0\ n_1, n_0 + n_1)$
- $C_2 = \forall e_0\ e_0'\ e_1\ R(e_0, e_0') \Rightarrow R(\mathsf{add}\ e_0\ e_1, \mathsf{add}\ e_0'\ e_1)$
- $C_3 = \forall n\ e_1\ e_1'\ R(e_1, e_1') \Rightarrow R(\mathsf{add}\ n\ e_1, \mathsf{add}\ n\ e_1')$

It is a good exercice in Agda to show that if $R(e, e')$ satisfies these three conditions then we have $R(e, e')$ whenever $e \to e'$. This amounts to define a function of type

$$\Pi(e\ e' : \mathsf{exp})(p : e \to e')\ R(e, e')$$

by structural induction on p.

Lemma 0.1 *If $e \to e'$ then $e \neq \mathsf{const}\ n$ for all n.*

Proof. We define $R(e, e')$ by $\forall n\ e \neq \mathsf{const}\ n$ and we can check that this relation satisfies the three C_1, C_2, C_3. □

A binary relation R is said to be *deterministic* iff we have

$$\forall\ e\ e'\ e''\ \ (R\ e\ e'\ \wedge\ R\ e\ e'') \Rightarrow e' = e''$$

Theorem 0.2 *The relation defined by the rule C, A_0, A_1 is deterministic.*

Proof. We see as defining a function which takes as argument a proof p of $e \to e'$ and a proof q of $e \to e''$ and produces a proof of $e' = e''$. We then do the proof by case analysis of p and q and by structural induction on p and q.

The first case is if p is directly the axiom (C). This implies that e is of the form $\mathsf{add}\ (\mathsf{const}\ n_0)\ (\mathsf{const}\ n_1)$ and e' is $\mathsf{const}\ (n_0 + n_1)$. Then we can see using the previous Lemma that q has to be (C) as well and so e'' has to be $\mathsf{const}\ (n_0 + n_1)$ as well. This concludes the analysis of the first case.

The other cases are exercises. □

Another way to state this result is that the rule

$$\frac{e \to e' \qquad e \to e''}{e' = e''}$$

is *admissible.*

We define a predicate on expressions: e is a *value* iff e is of the form $\mathsf{const}\ n$. We can describe the expressions as follows

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

and the rule (A_1) can be rewritten as

$$\frac{e_1 \to e_1'}{\mathsf{add}\ v\ e_1 \to \mathsf{add}\ v\ e_1'}(A_1)$$

Theorem 0.3 *(strong progress) For all e we have that either e is a value or $\exists\ e'\ e \to e'$.*

We say that e is in *normal form* iff we have $\neg \exists e'\ e \to e'$. It is direct to see that if e is a value then e is in normal form. The second exercise is to use strong progress to prove the following.

Theorem 0.4 *An expression is a value iff it is in normal form.*