# 15–399: Direct and indirect operational semantics of classical logic

April 13, 2005

## 1 Evaluation of classical proofs

The goal of this problem is to formulate a system for evaluating classical proofs. We need two concepts: *evaluation contexts* and **letcc/throw**.

First let us consider a language of proof terms obtained from constructive logic.

| | | | |
|---|---|---|---|
| type/proposition | $A$ | $::=$ | $A \supset A$ |
| proof term | $M$ | $::=$ | $x \mid \lambda x\!:\!A.\, M \mid M\ M$ |
| value | $V$ | $::=$ | $\lambda x\!:\!A.\, M$ |

There is a reduction rule for the implication connective:

$$(\lambda x\!:\!A.\, N)\ M \quad \to \quad [M/x]N$$

You can apply a reduction rule to any part of a proof term (and the type of the proof term remains the same), but we would like to specify some *reduction strategy* so that we can apply reduction rules in a systematic way. There are several ways to specify a reduction strategy, as we learned last week. For this problem, we need one based upon evaluation contexts.

### 1.1 Evaluation context

An evaluation context $\kappa$ is a term with a hole in it, where the hole specifies where we can apply the reduction rule given above. We write $\kappa\{M\}$ for the term resulting from filling the hole in $\kappa$ with $M$. If there is a term $\kappa\{M\}$, it means that:

- Currently we are evaluating a term $\kappa\{M\}$ (which may be thought of as an ML program), but we want to focus on the term $M$ only; we don't want to look at the surround context $\kappa$ (hence evaluation *context*).
- We should be able to apply a certain reduction rule to $M$.

#### 1.1.1 Call-by-value

Let us complete a call-by-value reduction strategy using evaluation contexts. First we redefine the reduction rule for the implication connective so that it is applied only when the argument is a value:

$$(\lambda x\!:\!A.\, N)\ V \quad \to \quad [V/x]N \qquad (\to \text{redefined})$$

This reduction rules gives rise to the following evaluation rule, for which we use a double arrow $\Rightarrow$:

$$\kappa\{(\lambda x\!:\!A.\, N)\ M\} \quad \Rightarrow \quad \kappa\{[M/x]N\}$$

Now all we have to do is to give a definition of evaluation contexts. Here is a definition of evaluation contexts for the *call-by-value* reduction strategy.

$$\text{evaluation context} \quad \kappa \quad ::= \quad [] \mid \kappa\ M \mid V\ \kappa$$

Example:

- $(\lambda x\!:\!A.\,N)\ V = []\{(\lambda x\!:\!A.\,N)\ V\} \quad \Rightarrow \quad []\{[V/x]N\} = [V/x]N$
  Intuitively we can apply the reduction rule $\rightarrow$ to the whole term because it is already in a suitable form.

- $((\lambda x\!:\!A.\,N)\ V)\ M = ([]\ M)\{(\lambda x\!:\!A.\,N)\ V\} \quad \Rightarrow \quad ([]\ M)\{[V/x]N\} = ([V/x]N)\ M$
  Intuitively the first subterm $(\lambda x\!:\!A.\,N$ is not yet a lambda abstraction, so we reduce it first.

**Question 1.** Show a reduction step for the following term: $(\lambda x\!:\!A.\,N)\ ((\lambda x'\!:\!A'.\,N')\ V)$

### 1.1.2 Evaluation context for pairs

Consider product types and their proof terms:

$$\begin{array}{llll} \text{type/proposition} & A & ::= & \cdots \mid A \wedge A \\ \text{proof term} & M & ::= & \cdots \mid (M, M) \mid \textbf{fst}\ M \mid \textbf{snd}\ M \\ \text{value} & V & ::= & \cdots \mid (V, V) \end{array}$$

Under the call-by-value reduction strategy, we evaluate $(M, N)$ by first evaluating $M$ and then $N$.

**Question 2.** Give two evaluation rules (one for **fst** and the other for **snd**) and augment the definition of evaluation contexts:

$$\text{evaluation context} \quad \kappa \quad ::= \quad \cdots \mid$$

$$\kappa\{ \qquad\qquad\qquad \} \;\Rightarrow\; \kappa\{ \qquad\qquad\qquad \}$$

$$\kappa\{ \qquad\qquad\qquad \} \;\Rightarrow\; \kappa\{ \qquad\qquad\qquad \}$$

## 1.2 letcc/throw

If we augment the previous language of proof terms with **letcc** $u$ **in** $M$ and **throw** $M$ **to** $u$, we obtain a language of proof terms for classical logic. Here is an informal description of how these two constructs work.

