Midterm Review (v1.1)

March 5, 2018

1 Syntax

1.1 Inductive Definitions

Inference Rules:

$$\frac{}{\mathtt{T} \in \text{exp}} \qquad \frac{}{\mathtt{F} \in \text{exp}} \qquad \frac{e_1 \in \text{exp} \quad e_2 \in \text{exp} \quad e_3 \in \text{exp}}{(\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3) \in \text{exp}}$$

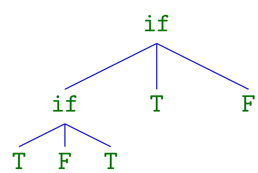
Grammar Schema:

$$e ::= \mathtt{T} \mid \mathtt{F} \mid \mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e$$

An example expression:

$$\mathtt{if}\ (\mathtt{if}\ \mathtt{T}\ \mathtt{else}\ \mathtt{F}\ \mathtt{else}\ \mathtt{T})\ \mathtt{else}\ \mathtt{T}\ \mathtt{else}\ \mathtt{F}$$

is the tree:



1.2 Metafunctions

$$\begin{aligned}
\text{size} &\in \text{exp} \to \mathbb{N} \\
\text{size}(\mathtt{T}) &\coloneqq 1 \\
\text{size}(\mathtt{F}) &\coloneqq 1 \\
\text{size}(\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3) &\coloneqq \text{size}(e_1) + \text{size}(e_2) + \text{size}(e_3) + 1
\end{aligned}$$

2 Semantics

One purpose of semantics is to distinguish the syntax of expressions with the results of their evaluation. E.g.:

$$e_1 \coloneqq \texttt{if T else F else T}$$
$$e_2 \coloneqq \texttt{if F else T else F}$$

These expressions are different:

$$e_1 \neq e_2$$

But they evaluate to the same value:

$$e_1 \longrightarrow^* \texttt{F}$$
$$e_2 \longrightarrow^* \texttt{F}$$

2.1 Small-step Operational Semantics

Three steps to defining a small-step operational semantics:

1. Define a one-step relation $e \longrightarrow e$.
2. Identify a subset of terms which are called *values*.
3. Induce a many-step relation $e \longrightarrow^* e$ as the reflexive, transitive closure of $e \longrightarrow^* e$.

$\boxed{e \longrightarrow^* e}$

$$\frac{}{\texttt{if T then } e_2 \texttt{ else } e_3 \longrightarrow e_2}\textsc{If-T} \qquad \frac{}{\texttt{if F then } e_2 \texttt{ else } e_3 \longrightarrow e_3}\textsc{If-F}$$

$$\frac{e_1 \longrightarrow e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \longrightarrow \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3}\textsc{If-Cong}$$

Values:

$$v ::= \texttt{T} \mid \texttt{F}$$

Many-step Relation:

$$e \longrightarrow^* e'$$

means:

$$e \longrightarrow e_1 \longrightarrow \cdots \longrightarrow e_{n-1} \longrightarrow e'$$

for some n.

E.g.:

$$\texttt{if } (\texttt{if T then F else T}) \texttt{ then F else T} \longrightarrow^* \texttt{T}$$

in two steps:

$$\begin{array}{c}\texttt{if (if T then F else T) then F else T} \\ \longrightarrow \\ \texttt{if F then F else T} \\ \longrightarrow \\ \texttt{T}\end{array}$$

Each step is justified via a derivation tree. The first step:

$$\dfrac{\dfrac{}{\texttt{if T then F else T} \longrightarrow \texttt{F}}\,\textsc{If-T}}{\texttt{if (if T then F else T) then F else T} \longrightarrow \texttt{if F then F else T}}\,\textsc{If-Cong}$$

and the second step:

$$\dfrac{}{\texttt{if F then F else T} \longrightarrow \texttt{T}}\,\textsc{If-F}$$

3 Arithmetic Expressions

Syntax:

$$\begin{aligned} e &::= \ldots \mid \texttt{0} \mid \texttt{succ}\ e \mid \texttt{pred}\ e \mid \texttt{iszero}\ e \\ v &::= \ldots \mid nv \\ nv &::= \texttt{0} \mid \texttt{succ}\ nv \end{aligned}$$

Semantics:

