CS 565

Programming Languages (graduate)

Spring 2024

Week 6

A Core Language - IMP

Defining a Language

One "recipe" for defining a language:

1. Syntax:

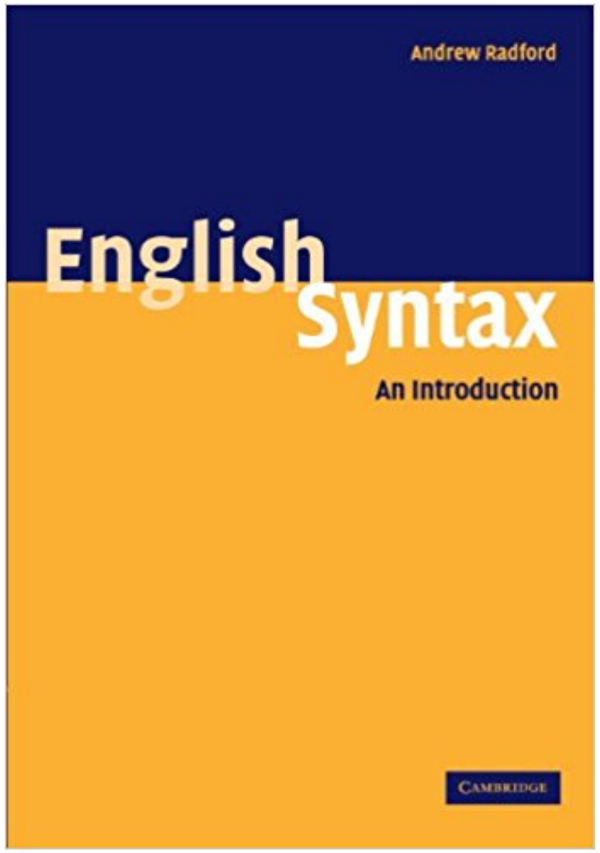
What are the valid sentences?

2. Semantics (Dynamic Semantics):

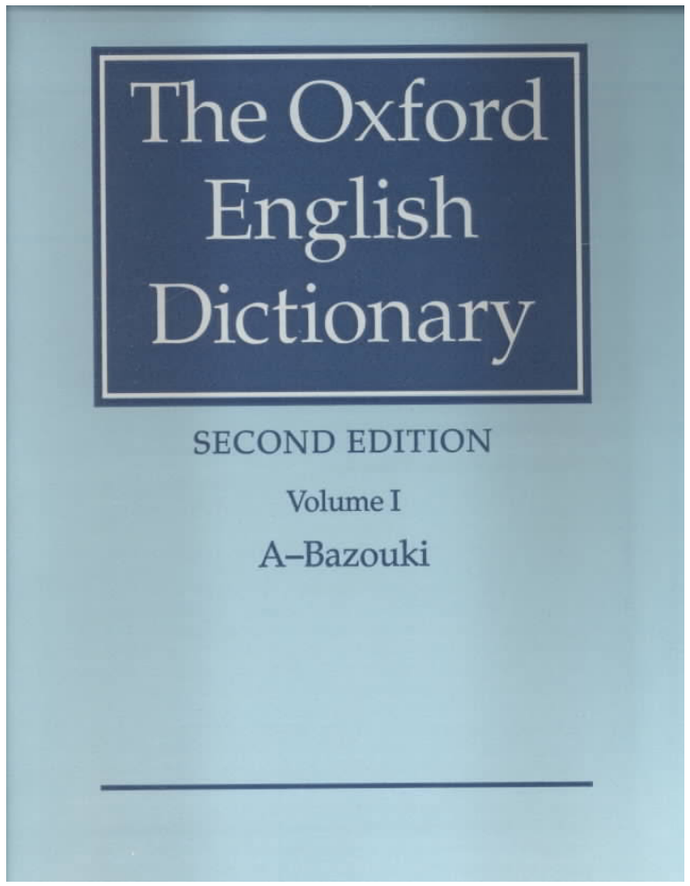
How do I evaluate valid sentences?

Defining English

1. Syntax:



2. Semantics



Defining Haskell

1. Syntax:

topdecl → type simpletype = type
| data [context =>] simpletype [= constrs] [deriving]
| newtype [context =>] simpletype = newconstr [deriving]
| class [scontext =>] tycls tyvar [where cdecls]
| instance [scontext =>] qtycls inst [where idecls]
| default ($type_1$, … , $type_n$) (n ≥ 0)
| foreign fdecl
| decl