Consider $\kappa\{\textbf{letcc}\ u\ \textbf{in}\ M\}$. We want to focus on **letcc** $u$ **in** $M$ and completely ignore the evaluation context $\kappa$, like we did before. However we may need $\kappa$ at some point later and have to store it somewhere. So we store it in variable $u$ and then proceed to evaluate $M$. The following evaluation rule show how it works:

$$\kappa\{\textbf{letcc}\ u\ \textbf{in}\ M\} \quad \Rightarrow \quad \kappa\{[\kappa/u]M\}$$

Consider $\kappa\{\textbf{throw}\ M\ \textbf{to}\ u\}$. We first need to finish evaluating $M$ so that we can have a value. Suppose that it evaluates to $\kappa\{\textbf{throw}\ V\ \textbf{to}\ u\}$ (*i.e*, we have evaluated $M$ to $V$). At this point, however, $u$ should have already been replaced by some other evaluation context $\kappa'$ (by the above evaluation rule). If you think of a proof term as a program, your source program may contain **throw** $M$ **to** $u$, but eventually $u$ should be replaced by an evaluation context. Then we forget about the current evaluation context $\kappa$ and *jumps to/goto* $\kappa'$ with $V$ as the value to fill the hole in $\kappa'$ with.

$$\kappa\{\textbf{throw}\ V\ \textbf{to}\ \kappa'\} \quad\Rightarrow\quad \kappa'\{V\}$$

The evaluation context is now defined as follows. It's a bit ambiguous because of the second $\kappa'$ in **throw** $\kappa$ **to** $\kappa'$, but the intuition has been explained above:

$$\text{evaluation context} \quad \kappa \quad ::= \quad [] \mid \kappa\ M \mid V\ \kappa \mid \textbf{throw}\ \kappa\ \textbf{to}\ \kappa'$$

**Question 3.** Show as many reduction steps as you can for the following term:

$$(\lambda x\!:\!A.\,N)\ (\textbf{letcc}\ u\ \textbf{in}\ ((\lambda x'\!:\!A'.\,\textbf{throw}\ (\lambda y\!:\!B.\,M)\ V'\ \textbf{to}\ u)\ V))$$

with **letcc** $u$ **in** $M$ and **throw** $M$ **to** $u$, the language corresponds to classical logic because $u$ stores a proof term for $A$ *false*.

## 2 CPS transformation

The CPS transformation translates the language with **letcc** $u$ **in** $M$ and **throw** $M$ **to** $u$ (*i.e.*, corresponding to classical logic) into a smaller language without **letcc** $u$ **in** $M$ and **throw** $M$ **to** $u$. (*i.e.*, corresponding to constructive logic). Intuitively this should be possible because constructive logic is more expressive than classical logic, so it should be able to simulate every proof in classical logic.

Here is the source language. For the sake of simplicity, we do not distinguish between variable $x$ for lambda abstractions and $u$ for evaluation contexts. We do not discuss its type system because it is quite tricky; in fact, there is a paper about the type system for this language (written by Professor Harper).

$$\text{source language} \quad M \quad ::= \quad x \mid \lambda x\!:\!A.\, M \mid M\ M \mid \textbf{letcc}\ x\ \textbf{in}\ M \mid \textbf{throw}\ M\ \textbf{to}\ x$$

Here is the target language (without **letcc** $u$ **in** $M$ and **throw** $M$ **to** $u$):

$$\text{source language} \quad M \quad ::= \quad x \mid \lambda x\!:\!A.\, M \mid M\ M$$

**Question 4.** We write $M^*$ (in the target language) for the result of applying the CPS transformation to $M$ (in the source language). If $M$ has type $A$, then $M^*$ should have type $\neg\neg A = (A \supset \bot) \supset \bot$. Complete the CPS transformation:

$$
\begin{aligned}
x^* &= \\
(\lambda x\!:\!A.\, M)^* &= \\
(M\ N)^* &= \\
(\textbf{letcc}\ x\ \textbf{in}\ M)^* &= \\
(\textbf{throw}\ M\ \textbf{to}\ x)^* &=
\end{aligned}
$$

Here is one way to look at the CPS transformation. Consider a term $M$ of type $A$ in the source language. Our stingy CPS transformer says:

- Okay I have a term $M$ of type $A$, and I can readily produce a value $V$ of the same type by evaluating it.
- But I don't want to give $V$ to you for some reason!
- So, if you ever wanna use $V$, you should give me a function taking the value and returning some "answer" that you want. The answer could be whatever you like.
- Then I myself will apply the function to the value (because I still don't want to reveal to you what the value $V$ is), and tell you only what the answer is.

So our CPS transformer transforms a term $M$ of type $A$ into another term $M^*$ of type

$$(A \supset \mathit{Answer}) \supset \mathit{Answer}.$$