$\boxed{e \longrightarrow e}$

$$\ldots \qquad \frac{e \longrightarrow e'}{\texttt{succ}\ e \longrightarrow \texttt{succ}\ e'} \qquad \frac{}{\texttt{pred 0} \longrightarrow \texttt{0}} \qquad \frac{}{\texttt{pred (succ}\ nv\texttt{)} \longrightarrow nv} \qquad \frac{e \longrightarrow e'}{\texttt{pred}\ e \longrightarrow \texttt{pred}\ e'}$$

$$\frac{}{\texttt{iszero 0} \longrightarrow \texttt{T}} \qquad \frac{}{\texttt{iszero (succ}\ nv\texttt{)} \longrightarrow \texttt{F}} \qquad \frac{e \longrightarrow e'}{\texttt{iszero}\ e \longrightarrow \texttt{iszero}\ e'}$$

4 Safety

A term e is *stuck iff* e is not a value *and* there is no e' such that $e \longrightarrow e'$.

A term e is *unsafe iff* there exists some e' such that $e \longrightarrow e'$ and e' is stuck.

A term e is *safe iff* for every e' such that $e \longrightarrow e'$, e' is not stuck.

If a term is not safe, then it is unsafe. If a term is not unsafe, then it is safe.

E.g.:

$$\texttt{if T then pred 0 else 0}$$

is safe.

However:

$$\texttt{if T then pred F else 0}$$

is unsafe.

5 Types

Typing is a syntactic relation $e : \tau$.

If there exists a type τ such that $e : \tau$ then e is *well-typed*.

If there does not exist a type τ such that $e : \tau$ then e is *untypeable*.

Type safety is a theorem that says if e is well-typed then e is safe.

Types for the Boolean Arithmetic Language:

$\boxed{e : \tau}$

$$\frac{}{\texttt{T} : \texttt{bool}} \qquad \frac{}{\texttt{F} : \texttt{bool}} \qquad \frac{e_1 : \texttt{bool} \qquad e_2 : \tau \qquad e_2 : \tau}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

$$\frac{}{\texttt{0} : \texttt{nat}} \qquad \frac{e : \texttt{nat}}{\texttt{succ}\ e : \texttt{nat}} \qquad \frac{e : \texttt{nat}}{\texttt{pred}\ e : \texttt{nat}} \qquad \frac{e : \texttt{nat}}{\texttt{iszero}\ e : \texttt{bool}}$$

5.1 Progress + Preservation = Safety

The proof mechanism for type safety is to use two lemmas: progress and preservation.

Progress: if $e : \tau$ then e is not stuck.

Preservation: if $e : \tau$ and $e \longrightarrow e'$ then $e' : \tau$.

Safety is a direct corollary from Progress + Preservation.

5.2 Inversion Lemmas

Type judgments can be "inverted", which give rise to a number of inversion lemmas. Here are two examples of inversion lemmas:

- If $e = \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$ and $e : \tau$, then:
  1. $e_1 : \texttt{bool}$
  2. $e_2 : \tau$
  3. $e_3 : \tau$
- If $e : \texttt{bool}$, then *either* of the following are true:
  1. $e = \texttt{T}$
  2. $e = \texttt{F}$
  3. $e = \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$, $e_1 : \texttt{bool}$, $e_2 : \texttt{bool}$ and $e_3 : \texttt{bool}$
  4. $e = \texttt{iszero}\ e'$ and $e' : \texttt{nat}$

6 Lambda Calculus

Syntax:

$$e ::= x \mid \lambda x.\ e \mid e\ e$$

When we write multiple terms in sequence:

$$e_1\ e_2\ e_3\ e_4$$

the implicit parenthesis placement is left-associative:

$$((e_1\ e_2)\ e_3)\ e_4$$

and when we write multiple lambda terms in sequence:

$$\lambda x.\ \lambda y.\ \lambda z.\ e$$

the implicit parenthesis placement is right-associative:

$$\lambda x.\ (\lambda y.\ (\lambda z.\ e))$$

Values:

$$v ::= \lambda x.\ e$$

Metafunctions for free variables and substitution:

