$$\frac{e \Rightarrow z}{@\ k\ e \Rightarrow @\ k\ z}$$

$$\frac{e \Rightarrow @\ k\ z \qquad e_k\ z \Rightarrow z'}{\text{sumcase } e \text{ of } (e_0, \cdots, e_{n-1}) \Rightarrow z'} \quad \text{when } k<n.$$

3. Recursion.

① We include letrec:

$$\langle exp \rangle ::= \ldots \mid \text{letrec } \langle var \rangle \equiv \lambda \langle var \rangle . \langle exp \rangle \text{ in } \langle exp \rangle .$$

(letrec $v = \lambda w.e$ in e') defines a recursive function v and performs e' with v bound to this recursive function.

Thus,

$$FV(\text{letrec } v = \lambda w.e \text{ in } e') = \left(\left(FV(e) \setminus \{w\} \right) \cup FV(e') \right) \setminus \{v\}.$$

This means that the occurrence of v in e denotes $\lambda w.e$, the recursive function being defined here, not the value of a free variable v.

② This construct imposes two important constraints. First, recursively defined entities should be functions, like $\lambda w.e$. For instance, we can't do

$$\text{letrec } v \equiv \langle 1, v \rangle \text{ in } e$$

which defines an infinite tuple $v = \langle 1, \langle 1, \langle 1, \langle 1, \cdots \rangle \rangle \cdots \rangle$

Second, the RHS of $\equiv$ should be a canonical form. For instance, the following is not allowed.

$$\text{letrec } v \equiv (\lambda x.(\lambda y.3))\ (\lambda z.z) \text{ in } v$$

Both restrictions are included because we use the eager evaluation. In a programming language based on the normal-order evaluation such as Haskell, we don't need to impose those restrictions. When we discuss denotational semantics, you will understand where these restrictions come from.

③ Since adding letrec doesn't add a new kind of denotable values by expressions, we don't change ⟨cfm⟩. (Although we don't show, letrec can be expressed using lamda expressions and applications). But we need to add a rule for evaluating letrec expressions. Here is the rule:

unrolling of recursive call v in $\lambda w.e$

$$\frac{e' / v \to (\lambda w.e / v \to \text{letrec } v \equiv \lambda w.e \text{ in } v) \Rightarrow z}{\text{letrec } v \equiv \lambda w.e \text{ in } e' \Rightarrow z}$$

execution of e' with v bound to its definition.

④ Programming exercise.

Suppose that we represent binary trees with integer leaves using alternative and tuple as follows.

- $@\ 0\ n$ (for a terminal node (or leaf) labelled by the integer n.)
  integer
- $@\ 1\ \langle l, r \rangle$ (for a nonterminal node with left subtree l and right subtree r)

Write a program that sums all the integers in a given tree.