Practical Foundations for Programming Languages

Jason Hu, Zhe Zhou, Ramana Nagasamudram

June 20, 2019

Lecture 2

1 Evaluation order

In this lecture, we tackle issues related to the evaluation order of expressions. Traditionally, the terminology that is used is *call-by-name* (CBN) and *call-by-value* (CBV). However, logically speaking, both have nothing to do with the notion of a "call". We therefore recast these issues as pertaining to the semantics of variables.

The key is to understand what variables in our language range over. This becomes especially important in languages that have a notion of undefinedness. To understand this more, we shall study Plotkin's PCF in the by-name and by-value settings.

2 PCF

2.1 Syntax

$$\tau ::= 2 \mid \omega \mid \tau_1 \to \tau_2$$
$$e ::= x \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(e; e; e) \mid \mathtt{zero} \mid \mathtt{succ}(e) \mid \mathtt{ifz}(e; e; x.e) \mid \lambda[\tau](x.e) \mid \mathtt{ap}(e; e) \mid \mathtt{fix}[\tau](x.e)$$

2.2 Statics of PCF

$$\frac{}{x : \tau \vdash x : \tau} \qquad \frac{}{\mathtt{tt} : 2} \qquad \frac{}{\mathtt{ff} : 2} \qquad \frac{}{zero : \omega} \qquad \frac{e : \omega}{\mathtt{succ}(e) : \omega} \qquad \frac{x : \tau_1 \vdash e_2 : \tau_2}{\lambda[\tau_1](x.e_2) : \tau_1 \to \tau_2}$$

$$\frac{e : 2 \qquad e_1 : \tau \qquad e_2 : \tau}{\mathtt{if}(e; e_1; e_2) : \tau} \qquad \frac{e : \omega \qquad e_1 : \tau \qquad x : \omega \vdash e_2 : \tau}{\mathtt{ifz}(e; e_1; x.e_2) : \tau} \qquad \frac{e_1 : \tau_2 \to \tau \qquad e_2 : \tau_2}{ap(e_1; e_2) : \tau}$$

2.3 Dynamics

Given in the form of a transition system.

With

$$e \Downarrow v$$

there are cases when the evaluation doesn't have a value. We want to distinguish cases that are supposed to be ruled out by statics to cases that are actually well formed.

Define in terms of $e \mapsto e'$, e `val`.

As before, but ...

$$\frac{}{\texttt{zero val}} \qquad \frac{e \texttt{ val}}{succ(e) \texttt{ val}}$$

$$\frac{e \mapsto e'}{\texttt{ifz}(e; e_1; x.e_2) \mapsto \texttt{ifz}(e'; e_1; x.e_2)} \qquad \frac{}{\texttt{ifz}(\texttt{zero}; e_1; x.e_2) \mapsto e_1}$$

$$\frac{\texttt{succ}(e) \texttt{ val}}{\texttt{ifz}(\texttt{succ}(e); e_1; x.e_2) \mapsto [e/x]e_2} \qquad \frac{}{\lambda[\tau](\tau_1.x.e_2) \texttt{ val}} \qquad \frac{e_1 \mapsto e_1'}{\texttt{ap}(e_1, e_2) \mapsto \texttt{ap}(e_1', e_2)}$$

$$\frac{e_1 \texttt{ val} \qquad e_2 \mapsto e_2'}{\texttt{ap}(e_1, e_2) \mapsto \texttt{ap}(e_1, e_2')} \text{ (BY-VALUE)} \qquad \frac{e_2 \texttt{ val}}{\texttt{ap}(\lambda[\tau](\tau_1.x.e), e_2) \mapsto [e_2/x]e} \text{ (BY-NAME)}$$

By-name (or lazy semantics) works well for "negative" types (function or product types). By-value works well for "positive" types (2, ω). Not vice-versa.

Example: In the by-name setting, it makes sense to define

$$\overline{V}_I = \lambda[2](x_1.\lambda[2](x_2.if(x_1; \texttt{ff}; not(x_2))))$$

This can be a function, exactly because the values are not evaluated. By in the by-value setting, we have to evaluate both branches of if before we get the final result. That is not the meaning of $\overline{V}_I$. (Lot of emphasis is placed on this (but it might be misplaced)).

You can encode the by-name in the by-value method but not the other way. Therefore by-value is superior. Hands down victory!

By-name: there are no booleans, and there are no natural numbers – you cannot have them. Because undefininedness plays the role of a value. It is wrong to say "if $x : 2$ then x converges", because x can stand for arbitrary divergent program. Reasoning by cases is never valid in by-name semantics.

What about by-value? We have a problem with `fix` because it is defined by unrolling.

By-name:

$$fix[\tau](x.e) \mapsto [fix[\tau](x.e)/x]e$$

By-value: ?

The real issue: how do I get the semantics of variables to work?

Reformulate to express by-value variables. Formulation is called lax logic / CBPV / polarity (look up these words).

3 PCF-Modality

Key-idea: **modality** that distinguish computations from values (valuables)

values (with strong typing):

$$\ldots x : \tau \ldots \vdash v : \tau$$

computations (with lax/weak typing):

$$\ldots x : \tau \ldots \vdash m \mathrel{\dot{\sim}} \tau$$

But in every cases, variables range over values, because they have a : .