$$\begin{aligned} FV &\in \text{exp} \to \mathcal{P}(\text{var}) \\ FV(x) &\coloneqq \{x\} \\ FV(\lambda x.\ e) &\coloneqq FV(e) \setminus \{x\} \\ FV(e_1\ e_2) &\coloneqq FV(e_1) \cup FV(e_2) \end{aligned}$$

$$\begin{aligned} [x{\mapsto}e_2] &\in \text{exp} \rightharpoonup \text{exp} && \\ [x \mapsto e_2]y &\coloneqq e_2 && \text{if } x = y \\ [x \mapsto e_2]y &\coloneqq y && \text{if } x \neq y \\ [x \mapsto e_2](\lambda y.\ e) &\coloneqq \lambda y.\ [x \mapsto e_2](e) && \text{if } x \neq y \text{ and } y \notin FV(e_2) \\ [x \mapsto e_2](e_{11}\ e_{12}) &\coloneqq ([x \mapsto e_2]e_{11})\ ([x \mapsto e_2]e_{12}) && \end{aligned}$$

E.g.:

$$FV(\lambda x.\ x\ y) = \{y\}$$
$$[x \mapsto z](\lambda x.\ x\ y) = \lambda x.\ x\ y$$
$$[y \mapsto z](\lambda x.\ x\ y) = \lambda x.\ x\ z$$
$$[y \mapsto x](\lambda x.\ x\ y) \neq$$

It is always okay to convert lambda terms to alpha-equivalent terms.

E.g.:

$$\lambda x.\ \lambda y.\ x\ y \approx \lambda x.\ \lambda z.\ x\ z \approx^a \lambda y.\ \lambda z.\ y\ z \approx^a \lambda y.\ \lambda x.\ y\ x$$

This allows for the last substitution example to be given meaning:

$$[y \mapsto x](\lambda x.\ x\ y) \approx^a [y \mapsto x](\lambda z.\ z\ y) = \lambda z.\ z\ x$$

There is one rule which describes how lambda terms take a step, the β rule:

$$\frac{}{(\lambda x.\ e_1)e_2 \longrightarrow [x \mapsto e_2]e_1}\ \beta$$

The full semantics for call-by-value lambda calculus:

$\boxed{e \longrightarrow e}$

$$\frac{}{(\lambda x.\ e)\ v \longrightarrow [x \mapsto v]e}\ \beta \qquad \frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{e \longrightarrow e'}{v\ e \longrightarrow v\ e'}$$

These three rules fully describe universal computation (i.e., equivalent in computational power to Turing machines.)

Boolean encodings:

$$\begin{aligned} \text{true} &\coloneqq \lambda x.\ \lambda y.\ x \\ \text{false} &\coloneqq \lambda x.\ \lambda y.\ y \\ \text{cond} &\coloneqq \lambda b.\ \lambda x.\ \lambda y.\ b\ x\ y \\ \text{cond-alt} &\coloneqq \lambda b.\ b \end{aligned}$$

An infinite loop:

$$\Omega \coloneqq (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

Recursion can be defined using the Y-combinator:

$$Y \coloneqq \lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$$

which has the following property:

$$Y\ f \longrightarrow^* f\ (Y\ f)$$

7 Typed Lambda Calculus

$$\begin{aligned} \tau &::= \ldots \mid \tau \Rightarrow \tau \\ \Gamma &::= [] \mid \Gamma, x : \tau \end{aligned}$$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash (\lambda x.\ e) : \tau_1 \Rightarrow \tau_2} \qquad \frac{\Gamma \vdash e_1 : (\tau_1 \Rightarrow \tau_2) \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

The same type safety theorem holds: if $e : \tau$ then e is safe.

8 Typed Lambda Calculus Extensions

Empty type:

$$\begin{aligned} \tau &::= \ldots \mid \text{empty} \\ e &::= \ldots \mid \text{absurd } e \\ v &::= \ldots \qquad \textit{(no new values)} \end{aligned}$$

$\boxed{e \longrightarrow e}$

(this rule is safe to have, but not useful or necessary...)

$$\frac{e \longrightarrow e'}{\text{absurd } e \longrightarrow \text{absurd } e'}$$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{\Gamma \vdash e : \text{empty}}{\Gamma \vdash \text{absurd } e : \tau}\textsc{Empty-Elim}$$

