Curry-Howard Correspondence for Classical Logic

Stéphane Graham-Lengrand
CNRS, Laboratoire d’Informatique de l’X

Stephane.Lengrand@Polytechnique.edu
Practicalities

E-mail address: Stephane.Lengrand@Polytechnique.edu

Website: http://www.lix.polytechnique.fr/~lengrand/

Slides …

Schedule: Tuesdays, from 16:15 to 19:15
7th December, 14th December, 3rd January, 10th January (guest lecture by Beniamino Accattoli)

Room: 2035, Sophie Germain building
Lecture I
Classical logic as a typing system
Contents

I. Introduction
II. What works and what does not
III. A bit of history
I. Introduction
Curry-Howard correspondence for Classical Logic

These lectures are part of the course Logique linéaire et paradigmes logiques du calcul (mostly)

Although, polarity and focusing -from linear logic- have played a major part in the understanding of C-H correspondence for Classical Logic. (see e.g. Olivier Laurent’s PhD work Laurent [2003])
Curry-Howard correspondence

One of two sides of the coin

"computational interpretation of a logic"

output of a computation = cut-free proof

Side 1 computation as proof search
(starting from a formula to prove)

Logic programming (see e.g. Dale Miller’s course)

Side 2 computation as composition of proofs / cut-elimination
(starting from a proof with cuts)

Curry-Howard (see e.g. this course)
II. What works and what does not
Where it all works smoothly

Intuitionistic/minimal logic

<table>
<thead>
<tr>
<th>Logic</th>
<th>Programming language</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositions</td>
<td>Types</td>
<td>Objects</td>
</tr>
<tr>
<td>Proofs</td>
<td>Typed programs</td>
<td>λ-terms</td>
</tr>
<tr>
<td>Cut/Composition</td>
<td>Program composition</td>
<td>β-redex</td>
</tr>
<tr>
<td>Where the stuff happens</td>
<td>Execution</td>
<td>β-reduction</td>
</tr>
<tr>
<td>(Cut-elimination)</td>
<td></td>
<td>Equality of morphisms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(commuting diagrams)</td>
</tr>
</tbody>
</table>
For minimal logic:

Curry Combinators (S,K,I) ↔ Hilbert-style system

Howard Howard [1980] Typed λ-terms ↔ Natural Deduction

\[ \begin{align*}
I & : A \rightarrow A \\
K & : A \rightarrow B \rightarrow A \quad \text{(provides erasure)} \\
S & : (A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C) \quad \text{(provides duplication)}
\end{align*} \]

\[ \begin{align*}
I M & \rightarrow M \\
K M N & \rightarrow M \\
S M N P & \rightarrow M P (N P)
\end{align*} \]
Original correspondence

For minimal logic:

Curry Combinators (S,K,I) $\leftrightarrow$ Hilbert-style system

Howard Howard [1980] Typed $\lambda$-terms $\leftrightarrow$ Natural Deduction

$$\Gamma, x : A \vdash x : A$$

$$\Gamma, x : A \vdash M : B \quad \Gamma \vdash M : A \to B \quad \Gamma \vdash N : A$$

$$\Gamma \vdash \lambda x . M : A \to B$$

$$\Gamma \vdash M \ N : B$$

$$(\lambda x . M) \ N \rightarrow_\beta \{^{\ N/x}_x\} M$$

$$\lambda x . M \ x \rightarrow_\eta M \quad \text{if } x \notin \text{FV}(M)$$

$=_{\beta \eta}$ sound and complete for Cartesian Closed Categories (CCC)
Generalising the approach

Decorate proofs with syntactic terms: \( \Gamma \vdash A \) becomes \( \Gamma \vdash M : A \)

**Proof transformations**

Reductions (execution of) \( M : M \to_S N \) given by rewrite system \( S \).

**Desired properties of the reduction system**

- **Progress**, i.e. any term containing undesirable structures can be reduced.
- **Subject reduction** property, i.e. preservation of typing:
  
  If \( \Gamma \vdash M : A \) and \( M \to_S N \) then \( \Gamma \vdash N : A \)

1. **Confluence**, programs are deterministic
2. **Normalisation** (strong), i.e. execution of programs terminates.
Example

From minimal logic to *intuitionistic logic* = *minimal logic* + “*Ex falso Quodlibet*”
add rule:

\[ \Gamma \vdash M : \bot \]
\[ \Gamma \vdash \text{abort}(M) : A \]

Computational behaviour:

\[ \text{abort}(M) \, N \rightarrow \text{abort}(M) \]

In category theory: add to a CCC an *initial* object \( \bot \)
(i.e. for every object \( A \) there is a unique morphism \( \bot \rightarrow A \))

Now, remember that \( \neg A \) is \( A \rightarrow \bot \)
How to get classical logic

Either add axiom schemes

\[(\neg \neg A) \Rightarrow A\] (Elimination of double negation)

\[((A \Rightarrow B) \Rightarrow A) \Rightarrow A\] (Peirce’s law)

\[A \lor \neg A\] (Law of excluded middle)