The syntax can't adapt the same fix; it needs to be changed.

$$\begin{aligned} \tau &::= 2 \mid \omega \mid \tau_1 \to \tau_2 \mid \tau \ \mathtt{comp} \\ v &::= x \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{zero} \mid \mathtt{succ}(v) \mid \lambda[\tau](x.m) \mid m \ \mathtt{comp} \\ m &::= \mathtt{if}(v; m; m) \mid \mathtt{ifz}(v; m; x.m) \mid \mathtt{ret}(v) \mid \mathtt{bnd}[\tau](v; x.m) \mid \mathtt{ap}(v, v) \end{aligned}$$

3.1 Statics of PCF-Modality

$$\frac{}{x : \tau \vdash x : \tau} \qquad \frac{}{\mathtt{tt} : 2} \qquad \frac{}{\mathtt{ff} : 2} \qquad \frac{}{zero : \omega} \qquad \frac{v : \omega}{\mathtt{succ}(v) : \omega} \qquad \frac{x : \tau_1 \vdash m_2 \mathrel{\dot{\sim}} \tau_2}{\lambda[\tau_1](x.m_2) : \tau_1 \to \tau_2}$$

$$\frac{m \mathrel{\dot{\sim}} \tau}{\mathtt{comp}(m) : \tau \ \mathtt{comp}}$$

$$\frac{v : 2 \qquad M_1 \mathrel{\dot{\sim}} \tau \qquad M_2 \mathrel{\dot{\sim}} \tau}{\mathtt{if}(v; M_1; M_2) \mathrel{\dot{\sim}} \tau} \qquad \frac{v : \omega \qquad e_1 \mathrel{\dot{\sim}} \tau \qquad x : \omega \vdash e_2 \mathrel{\dot{\sim}} \tau}{\mathtt{ifz}(v; e_1; x.e_2) \mathrel{\dot{\sim}} \tau} \qquad \frac{v_1 : \tau_2 \to \tau \qquad v_2 : \tau_2}{ap(v_1; v_2) \mathrel{\dot{\sim}} \tau}$$

$$\frac{v_1 : \tau_1 \ \mathtt{comp} \qquad x : \tau_1 \vdash m_2 \mathrel{\dot{\sim}} \tau_2}{\mathtt{bnd}(v_1; x.m_2) \mathrel{\dot{\sim}} \tau_2} \qquad \frac{v : \tau}{\mathtt{ret}(v) \mathrel{\dot{\sim}} \tau}$$

Lemma 3.1 (Substitution)**.** *1. If $x : \tau \vdash v' : \tau'$ and $v : \tau$ then $[v/x]v' : \tau'$*

2. If $x : \tau \vdash m' \mathrel{\dot{\sim}} \tau'$ and $v : \tau$ then $[v/x]m' \mathrel{\dot{\sim}} \tau'$

Exercise: Check that we have progress + preservation

3.2 Dynamics of PCF-Modality

Either a computation is done or it makes a step. Values and computations are linked by introducing a computational modality.

m *done*, $m \mapsto m'$

Example transition between computations:

$\mathtt{ap}(\lambda[\tau_1](x.e_2, v_2) \mapsto [v_2/x]m_2$

What about the dynamics of bind?

$$\frac{}{\mathtt{ret}(v)\ \mathit{done}}$$

$$\frac{}{\mathtt{ap}(\lambda[\tau](\tau.x.m), v) \mapsto [v/x]m} \qquad \frac{m_1 \mapsto m_1'}{\mathtt{bnd}(\mathtt{comp}(m_1); x.m_2) \mapsto \mathtt{bnd}(\mathtt{comp}(m_1'); x.m_2)}$$

$$\frac{}{\mathtt{bnd}(\mathtt{comp}(\mathtt{ret}(v)), x.m_2) \mapsto [v/x]m_2}$$

Idea: $\tau_1 \to \tau_2$ when applied is a computation. Reformulate it so that it is a total function that returns a computation: $\tau_1 \to (\tau_2\ \mathtt{comp})$

Ah! : $\tau_1\ \mathtt{comp} \to \tau_2\mathtt{comp}$ would be a "by name" function. $lsucc : \omega\ \mathtt{comp} \to \omega$

Notice: a closed value of type 2 is a boolean! and ω is actually a numeral!

We recover mathematical induction because $\forall x : 2$ means for all values! We can reason by cases.

All of this arises from the semantics of variables. The formulation shown above deals with this semantics of variables.

