

Type Systems for Programming Languages (15-814)

Lecture Notes, Fall 2006, Week 10

John Bethencourt
bethenco@cs.cmu.edu

November 14 and 16, 2006

1 Control Effects and Classical Logic

1.1 Motivating Discussion

Last week we finished our discussion of continuations and using them to express exceptions. Now we will examine a relationship between control effects and classical logic that makes explicit the “computational content” of proofs in classical logic.

First, we consider the following theorem to motivate a particular distinction.

Theorem. $\exists a, b \in \mathbb{R} \setminus \mathbb{Q} \text{ s.t. } a^b \in \mathbb{Q}$.

Proof. Consider $a = b = \sqrt{2}$. Either $a^b \in \mathbb{Q}$ or not. If $a^b \in \mathbb{Q}$, we are done (since $\sqrt{2} \notin \mathbb{Q}$). Otherwise, let $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. Then $x, y \notin \mathbb{Q}$ and

$$x^y = \left(\sqrt{2}^{\sqrt{2}} \right)^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\sqrt{2}} = \sqrt{2}^2 = 2 \in \mathbb{Q}$$

□

The above is the typical proof of this theorem. However, contrary to what you might expect when first reading the theorem, the proof does not give you an a and b . Now consider this alternative statement of the theorem (which is equivalent in classical logic) and the proof following it.

Theorem. $\neg(\forall a, b \in \mathbb{R} \setminus \mathbb{Q}, a^b \in \mathbb{R} \setminus \mathbb{Q})$.

Proof. Assume for contradiction that $\forall a, b \in \mathbb{R} \setminus \mathbb{Q}, a^b \in \mathbb{R} \setminus \mathbb{Q}$. Since $\sqrt{2} \in \mathbb{R} \setminus \mathbb{Q}$, by the assumption we have that $\sqrt{2}^{\sqrt{2}} \in \mathbb{R} \setminus \mathbb{Q}$. Applying the assumption again, we obtain

$$\begin{aligned} \left(\sqrt{2}^{\sqrt{2}} \right)^{\sqrt{2}} &\in \mathbb{R} \setminus \mathbb{Q} \\ \implies 2 &\in \mathbb{R} \setminus \mathbb{Q} \\ \text{but } 2 &\in \mathbb{Q}. \end{aligned}$$

This is a contradiction, so we obtain the original theorem as the negation of our assumption. □

To some, the second form of the theorem may seem more “honest” in the sense that it states more directly what the proof is going to give you. Namely, rather than exhibiting an irrational a and b such that a^b is rational, the proof shows that a^b cannot be irrational for every irrational a and b . Classical logic fails to distinguish such non-constructive proofs of an existential from proofs which actually produce an example.

1.2 Our Logic

We now continue toward the main topic, which is a formal relationship between control effects and classical logic. One goal of pointing out this relationship is to emphasize the “social process” view of a proof. We wish to make more explicit the computational information that the proof communicates to its verifier. To begin, we define our version of propositional logic.

1.2.1 Judgements

The following three judgements form the basis for our logic:

$A \text{ true}$ “affirmation” (or “proof”)
 $A \text{ false}$ “refutation”
 $\#$ “contradiction”

Here, A can be considered a proposition. All the proofs done in this logic we are constructing will be by contradiction, i.e., the first step will always be to assume the negation of the theorem. Hypothetical judgements will be denoted as $\Gamma; \Delta \vdash J$, where J is one of the above three judgements. Γ is the set of all the current **true** judgements we are assuming, and Δ is the set of **false** judgements we are assuming.

1.2.2 Rules

Now we give a set of rules for reasoning about expressions of propositional logic using these judgements.

$$\frac{\Gamma; \Delta \vdash A \text{ true} \quad \Gamma; \Delta \vdash A \text{ false}}{\Gamma; \Delta \vdash \#} \quad (\#-I) \qquad \frac{}{\Gamma, A \text{ true}; \Delta \vdash A \text{ true}} \quad \frac{}{\Gamma; \Delta, A \text{ false} \vdash A \text{ false}} \quad (\text{REFLEXIVITY})$$

(Γ 's and Δ 's suppressed hereafter.)

$$\frac{A \text{ true} \quad B \text{ true}}{A \wedge B \text{ true}} \quad (\wedge\text{-true}) \qquad \frac{A \text{ false}}{A \wedge B \text{ false}} \quad (\wedge\text{-false-L}) \qquad \frac{B \text{ false}}{A \wedge B \text{ false}} \quad (\wedge\text{-false-R})$$

$$\frac{A \text{ true}}{A \vee B \text{ true}} \quad (\vee\text{-true-L}) \qquad \frac{B \text{ true}}{A \vee B \text{ true}} \quad (\vee\text{-true-R}) \qquad \frac{A \text{ false} \quad B \text{ false}}{A \vee B \text{ false}} \quad (\vee\text{-false})$$

$$\frac{A \text{ true} \vdash B \text{ true}}{A \supset B \text{ true}} \quad (\supset\text{-true}) \qquad \frac{A \text{ true} \quad B \text{ false}}{A \supset B \text{ false}} \quad (\supset\text{-false})$$

$$\frac{A \text{ false}}{\neg A \text{ true}} \quad (\neg\text{-true}) \qquad \frac{A \text{ true}}{\neg A \text{ false}} \quad (\neg\text{-false})$$

$$\frac{}{\top \text{ true}} \quad \frac{}{\perp \text{ false}} \quad (\text{CONSTANTS})$$

$$\frac{A \text{ false} \vdash \#}{A \text{ true}} \quad (\#-E\text{-true}) \qquad \frac{A \text{ true} \vdash \#}{A \text{ false}} \quad (\#-E\text{-false})$$

1.2.3 Example Proof

As an example of proofs in this logic, here is a proof of a derivable rule for and elimination.