In presence of “Ex falso Quodlibet” \((\bot \Rightarrow A)\):

All equivalent

Without “Ex falso Quodlibet”:

only EDN⇒PL⇒LEM

\[\Gamma, \neg A \vdash \bot \quad \text{for EDN}\]

\[\Gamma \vdash A\]

or by the structure of formalism cf. classical sequent calculus and right-contraction
The bad news

Take a CCC with initial object \( \bot \)

Require that for all object \( A \), \( A \) is naturally isomorphic to \((A \Rightarrow \bot) \Rightarrow \bot\)

The category collapses to a boolean algebra:

at most 1 morphism between any 2 objects

= cannot distinguish 2 proofs of the same theorem

Quite useless for a theory of proofs, or for the proofs-as-program paradigm

Has classical logic a computational content?

Girard, Lafont, Taylor *Proofs and types*: No Girard et al. [1989]
III. A bit of history
The notion of continuation

Program execution flow:

\[ \downarrow \text{code } P \text{ that has been executed, producing data } v \]
\[ v \text{ its output} \]
\[ \downarrow \text{code } E \text{ that remains to be executed, consuming data } v \]

Continuation

\[ = \text{programming environment/context within which some code is executed} \]
The notion of continuation

...is also useful for compiling recursive calls

```plaintext
myfunction(a1, ..., an) {
    some code;
    x = myfunction(a1', ..., an');
    some code possibly using x;
}
```

When executing recursive call, whole environment must be saved to resume computation (code that remains to be executed + state of memory).

Not needed if some code possibly using x is empty (tail recursion).

Trick = pass it to the recursive call as a “continuation” c':

```plaintext
myfunction(a1, ..., an, c) {
    some code;
    return myfunction(a1', ..., an', c');
}
```
The notion of continuation in $\lambda$-calculus

Program execution flow:

\[ \downarrow \quad \text{code } P \text{ that has been executed, producing data } v \]

\[ v \quad \text{its output} \]

\[ \downarrow \quad \text{code } E \text{ that remains to be executed, consuming data } v \]

can be seen in

- $P$ is a $\lambda$-term that is reduced
- $E$ is the context, in the syntactic sense (a term with a hole $E[\ ]$)

$E[\ ]$ can also be seen as a function $\lambda x. E[x]$
The notion of control

In pure $\lambda$-calculus, $P$ has no knowledge of $E[~]$ while being evaluated.

Control =

letting a program know and manipulate its environment/continuation
getting “unpure features”, modelling $\texttt{goto}$ instructions

on continuations and $\texttt{call-with-current-continuation (call-cc)}$: $\texttt{cc}$
Added to programming langage Scheme

- Felleisen's PhD work Felleisen [1987] on Syntactic Theory of Control: the $C \text{ operator}$
The general idea:

\[ E[\text{abort}(M)] \rightarrow M \]
\[ E[cc \ M] \rightarrow E[M (\lambda x. E[x])] \]
\[ E[C \ M] \rightarrow M (\lambda x. E[x]) \]

In presence of \text{abort()}, \text{cc} and \text{C} are interdefinable:

\[ C \ M := \text{cc} (\lambda k. \text{abort}(M \ k)) \quad k \not\in \text{FV}(M) \]
\[ \text{cc} \ M := C (\lambda k. k \ (M \ k)) \quad k \not\in \text{FV}(M) \]

Griffin [1990]:

\text{cc} can be typed by \((\neg\neg A) \rightarrow A\)

\text{C} can be typed by \((\neg\neg A) \rightarrow A\)
Central question about control

\[
\begin{align*}
E[\text{abort}(M)] & \rightarrow M \\
E[\text{cc } M] & \rightarrow E[M (\lambda x. E[x])] \\
E[C M] & \rightarrow M (\lambda x. E[x])
\end{align*}
\]

Above rules are not “standard” rewrite rules…

What exactly does \( E[\ ] \) stand for / range over?

More fundamentally:

What kind of continuation can be captured by a control operator?

Is the capture delimited? undelimited? etc
One proposed formalisation: Parigot’s $\lambda\mu$-calculus Parigot [1992]

Terms

\[ M, N, P \ldots ::= x \mid \lambda x. M \mid M N \mid \mu\alpha.c \]

Commands

\[ c ::= [\alpha]M \]

\[ \Gamma, x : A \vdash x : A; \Delta \]
\[ \Gamma, x : A \vdash M : B; \Delta \]
\[ \Gamma \vdash \lambda x. M : A \rightarrow B; \Delta \]
\[ \Gamma \vdash \lambda x. M : A \rightarrow B; \Delta \]
\[ \Gamma \vdash M N : B; \Delta \]
\[ c : (\Gamma \vdash ; \alpha : A, \Delta) \]
\[ \Gamma \vdash \mu\alpha.c : A; \Delta \]
\[ \Gamma \vdash \mu\alpha.c : A; \Delta \]
\[ [\alpha]M : (\Gamma \vdash ; \alpha : A, \Delta) \]
Parigot’s $\lambda\mu$-calculus Parigot [1992]