4 PCF-Modality-SelfRef

Q. How do we deal with self-references?

Change the "$m\ \mathtt{comp}$" to "$\mathtt{comp}[\tau](x.m)$". We treat all "comp" terms as a self-reference terms.

$$v ::= \cdots \mid \mathtt{comp}[\tau](x.m)$$

x in $\mathtt{comp}[\tau](x.m)$ is the self-reference and the value in question is the encapsulated value.

$$\frac{\Gamma, x : \tau\ \mathtt{comp} \vdash m \mathrel{\dot{\sim}} \tau}{\Gamma \vdash \mathtt{comp}[\tau](x.m) : \tau\ \mathtt{comp}}$$

One way to formulate (intuition: unroll when you inspect!),

$$\frac{[\texttt{comp}[\tau](x.m_1)/x]m_1 \mapsto m_1'}{\texttt{bnd}(\texttt{comp}[\tau](x.m_1); x_2.m_2) \mapsto \texttt{bnd}(\texttt{comp}[\tau](x.m_1'); x_2.m_2)}$$

Exercise: write down a RSL latch using this method.

5 Exceptions

Convenient way to express control flow.

Benton and Kennedy: allow for two forms of termination in a bind! (divergence vs explicitly declining to converge)

Generalize bind,

$$\texttt{bnd}(v; x_1.m_1; x_2.m_2)$$

where $x_1.m_1$ is the ordinary return, and $x_2.m_2$ is an exceptional return.

$$\texttt{ret}(v)\ \mathit{ord}\ \mathit{ret}$$
$$\texttt{raise}(v)\ \mathit{exe}\ \mathit{ret}$$

$$m ::= ... \mid \texttt{ret}(v) \mid \texttt{raise}(v) \mid \texttt{bnd}(v; x_1.m_1; x_2.m_2)$$

$$\frac{}{\texttt{raise}(v) : \tau_{exn}}$$

The bind we have here handles both forms of return – ordinary and exceptional.

5.1 Statics

$$\frac{v : \tau_{exn}}{\texttt{raise}(v) \mathrel{\dot{\sim}} \tau} \qquad \frac{v : \tau\texttt{comp} \qquad x_1 : \tau \vdash m_1 \mathrel{\dot{\sim}} \tau' \qquad x_2 : \tau_{exn} \vdash m_2 \mathrel{\dot{\sim}} \tau'}{\texttt{bnd}(\texttt{comp}(\texttt{raise}(v)); x_1.m_1; x_2.m_2) \mathrel{\dot{\sim}} \tau'}$$

5.2 Dynamics

$$\frac{}{\texttt{bnd}(\texttt{comp}(\texttt{raise}(v)); x_1.m_1; x_2.m_2) \mapsto [v/x_2]m_2}$$

The idea is that exceptional values are shared secrets between the raiser and handler.

(Will talk about τ_{exn} in a later lecture).

6 Parallelism

Falls out naturally. The idea is to change the modality.

Parallelism is a matter of efficiency (not correctness). We want to be "work-efficient" – it should not be the case that you are doing more work just to be parallel. With by-value, we know exactly which pieces we can parallelize, so we are "work-efficient".

Parallelism is incompatible with by-value.

Lax type-system *seems* to impose sequentiality.

The essence of parallelism is sequentiality. If two operations depend on each other, the there is not parallelism.

Add to the type system $\tau_1 \otimes \cdots \otimes \tau_n$ which represents n-ary tuples of values.

$$\tau ::= \cdots \mid \tau_1 \& \tau_2 \mid \tau_1 \otimes \tau_2$$
$$v ::= \cdots \mid m_1 \& m_2 \mid v_1 \otimes v_2$$

Note that $m_1 \& m_2$(means parallel operator) are not evaluated.

$$m ::= \cdots \mid \texttt{parbnd}[\tau](v; x.m) \mid \texttt{split}(v; x_1, x_2.m)$$

Statics.

$$\frac{v : \tau_1 \& \tau_2 \qquad x_1 : \tau_1, x_2 : \tau_2 \vdash m \mathbin{\dot{\sim}} \tau}{\texttt{parbnd}[\tau](v; x_1.x_2.m) \mathbin{\dot{\sim}} \tau} \qquad \frac{v : \tau_1 \otimes \tau_2 \qquad x_1 : \tau_1, x_2 : \tau_2 \vdash m \mathbin{\dot{\sim}} \tau}{\texttt{split}(v; x_1.x_2.m) \mathbin{\dot{\sim}} \tau}$$

Dynamics.

$$\frac{m_1 \mapsto m_1' \qquad m_2 \mapsto m_2'}{\texttt{parbnd}[\tau](m_1 \& m_2; x_1.x_2.m) \mapsto \texttt{parbnd}[\tau](m_1 \& m_2; x_1.x_2.m)}$$

$$\frac{}{\texttt{parbnd}[\tau](\texttt{ret}(v_1) \& \texttt{ret}(v_2); x_1.x_2.m) \mapsto [v_1, v_2/x_1, x_2]m}$$

$$\frac{m_2 \mapsto m_2'}{\texttt{parbnd}[\tau](\texttt{ret}(v_1) \& m_2; x_1, x_2.m) \mapsto \texttt{parbnd}[\tau](\texttt{ret}(v_1) \& m_2'; x_1, x_2.m)} \qquad (+ \text{ the other way})$$

Last rule goes the other way as well. Essentially we have to wait for the other computation to finish as well.

Exercise Check that type safety continues to hold.

Take a look at supplementary notes.

Things left to do: assign a cost semantics to this language.

Take a look at the "Brent type theorem"