Theorem. *The rule $\frac{A \wedge B \text{ true}}{A \text{ true}}$ (\wedge -E-L) is derivable, that is, $A \wedge B \text{ true} \vdash A \text{ true}$.*

Proof. As mentioned before, we always proceed by contradiction:

Assume $A \wedge B$ true (premise).

Assume A false (for contradiction).

By \wedge -false-L, $A \wedge B$ false.

By $\#$ -I, $\#$.

By $\#$ -E-true, A true. □

1.3 Classical Logic in Analytic Form

We now expand our logic so that affirmations and refutations of propositions “carry along” the steps used to construct them.

1.3.1 Judgements

The new version uses the following three judgements:

$M : A$ M is an affirmation or proof of A
 $K \div A$ K is a refutation of A
 $K \# M$ “proof meets refutation”

In our hypothetical judgements, Γ will consist of a set of affirmations $x_1 : A_1, x_2 : A_2, \dots, x_n : A_n$, and Δ will consist of a set of refutations $u_1 \div B_1, u_2 \div B_2, \dots, u_n \div B_n$.

1.3.2 Rules

Below are the new rules (again omitting Γ 's and Δ 's except for the reflexivity rules):

$$\frac{}{\Gamma, x : A; \Delta \vdash x : A} \quad \frac{}{\Gamma; \Delta, u \div A \vdash u \div A} \quad (\text{REFLEXIVITY})$$

$$\frac{M : A \quad N : B}{\langle M, N \rangle : A \wedge B} \quad \frac{K \div A}{\text{fst}; K \div A \wedge B} \quad \frac{K \div B}{\text{snd}; K \div A \wedge B} \quad (\wedge)$$

$$\frac{M : A}{\text{inl}(M) : A \vee B} \quad \frac{M : B}{\text{inr}(M) : A \vee B} \quad \frac{K \div A \quad L \div B}{\{K, L\} \div A \vee B} \quad (\vee)$$

$$\frac{x : A \vdash M : B}{\lambda x : A. M : A \supset B} \quad \frac{M : A \quad K \div B}{[M, K] \div A \supset B} \quad (\supset)$$

$$\frac{}{\langle \rangle : \top} \quad \frac{}{\text{abort} \div \perp} \quad (\top \text{ AND } \perp)$$

$$\frac{M : A \quad K \div A}{K \# M} \quad \frac{x : A \vdash S}{\mu x : A. S \div A} \quad \frac{u \div A \vdash S}{\bar{\mu} u \div A. S : A}$$

The last two rules correspond to $\text{ccp}(x : A.S)$ and $\text{crr}(u \div A.S)$ respectively, that is, call with current proof and call with current refutation.

The above rules give the statics of our proofs / programs. Now we define the dynamics.

$$\begin{aligned} \text{fst}; K \# \langle M, N \rangle &\mapsto K \# M \\ \text{snd}; K \# \langle M, N \rangle &\mapsto K \# N \end{aligned}$$

$$\begin{aligned} \{K, L\} \# \text{inl}(M) &\mapsto K \# M \\ \{K, L\} \# \text{inr}(M) &\mapsto L \# M \end{aligned}$$

$$[M, K] \# \lambda x : A. N \mapsto K \# [M/x]N$$

$$\text{not}(M) \# \text{not}(K) \mapsto K \# M$$

$$\begin{aligned} K \# \text{ccr}(u \div A. S) &\mapsto [K/u]S \quad (\text{“letcc”}) \\ (\text{ccp}(x : A. S)) \# M : A &\mapsto [M/x]S \end{aligned}$$

1.3.3 Discussion

Note that the meaning of “ \vdash ” or a proof is weaker in this setting (which corresponds to classical logic) than in constructive logic.