decls → { $decl_1$; … ; $decl_n$ } (n ≥ 0)
decl → gendecl
| (funlhs | pat) rhs

…

2. Semantics

$$
\begin{array}{lcl}
(\Lambda a.e)\ \varphi & \longrightarrow & [\varphi/a]e \\
(\lambda x.e)\ e' & \longrightarrow & [e'/x]e \\
\textbf{case}\ (K\ \overline{\sigma}\ \overline{\varphi}\ \overline{e})\ \textbf{of}\ \ldots\ K\ \overline{b}\ \overline{x} \to e'\ \ldots & \longrightarrow & [\overline{\varphi/b}, \overline{e/x}]e' \\
(v \blacktriangleright \gamma_1) \blacktriangleright \gamma_2 & \longrightarrow & v \blacktriangleright (\gamma_1 \circ \gamma_2)
\end{array}
$$

$$
\begin{array}{lcl}
((\Lambda a:\kappa.\ e) \blacktriangleright \gamma)\ \varphi & \longrightarrow & (\Lambda a:\kappa.\ (e \blacktriangleright \gamma@a))\ \varphi \\
 & \text{where} & \gamma : (\forall a:\kappa.\ \sigma_1) \sim (\forall b:\kappa.\ \sigma_2)
\end{array}
$$

$$
\begin{array}{lcll}
((\Lambda a:\kappa.\ e) \blacktriangleright \gamma)\ \varphi & \longrightarrow & (\Lambda a':\kappa'.\ (([(a' \blacktriangleright \gamma_1)/a]e) \blacktriangleright \gamma_2))\ \varphi & \\
 & \text{where} & \gamma : (\kappa \Rightarrow \sigma) \sim (\kappa' \Rightarrow \sigma') & \\
 & & \gamma_1 = \text{sym}\ (\text{leftc}\ \gamma) & \text{– coercion for argument} \\
 & & \gamma_2 = \text{rightc}\ \gamma & \text{– coercion for result}
\end{array}
$$

$$
\begin{array}{lcll}
((\lambda x.e) \blacktriangleright \gamma)\ e' & \longrightarrow & (\lambda y.([(y \blacktriangleright \gamma_1)/x]e \blacktriangleright \gamma_2))\ e' & \\
 & \text{where} & \gamma_1 = \text{sym}\ (\text{right}\ (\text{left}\ \gamma)) & \text{– coercion for argument} \\
 & & \gamma_2 = \text{right}\ \gamma & \text{– coercion for result}
\end{array}
$$

$$
\begin{array}{lcl}
\textbf{case}\ (K\ \overline{\sigma}\ \overline{\varphi}\ \overline{e} \blacktriangleright \gamma)\ \textbf{of}\ \overline{p \to rhs} & \longrightarrow & \textbf{case}\ (K\ \overline{\tau}\ \overline{\varphi'}\ \overline{e'})\ \textbf{of}\ \overline{p \to rhs} \\
 & \text{where} & \gamma : T\ \overline{\sigma} \sim T\ \overline{\tau} \\
 & & K : \forall \overline{a:\kappa}.\ \forall \overline{b:\iota}.\ \overline{\rho} \to T\ \overline{a}^n \\
 & & \varphi'_i = \begin{cases} \varphi_i \blacktriangleright \theta(v_1 \sim v_2) & \text{if } b_i : v_1 \sim v_2 \\ \varphi_i & \text{otherwise} \end{cases} \\
 & & e'_i = e_i \blacktriangleright \theta(\rho_i) \\
 & & \theta = [\overline{\gamma_i/a_i}, \overline{\varphi_i/b_i}] \\
 & & \gamma_i = \text{right}\ \underbrace{(\text{left} \ldots (\text{left}}_{n-i}\ \gamma))
\end{array}
$$

Defining IMP

1. Syntax

```
C := skip
   | X := A
   | C ; C
   | if B then C
          else C end
   | while B do C end
```

2. Semantics

$$\frac{}{\sigma,\text{skip} \Downarrow \sigma}\ \textsc{ESkip}$$

$$\frac{\sigma,a \Downarrow_A v}{\sigma,x:=a \Downarrow [x \mapsto v]\sigma}\ \textsc{EAssn}$$

★ **Theorem [IMP Is Deterministic]:**

For any command c, from any starting state σ, c can evaluate to at most one unique final state:

If $\sigma,\ c \Downarrow \sigma_1$ and $\sigma,\ c \Downarrow \sigma_2$, then $\sigma_1 = \sigma_2$.

