1 Lecture

1.1 Data

- Products: positional, sequences, lists, named, nonstrict, streams.
- Sums: positional, named, tagged.
- Sums-of-products: composite design pattern, dynamic dispatch, double dispatch.

1.2 Continuations

A computation can be viewed as an iteration over currently evaluated expression and the continuation of the current expression. Thanks to enhanced control of the program flow, continuations can be used to return multiple values, nonlocal exits, error-handling and backtracking.

$$multiply : R^* \to R \tag{1}$$
$$multiply(\langle\rangle) = 1 \tag{2}$$
$$multiply(\langle a_1, a_2, \ldots, a_n\rangle) = a_1 \cdot multiply(\langle a_2, \ldots, a_n\rangle) \quad \text{if } a_1 \neq 0 \tag{3}$$
$$multiply(\langle a_1, \ldots, a_n\rangle) = 0 \quad \text{otherwise} \tag{4}$$

$$multiplyCPS : R^* \to R \tag{5}$$
$$multiplyCPS(\langle a_1, \ldots, a_n\rangle) = mult(\langle a_1, \ldots, a_n\rangle, \lambda x.x) \tag{6}$$

$$mult : R^* \times (R \to R) \to R \tag{7}$$
$$mult(\langle\rangle, k) = k(1) \tag{8}$$
$$mult(\langle a_1, a_2, \ldots, a_n\rangle, k) = mult(\langle a_2, \ldots, a_n\rangle, \tag{9}$$
$$\lambda x.k(a_1 \cdot x)) \quad \text{if } a_1 \neq 0 \tag{10}$$
$$mult(\langle a_1, \ldots, a_n\rangle, k) = 0 \quad \text{if } a_1 = 0 \tag{11}$$

$$A^\perp = A^* \cup \{\perp\} \tag{12}$$
$$m : RegExp \times A^\perp \times (A^\perp \to A^\perp) \to A^\perp \tag{13}$$
$$m(r, \perp, k) = k(\perp) \tag{14}$$
$$m(\epsilon, \langle\rangle, k) = k(\langle\rangle) \tag{15}$$
$$m(\epsilon, \langle a_1, \ldots, a_n\rangle, k) = k(\langle a_1, \ldots, a_n\rangle) \tag{16}$$
$$m(a, \langle\rangle, k) = k(\perp) \tag{17}$$
$$m(a, \langle a_1, a_2, \ldots, a_n\rangle, k) = k(\langle a_2, \ldots, a_n\rangle) \quad \text{if } a = a_1 \tag{18}$$
$$m(a, \langle a_1, \ldots, a_n\rangle, k) = k(\perp) \quad \text{if } a \neq a_1 \tag{19}$$
$$m(r_1 \cdot r_2, \langle a_1, \ldots, a_n\rangle, k) = m(r_1, \langle a_1, \ldots, a_n\rangle, \lambda x.m(r_2, x, k)) \tag{20}$$
$$\begin{aligned} m(r_1 + r_2, \langle a_1, \ldots, a_n\rangle, k) = m(r_1, \langle a_1, \ldots, a_n\rangle, \\ \lambda x.\text{if } k(x) = \langle\rangle \text{ then } \langle\rangle \text{ else } m(r_2, \langle a_1, \ldots, a_n\rangle, k)) \end{aligned} \tag{21}$$

$$match : RegExp \times A^* \to Boolean \tag{22}$$
$$match(r, \langle a_1, \ldots, a_n\rangle) = true \quad \text{if } m(r, \langle a_1, \ldots, a_n\rangle, \lambda x.x) = \langle\rangle \tag{23}$$
$$match(r, \langle a_1, \ldots, a_n\rangle) = false \quad \text{otherwise} \tag{24}$$

2 Seminar

1. Implement the regexp matching code in Java, use the composite pattern.

   See the Dropbox folder.

2. Add Kleene star to the regexp matching code.

$$m(r^*, \langle a_1, \ldots, a_n\rangle, k) = m(\epsilon + (r \cdot r^*), \langle a_1, \ldots, a_n\rangle, k) \tag{25}$$

3 Homework

Your task will be to implement translation from an extended version of Featherweight Java to plain Featherweight Java (FJ). The extended FJ differs from plain FJ in three aspects: i) it has built-in support for numbers and booleans, ii) it has a fairly standard set of control flow statements and iii) it has mutable local variables.