2. Constants and Primitive Operations. Basic (Dynamic) Types

① Recall the syntax of the lambda calculus and the notion of canonical form.

$$\langle exp\rangle ::= \langle var\rangle \mid \langle exp\rangle\ \langle exp\rangle \mid \lambda\langle var\rangle.\ \langle exp\rangle$$

$$\langle cfm\rangle ::= \langle funcfm\rangle$$

$$\langle funcfm\rangle ::= \lambda\langle var\rangle.\ \langle exp\rangle.$$

Here we expressed the notion of canonical forms using the abstract grammar.

We also recall the rules for evaluation; where we omit E in $\Rightarrow_E$ since we are interested in eager evaluation only here.

Canonical forms.

$$\frac{}{z \Rightarrow z}$$

(z is a canonical form).

β_E - evaluation

$$\frac{e \Rightarrow \lambda v.e'' \quad e' \Rightarrow z' \quad e''/_{v \to z'} \Rightarrow z}{e\,e' \Rightarrow z}$$

(z, z' are canonical forms. In general, we use z with primes, subscripts or superscripts to denote canonical forms.).

② Adding constants, primitive operations, and those for basic type constructors (tuple., alternative/sum).

amounts to extending ⟨exp⟩, ⟨cfm⟩, and ⇒.

Extending ⟨cfm⟩ with more cases intuitively means that . we make the language support values other than lambda expressions. The extension of ⇒ . encodes the meanings of newly introduced constants and operations.

③ We add four new kinds of values. That is, we change the grammar for ⟨cfm⟩ as follows:

integers

booleans

$$\langle cfm\rangle ::= \langle funcfm\rangle \mid \langle intcfm\rangle \mid \langle boolcfm\rangle \mid \langle tuplecfm\rangle \mid \langle altcfm\rangle.$$

tuples of canonical forms

canonical forms with tags

$$\langle intcfm\rangle ::= \ldots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \ldots$$

$$\langle boolcfm\rangle ::= \mathrm{true} \mid \mathrm{false}$$

$$\langle tuplecfm\rangle ::= \underline{\langle}\,\langle cfm\rangle, \ldots, \langle cfm\rangle\,\underline{\rangle}$$

$$\langle altcfm\rangle ::= \underline{ⓐ\ \langle tag\rangle}\ \langle cfm\rangle$$

$$\langle tag\rangle ::= 0 \mid 1 \mid 2 \mid \ldots$$

This extension means that expressions in this language denote five different kinds of values. Another important point is that we only include operations for constructing tuples and alternatives., not those for destructing them; such as projection and case operation. In a sense, this comes from the fact that destructors , represent unfinished computation and we regard something as value or canonical form when it represents completed computation.

④ Next we extend ⟨exp⟩ with appropriate constants and operations so that we can write expressions denoting computations with those newly introduced canonical forms, i.e. integers., booleans, tuples and alternatives.