Defining IMP+FLIP

1. Syntax

```
C := skip
   | X := A
   | C ; C
   | if B then C
          else C end
   | while B do C end
   | if flip C
```

2. Semantics

$$\frac{\sigma, a \Downarrow v}{\sigma, x := a \Downarrow [x \mapsto v]\sigma}\ \textsc{EAssn}$$

$$\frac{\sigma_1, c \Downarrow \sigma_2}{\sigma 1, \texttt{if flip c} \Downarrow \sigma 2}\ \textsc{EFlipT}$$

$$\frac{}{\sigma, \texttt{if flip c} \Downarrow \sigma}\ \textsc{EFlipF}$$

Concept Check

Theorem [IMP+Flip Is Not Deterministic]:

For some commands c, from any starting state σ, c can evaluate to multiple final states:

$\exists\ \sigma\ c\ \sigma_1\ \sigma_2$. If σ, c $\Downarrow$ σ_1 and σ, c $\Downarrow$ σ_2 and $\sigma_1 \neq \sigma_2$.

Can you write an IMP+Flip program that evaluates to different final states?

Can you write an IMP+Flip program that evaluates to an infinite number of final states?

Defining IMP +RAND

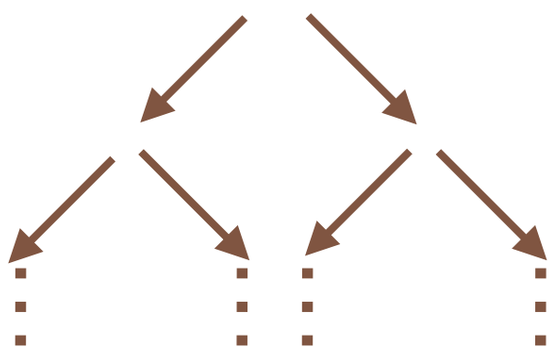
1. Syntax

```
C := skip
   | X := A
   | C ; C
   | if B then C
          else C end
   | while B do C end
   | X := Any
```

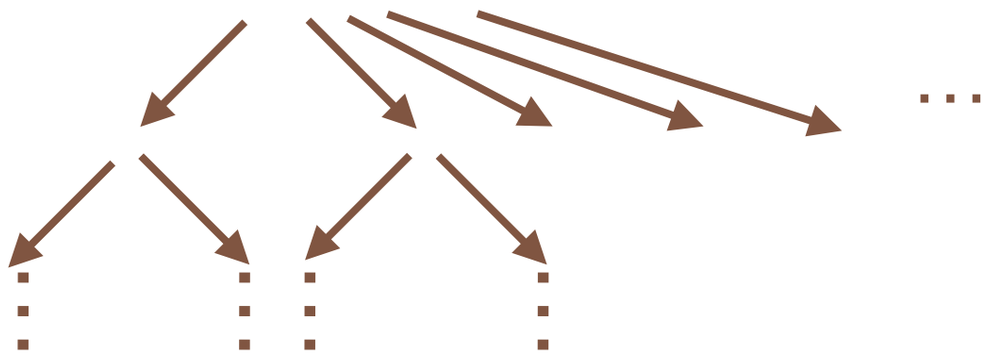
2. Semantics

$$\frac{v \in \mathbb{N}}{\sigma,\ \texttt{X := Any} \Downarrow [x \mapsto v]\sigma}\ \text{ERAND}$$

IMP+Flip: finite number of branches
infinite final states



IMP+Rand: infinite number of branches
infinite final states



Fun IMP

1. Syntax

C := skip
 | X := A
 | C ; C
 | **if** B **then** C
 else C **end**
 | **while** B **do** C **end**
 | X := F(A)

F ∈ 𝔽

FD := F(X) {C; **return** a}

```
Double(Y) {
 skip;
 return Y + Y}
```

```
Double(Y) {
 Z := Y + Y;
 return Z}
```

```
X := Double(5)
```

```
Y := 5;
X := Double(Y)
```

Fun IMP

C := skip
 | X := A
 | C ; C
 | **if** B **then** C
 else C **end**
 | **while** B **do** C **end**
 | X := F(A)

$F \in \mathbb{F}$

FD := F(X) {C; **return** a}

- How to model set of function calls?
- Update the judgement!

$$\Delta \vdash \sigma_1,\ c \Downarrow \sigma_2$$

$$\Delta : F \rightarrow FD$$

Read as 'When run in initial state σ1 and using the function definitions in Δ, c produces (i.e. evaluates to) final state σ2"

Fun IMP

$$\frac{\Delta \vdash \sigma, a \Downarrow v}{\Delta \vdash \sigma, x := a \Downarrow [x \mapsto v]\sigma} \text{EASSN}$$

$$\frac{\Delta(F) = F(y)\ \{c;\ \textbf{return}\ a_2\} \qquad \Delta \vdash \sigma_1, \bar{a} \Downarrow \bar{v} \qquad \Delta \vdash [\bar{y} \mapsto \bar{v}], c \Downarrow \sigma_2 \qquad \Delta \vdash \sigma_2, a_2 \Downarrow v_2}{\Delta \vdash \sigma_1, x := F(\bar{a}) \Downarrow [x \mapsto v_2]\sigma} \text{ECALL}$$