On another note, the elimination forms are definable. For example, we may define

$$\begin{aligned} \text{fst}(M) &:= \text{ccr}(u \div A. \text{fst}; u \# M) \\ \text{snd}(M) &:= \text{ccr}(u \div B. \text{snd}; u \# M) \\ \text{case } M \{ \text{inl}(a : A) \Rightarrow M_1 \mid \text{inr}(y : B) \Rightarrow M_2 \} : C &:= \text{ccr}(u \div C. \{ \text{ccp}(x : A. u \# M_1), \text{ccp}(y : B. u \# M_2) \} \# M) \end{aligned}$$

As an example of proofs / programs in this system, consider the law of the excluded middle:

$$\frac{\begin{array}{l} \Gamma, u \div A; \Delta \vdash M : B \quad \text{“if } A \text{ false then } B\text{”} \\ \Gamma; \Delta, x : A \vdash N : B \quad \text{“if } A \text{ true then } B\text{”} \end{array}}{\Gamma; \Delta, \vdash ? : B \quad \text{“therefore } B\text{”}}$$

What should go in “?”? Here are two type correct possibilities, which are duals of one another.

$$\begin{aligned} \text{ccr}(v \div B. v \# ([\text{ccp}(x : A. v \# N)/u]M)) \\ \text{ccr}(v \div B. v \# [\text{ccr}(u \div A. v \# M)/x]N) \end{aligned}$$

How would these operate? Let’s consider the second one.

$$\begin{aligned} K \# \text{ccr}(v \div B. v \# [\text{ccr}(u \div A. v \# M)/x]N) \\ \mapsto K \# [\text{ccr}(u \div A. K \# M)/x]N \end{aligned}$$

In this step, the program may be thought of as “bluffing”. Now things can turn out one of two ways. Either $\mapsto^* K \# P$ or

$$\begin{aligned} \mapsto^* K' \# \text{ccr}(u \div A. K \# M) \\ \mapsto^* K \# [K'/u]M \end{aligned}$$

In the first case, N doesn’t even use x , and the program was never called on its “bluff”. In the second case, N does look at x , so the program backs up the proof and is then able to “cheat”. This concludes our development of the relationship between control effects and classical logic, and the topic of control effects in general.

2 Storage Effects

The next topic is storage effects. There are two ways to do storage effects, the ML-style references and the Haskell-style monads / modal logic. Right now we will consider the references of ML. As with control effects, by adding references, we are weakening the meaning of the typing judgement “:”. We’ll call the new system we develop “PCF + refs” and use the following concrete syntax:

$$\begin{aligned}\tau &::= \dots \mid \tau \text{ ref} \\ e &::= \dots \mid \text{new}(e) \mid !(e) \mid e_1 := e_2 \mid \ell\end{aligned}$$

The ℓ form is a special form that arises only in the dynamics. The $!(e)$ and $e_1 := e_2$ forms have alternative notation $\text{get}(e)$ and $\text{set}(e_1, e_2)$.

2.1 Statics

Below is a first attempt at defining the statics of the language.

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{new}(e) : \tau \text{ ref}} \quad \frac{\Gamma \vdash e : \tau \text{ ref}}{\Gamma \vdash \text{get}(e) : \tau} \quad \frac{\Gamma \vdash e_1 : \tau \text{ ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{set}(e_1, e_2) : \tau}$$

2.2 Dynamics

To define the dynamics, we need some more infrastructure to capture the state of our abstract machine during evaluation. By $e@μ$, we denote the situation of the abstract machine having state $μ$ (i.e., $μ$ is the machine’s memory) at this point in e ’s evaluation. Here $μ$ will be considered to be a function $L \rightarrow V$, where L is a set of “loc’s” (i.e., memory addresses) and V is the set of closed values. Evaluation of the original expression e begins with $e@{\}$ and ends with $e’@μ'$, where e' is a value. The expressions and machine state are stepped according to the following rules.

$$\begin{array}{c} \frac{e@μ \mapsto e’@μ'}{\text{new}(e)@μ \mapsto \text{new}(e’)@μ'} \quad \frac{e \text{ value} \quad \ell \# u \quad (\text{here “\#” means apart})}{\text{new}(e)@μ \mapsto \ell@μ[\ell := e]} \quad \frac{e@μ \mapsto e’@μ'}{\text{get}(e)@μ \mapsto \text{get}(e’)@μ'} \\ \frac{μ(\ell) = e}{\text{get}(\ell)@μ \mapsto e@μ} \quad \frac{e_1@μ \mapsto e'_1@μ'}{\text{set}(e_1, e_2)@μ \mapsto \text{set}(e'_1, e_2)@μ'} \quad \frac{e_1 \text{ value} \quad e_2@μ \mapsto e'_2@μ'}{\text{set}(e_1, e_2)@μ \mapsto \text{set}(e_1, e'_2)@μ'} \\ \frac{e \text{ value}}{\text{set}(\ell, e)@μ \mapsto e@μ[\ell := e]} \end{array}$$

Add this point, we could go on to try to prove progress and preservation.

2.3 Example

As an example, consider using references to implement recursion via “backpatching”. Suppose we are trying to implement the following factorial function:

```
fun f(n : nat) : nat is
  ifz(n, 1, x . (s(x) * f(x)))
```

We could then do something like

$$\begin{aligned}F &:= \text{new}(\lambda n : \text{nat}.n) \quad (\text{just any } \text{nat} \rightarrow \text{nat} \text{ as a placeholder}) \\ F &:= \lambda n : \text{nat} . \text{ifz}(n, 1, x.(s(x) \cdot !(F)(x))) \\ \text{fact} &:= !(F)\end{aligned}$$

This creates a cycle of pointers. Backpatching is a particularly bad way of implementing recursion.