Extra reduction rules:

\[
(\mu\alpha.c) \; N \quad \longrightarrow \quad \mu\beta.\{\beta^M \; N_{[\alpha]^M}\} \; c
\]

\[
[\beta]\mu\alpha.c \quad \longrightarrow \quad \{\beta/\alpha\} \; c
\]

Integrates Peirce’s law: \(cc := \lambda x.\mu\alpha.[\alpha](x \; \lambda y.\mu\beta.[\alpha]y) : ((A \rightarrow B) \rightarrow A) \rightarrow A\)

Consider that contexts are of the form \(E[] = [\gamma][[] \; N_1 \ldots \; N_n]\)

If given a top-level continuation variable variable \(\text{top} : \bot\) (Ariola-Herbelin Ariola and Herbelin [2003]),

- integrates “Ex falso quod libet” \(\lambda x.\mu\alpha.[\text{top}]x : \bot \rightarrow A\)

- integrates DNE \(C := \lambda x.\mu\alpha.[\text{top}](x \; \lambda y.\mu\beta.[\alpha]y) : (\neg\neg A) \rightarrow A\)

So far, so good
Symmetry

There's something symmetric about classical logic:

- Boolean algebras
- De Morgan duality:

\[ \neg(A \land B) = \neg A \lor \neg B \]
\[ \neg(A \lor B) = \neg A \land \neg B \]

- Classical Sequent calculus LK
  
  left-contraction symmetric to right-contraction

So far, not explicit in our proof theory + term calculi

Filinski Filinski [1989]: first formalisation of a duality between

- functions as values
- functions as continuations

*Symmetric \( \lambda \)-calculus*, with explicit conversions from one view to the other

No explicit connection with logic. Is there one?
Yes, there’s one.

See BB’s symmetric $\lambda$-calculus Barbanera and Berardi [1996]: Natural Deduction with continuations

It is more like a one-sided sequent calculus

Symmetry of the calculus corresponds to symmetry/duality of LK

Other calculi for (bi-sided) versions of LK, with cut-elimination as computation:

- Urban’s calculus Urban [2000],
- Curien-Herbelin’s $\tilde{\lambda}\tilde{\mu}\tilde{\mu}$ Curien and Herbelin [2000] for $\Rightarrow$ (easier in bi-sided sequent calculus),
  later extended by Wadler Wadler [2003] for $\land$ and $\lor$ (connecting to De Morgan)
Two independent works

Curien-Herbelin’s aim:
Express duality of computation syntactically (with a Filinski-like calculus)
Semantics, no proof of SN.

Urban’s aim:
Have a typing system as close as possible to LK, have a reduction system as close as possible to basic cut-elimination procedures
SN, but no semantics.

Then: a broad literature on comparing such calculi.
Curien-Herbelin-Wadler - syntax

\[
\begin{align*}
t & ::= x \mid \mu \beta.c \mid \lambda x.t \mid \langle t_1, t_2 \rangle \mid \text{inj}_i(t) \\
e & ::= \alpha \mid \mu x.c \mid t::e \mid \langle e_1, e_2 \rangle \mid \text{inj}_i(e) \\
c & ::= \langle t \bullet e \rangle
\end{align*}
\]

Intuition:

- \( x, y, \ldots \): inputs (variables standing for terms)
- \( \alpha, \beta, \ldots \): outputs (variables standing for continuations)
- \( \text{terms} = \) some inputs (free term variables)
  + one main output
  + alternative outputs (free continuation variables)
- \( \text{continuations} = \) one main input
  + additional inputs (free term variables)
  + some outputs (free continuation variables)
- \( \text{commands} = \) a term facing a continuation (this interaction creates computation)
\[
\begin{align*}
\Gamma, x : A & \vdash x : A ; \Delta \\
\Gamma, x : A & \vdash t : B ; \Delta \\
\Gamma & \vdash \lambda x.t : A \to B ; \Delta \\
\Gamma & \vdash t_1 : A_1 ; \Delta \\
\Gamma & \vdash t_2 : A_2 ; \Delta \\
\Gamma & \vdash \langle t_1, t_2 \rangle : A_1 \land A_2 ; \Delta \\
\Gamma & \vdash t : A_i ; \Delta \\
\Gamma & \vdash \text{inj}_i(t) : A_1 \lor A_2 ; \Delta \\
\Gamma & \vdash \mu \alpha.c : A ; \Delta \\
\Gamma & \vdash t : A ; \Delta \\
\Gamma & \vdash e : A \vdash \Delta \\
\langle t \cdot e \rangle & : (\Gamma \vdash \Delta)
\end{align*}
\]
Example: Law of Excluded Middle

A story: The devil, the fool, and the $1000000. (borrowed from Phil Wadler)

- I have an offer for you! My promise is:

  Either I offer you $1000000
  or, if you give me $1000000
  then I will grant you any wish

  I choose to offer you the latter.

- Here’s $1000000! I want immortality.

- Well done and thank you!

Now, I’ve changed my mind.

I’ve now decided to fulfill my promise by offering you $1000000.

Here is your money back!
Questions?
References


