1 Lecture

1.1 State

- Purely functional languages (and math) are stateless.
- Stateful computation can be represented as an iteration over states.

$$\begin{aligned} max &: N^* \to N \\ max(\langle a_1, \ldots, a_n \rangle) &= loop(\langle a_1, \ldots, a_n \rangle, 1, 0) \end{aligned} \tag{1}$$

$$\begin{aligned} loop &: N^* \times N \times N \to N \\ loop(\langle a_1, \ldots, a_n \rangle, c, m) &= m \quad \text{if } c > n \\ loop(\langle a_1, \ldots, a_n \rangle, c, m) &= loop(\langle a_1, \ldots, a_n \rangle, c+1, m) \quad \text{if } c \leq n \wedge a_c \leq m \\ loop(\langle a_1, \ldots, a_n \rangle, c, m) &= loop(\langle a_1, \ldots, a_n \rangle, c+1, a_c) \quad \text{otherwise} \end{aligned} \tag{2}$$

- Monadic style separates state-handling code.

$$State = N^* \times N \times N \tag{3}$$

$$Action = State \to State \tag{4}$$

$$Condition = State \to Boolean \tag{5}$$

$$\begin{aligned} updateMax &: Action \\ updateMax(\langle a_1, \ldots, a_n \rangle, c, m) &= (\langle a_1, \ldots, a_n \rangle, c, a_c) \end{aligned} \tag{6}$$

$$\begin{aligned} updateNeeded &: Condition \\ updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) &= true \quad \text{if } a_c > m \\ updateNeeded(\langle a_1, \ldots, a_n \rangle, c, m) &= false \quad \text{otherwise} \end{aligned} \tag{7}$$

$$\begin{aligned} increaseIndex &: Action \\ increaseIndex(\langle a_1, \ldots, a_n \rangle, c, m) &= (\langle a_1, \ldots, a_n \rangle, c+1, m) \end{aligned} \tag{8}$$

$$\begin{aligned} finished &: Condition \\ finished(\langle a_1, \ldots, a_n \rangle, c, m) &= true \quad \text{if } c > n \\ finished(\langle a_1, \ldots, a_n \rangle, c, m) &= false \quad \text{otherwise} \end{aligned} \tag{9}$$

$$\begin{aligned} ifStatement &: Condition \times Action \to Action \\ ifStatement(cond, body) &= \lambda s.body(s) \quad \text{if } cond(s) = true \\ ifStatement(cond, body) &= \lambda s.s \quad \text{otherwise} \end{aligned} \tag{10}$$

$$\begin{aligned} forLoop &: Condition \times Action \times Action \to Action \\ forLoop(cond, iter, body) &= \lambda s.ifStatement(cond, \\ &\qquad forLoop(cond, iter, body)(iter(body(s)))) \end{aligned} \tag{11}$$

$$\begin{aligned} max &: N^* \to N \\ max(\langle a_1, \ldots, a_n \rangle) &= \pi_3(forLoop(finished, increaseIndex, \\ &\qquad ifStatement(updateNeeded, updateMax)) \\ &\qquad (\langle a_1, \ldots, a_n \rangle, 1, 0)) \end{aligned} \tag{12}$$

- A program fragment is said to be *referentially transparent* if it has the same meaning regardless of its context.

1.2 Naming

- Parameter passing: call-by-name, call-by-value, call-by-denotation.
- Static scope.

```
function f(int a) {

    function g(int b) {
        return a + b;
    }

    return a + g(3);
}
```

- Multiple namespaces.

```
class X {
    int x;

    X(int x) {
        this.x = x;
    }

    int x() { return x; }
}
```

- Dynamic scope.

2 Seminar

1. Define function $switch : (State \to N) \times (N \to Action) \to Action$.

$$switch(expr, bodies) = \lambda s.bodies(expr(s))(s) \tag{13}$$

2. Prove correctness of the max function.

Formally, we need to prove that

$$\begin{aligned} \forall \langle a_1, \ldots, a_n \rangle \in N^* : max(\langle a_1, \ldots, a_n \rangle) \in \{a_1, \ldots, a_n\} \wedge \\ \forall i \in \{1, \ldots, n\} : a_i \leq max(\langle a_1, \ldots, a_n \rangle). \end{aligned} \tag{14}$$

Proof: by induction over the length of the input sequence. The inductive step needs to prove the following:

$$\begin{aligned} \forall \langle a_1, \ldots, a_n \rangle \in N^* : max(\langle a_1, \ldots, a_n \rangle) \in \{a_1, \ldots, a_n\} \wedge \\ \forall i \in \{1, \ldots, n\} : a_i \leq max(\langle a_1, \ldots, a_n \rangle) \\ \Downarrow \\ \forall \langle b_1, \ldots, b_{n+1} \rangle \in N^* : max(\langle b_1, \ldots, b_{n+1} \rangle) \in \{b_1, \ldots, b_{n+1}\} \wedge \\ \forall i \in \{1, \ldots, n+1\} : b_i \leq max(\langle b_1, \ldots, b_{n+1} \rangle). \end{aligned} \tag{15}$$

3. Define the denotational semantics of an imperative language with the following syntax:

$$\begin{aligned} Expr ::= {} & Num \mid \\ & Bool \mid \\ & \triangle Expr \mid \\ & Expr \odot Expr \mid \\ & \texttt{!} VarName \\ Statement ::= {} & VarName \texttt{ = } Expr \\ Program ::= {} & Statement\texttt{;}\ Program \mid \\ & \epsilon \end{aligned} \tag{16}$$

$$[\![n]\!] = \lambda env.n \tag{17}$$
$$[\![b]\!] = \lambda env.b \tag{18}$$
$$[\![\triangle]\!] = \lambda e.\lambda env. - e(env) \tag{19}$$
$$[\![\odot]\!] = \lambda e_1, e_2.\lambda env.e_1(env) + e_2(env) \tag{20}$$
$$[\![v]\!] = v \tag{21}$$
$$[\![\texttt{!}]\!] = \lambda v.\lambda env.env(v) \tag{22}$$
$$[\![\texttt{=}]\!] = \lambda v, e.\lambda env.env[v \mapsto e(env)] \tag{23}$$
$$[\![\texttt{;}]\!] = \lambda s, p.\lambda env.p(s(env)) \tag{24}$$
$$[\![\epsilon]\!] = \lambda env.env \tag{25}$$