Unit type:

$$\begin{aligned} \tau &::= \ldots \mid \text{unit} \\ e &::= \ldots \mid \bullet \\ v &::= \ldots \mid \bullet \end{aligned}$$

$\boxed{e \longrightarrow e}$

(no new rules)

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash \bullet : \text{unit}}\textsc{Unit-Intro}$$

Sum type:

$$
\begin{aligned}
\tau &::= \ldots \mid \tau + \tau \\
e &::= \ldots \mid \text{inl } e \mid \text{inr } e \mid \text{case}(e)\{x.\ e\}\{x.\ e\} \\
v &::= \ldots \mid \text{inl } v \mid \text{inr } v
\end{aligned}
$$

$\boxed{e \longrightarrow e}$

$$
\frac{e \longrightarrow e'}{\text{inl } e \longrightarrow \text{inl } e'} \qquad \frac{e \longrightarrow e'}{\text{inl } e \longrightarrow \text{inl } e'} \qquad \frac{}{\text{case}(\text{inl}(v))\{x_2.\ e_2\}\{x_3.\ e_3\} \longrightarrow [x_2 \mapsto v]e_2}
$$

$$
\frac{}{\text{case}(\text{inr}(v))\{x_2.\ e_2\}\{x_3.\ e_3\} \longrightarrow [x_3 \mapsto v]e_3} \qquad \frac{e_1 \longrightarrow e_1'}{\text{case}(e_1)\{x_2.\ e_2\}\{x_3.\ e_3\} \longrightarrow \text{case}(e_1')\{x_2.\ e_2\}\{x_3.\ e_3\}}
$$

$\boxed{\Gamma \vdash e : \tau}$

$$
\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{inl } e : \tau_1 \times \tau_2}\,\textsc{Sum-Intro-1} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{inr } e : \tau_1 \times \tau_2}\,\textsc{Sum-Intro-2}
$$

$$
\frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \qquad \Gamma, x_2 : \tau_1 \vdash e_2 : \tau \qquad \Gamma, x_3 : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \text{case}(e_1)\{x_2.\ e_2\}\{x_3.\ e_3\} : \tau}\,\textsc{Sum-Elim}
$$

Product type:

$$
\begin{aligned}
\tau &::= \ldots \mid \tau \times \tau \\
e &::= \ldots \mid \langle e_1, e_2 \rangle \mid \text{projl } e \mid \text{projr } e \\
v &::= \ldots \mid \langle v_1, v_2 \rangle
\end{aligned}
$$

$\boxed{e \longrightarrow e}$

$$
\frac{e_1 \longrightarrow e_1'}{\langle e_1, e_2 \rangle \longrightarrow \langle e_1', e_2 \rangle} \qquad \frac{e \longrightarrow e'}{\langle v, e \rangle \longrightarrow \langle v, e' \rangle} \qquad \frac{}{\text{projl } \langle v_1, v_2 \rangle \longrightarrow v_1} \qquad \frac{e \longrightarrow e'}{\text{projl } e \longrightarrow \text{projl } e'}
$$

$$
\frac{}{\text{projr } \langle v_1, v_2 \rangle \longrightarrow v_2} \qquad \frac{e \longrightarrow e'}{\text{projr } e \longrightarrow \text{projr } e'}
$$

$\boxed{\Gamma \vdash e : \tau}$

$$
\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}\,\textsc{Prod-Intro} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{projl } e : \tau_1}\,\textsc{Prod-Elim-1} \qquad \frac{\Gamma \vdash e : \tau_2 \times \tau_2}{\Gamma \vdash \text{projr } e : \tau_2}\,\textsc{Prod-Elim-2}
$$

Errata

v1.1 Fixed typo in the rule:

$$
\frac{e_1 \longrightarrow \boxed{e_2'}}{\text{case}(e_1)\{x_2.\ e_2\}\{x_3.\ e_3\} \longrightarrow \text{case}(e_1')\{x_2.\ e_2\}\{x_3.\ e_3\}}
$$

which has been corrected to be:

$$\frac{e_1 \longrightarrow e_1'}{\text{case}(e_1)\{x_2.\ e_2\}\{x_3.\ e_3\} \longrightarrow \text{case}(e_1')\{x_2.\ e_2\}\{x_3.\ e_3\}}$$