Functional Programming Languages

Programming Languages CS442

David Toman

School of Computer Science
University of Waterloo

Core of a Functional Language

- Types: $bool, int, \tau list$
- Syntax and Typing Rules:

$$\textbf{true} : bool \qquad \textbf{false} : bool \qquad \frac{E : bool \quad E_1 : \tau \quad E_2 : \tau}{\textbf{if}\ E\ \textbf{then}\ E_1\ \textbf{else}\ E_2\ \textbf{fi} : \tau}$$

$$N : int \qquad \frac{E_1 : int \quad E_2 : int}{E_1 + E_2 : int} \qquad \frac{E_1 : int \quad E_2 : int}{E_1 = E_2 : bool}$$

$$\textbf{nil} : \tau list \qquad \frac{E_1 : \tau \quad E_2 : \tau list}{\textbf{cons}\ E_1\ E_2 : \tau list} \qquad \frac{E : \tau list}{\textbf{hd}\ E : \tau} \qquad \frac{E : \tau list}{\textbf{tl}\ E : \tau list} \qquad \frac{E : \tau list}{\textbf{null}\ E : bool}$$

$\Rightarrow$ what happened to loops? left for recursive abstractions

Reductions

- Booleans (values $\{\textbf{true}, \textbf{false}\}$):

$$\textbf{if}\ \textbf{true}\ \textbf{then}\ u\ \textbf{else}\ v\ \textbf{fi} \to u$$
$$\textbf{if}\ \textbf{false}\ \textbf{then}\ u\ \textbf{else}\ v\ \textbf{fi} \to v$$

- Integers (values $\{0, 1, 2, \ldots\}$):

$$m + m \to p \textit{ where } p \textit{ is the sum of values } m \textit{ and } n$$
$$m = m \to \textbf{true}$$
$$m = n \to \textbf{false}$$

- Lists (what are “values” here???)

$$\textbf{hd}(\textbf{cons}\ E_1\ E_2) \to E_1 \qquad \textbf{null}(\textbf{nil}) \to \textbf{true}$$
$$\textbf{tl}(\textbf{cons}\ E_1\ E_2) \to E_2 \qquad \textbf{null}(\textbf{cons}\ E_1\ E_2) \to \textbf{false}$$

$$\mathit{value}(\tau\mathit{list}) = \{\textbf{nil}\} \cup \{\textbf{cons}\ E_1\ E_2 \mid E_1 \in \mathit{value}(\tau), E_2 \in \mathit{value}(\tau\mathit{list})\}$$

Abstraction and Qualification

- Syntax:

$$\frac{\pi \vdash E : \tau}{\pi \vdash \textbf{val}\, I = E : \{I : \tau\}} \qquad \frac{\pi \vdash E_1 : \pi_1 \qquad E_2 : \pi_2}{\pi \vdash E_1, E_2 : \pi_1 \dot\cup \pi_2}$$

$$\frac{\pi \vdash E_1 : \pi_1 \qquad \pi \uplus \pi_1 \vdash E_2 : \tau}{\pi \vdash \textbf{let}\, E_1\, \textbf{in}\, E_2 : \tau} \qquad \frac{(I : \tau) \in \pi}{\pi \vdash I : \tau} \qquad \frac{\pi \uplus \{I : \tau\} \vdash E : \tau}{\pi \vdash \textbf{rec}\, I.E : \tau}$$

- Reductions:

$$\textbf{let}\,\textbf{val}\, I_1 = E_1, \ldots, \textbf{val}\, I_k = E_k\, \textbf{in}\, E \to [E_1/I_1, \ldots, E_k/I_k]E$$

$$\textbf{rec}\, I.E \to [\textbf{rec}\, I.E/I]E$$

What are “values” now? i.e., can we “substitute” non-values?

Parametrization

- Syntax:

$$\frac{\pi \uplus \{x : \tau\} \vdash E : \tau'}{\pi \vdash \lambda x.E : \tau \to \tau'} \qquad \frac{\pi \vdash E_1 : \tau \to \tau' \quad \pi \vdash E_2 : \tau}{\pi \vdash (E_1\ E_2) : \tau'}$$

- Reductions:

$$(\lambda I : \tau . E) E_2 \to [E_2/I]E$$

What are "values" here?

$$value(\tau \to \tau') = \{\lambda I : \tau . E \mid \pi \vdash \lambda I : \tau . E : \tau \to \tau' \text{ for some } \pi\}$$

Denotational Semantics

- most of the language = simply typed λ-calculus w/names

  $\Rightarrow$ same semantic equations (cf. soundness of β and β-val)

  $\Rightarrow$ no "store" to trigger evaluation!
- we need *improper* fail and $\perp$ values

  $\Rightarrow$ impacts the eager/lazy issue!
- but what is the meaning of lists?

Eager	finite lists of τ values
Lazy	finite-or-infinite lists of τ values (or "fail", $\perp$)

$\Rightarrow$ what does $[\![\mathbf{rec}\, X : \mathit{int\ list} \,.\, \mathbf{cons}\, 0\ X]\!]$ mean?

Summary

- Functional languages = everything is an *expression*

  $\Rightarrow$ no *store* and side-effects caused by assignments

  $\Rightarrow$ loops *usually* realized by recursion

  $\Rightarrow$ often has *complex data-types* (a.k.a., lists)

- Eager/Lazy issues *still* around

  $\Rightarrow$ determined by designating *values* for all types

  $\Rightarrow$ impacts *data* (e.g., lists) as well as *functions*

- Questions:
  1. can *failure* be handled using exceptions?
  2. what can we do about those annoying type tags?