Programming = proving?
The Curry-Howard correspondence today

Fourth lecture

You've got to decide one way or the other! Classical logic, continuations, and control operators

Xavier Leroy

Collège de France

2018-12-05

COLLÈGE
DE FRANCE
1530

Today's lecture

Let's try to answer the question

> *If intuitionistic logic corresponds to typed lambda-calculus, what does classical logic corresponds to?*

For this purpose:

- Understand classical logic in relation with intuitionistic logic.
- Understand "control operators" (`call/cc` et al) ($\approx$ the `goto` of lambda-calculus).

I

Classical logic, intuitionistic logic

Classical logic, intuitionistic logic

The first lecture introduced intuitionistic logic as classical logic "minus" some reasoning principles:

- excluded middle $P \vee \neg P$, that is, every P is either true or false;
- $\neg\neg P \Rightarrow P$, that is, proof by contradiction.

This idea offended many mathematicians, starting with Hilbert, who wrote in 1927, criticizing Brouwer's ideas:

> *Taking the principle of excluded middle from the mathematician would be the same, say, as proscribing the telescope to the astronomer or to the boxer the use of his fists. To prohibit existence statements and the principle of excluded middle is tantamount to relinquishing the science of mathematics altogether.*
>
> (Hilbert, *The foundations of mathematics*, 1927; in van Heijenoort, *From Frege to Gödel: A Source Book in Mathematical Logic*, 1976)

Classical logic, intuitionistic logic

Let's offend even more: take intuitionistic logic as a well-understood starting point and use it to better understand classical logic.

- Classical logic as intuitionistic logic "plus" some laws and some symmetries.
- Classical logic as a fragment of intuitionistic logic (modulo encodings).

Classical laws

The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical laws

The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical laws

The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical laws

The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical laws

The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical laws

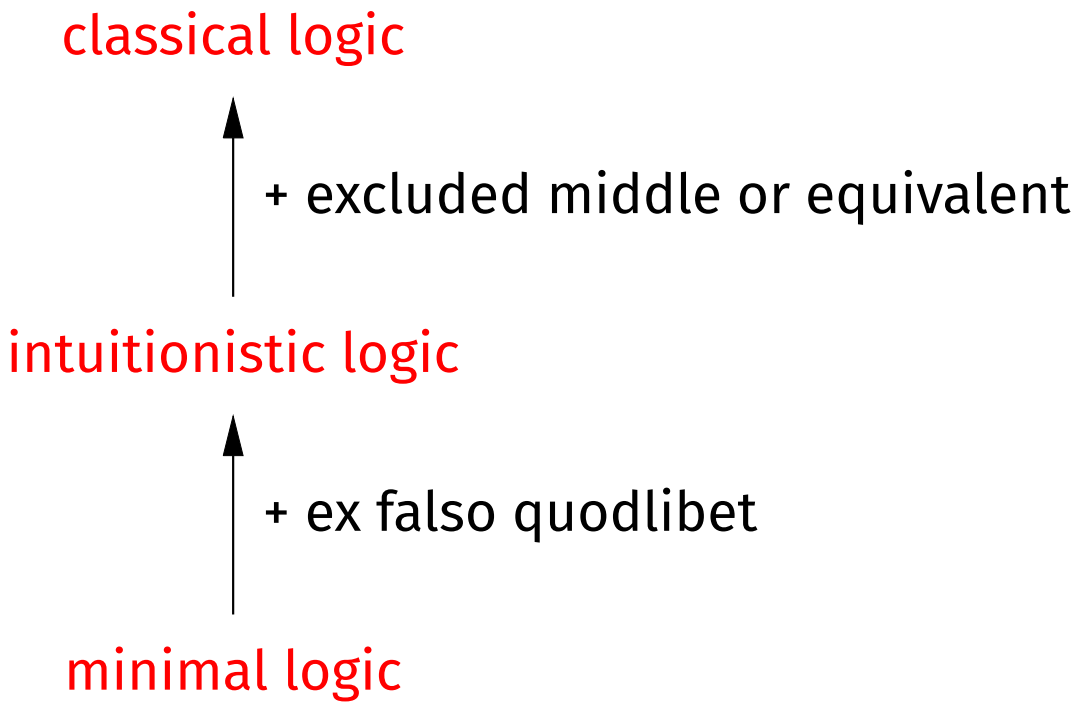
The propositions below ($\forall P, Q, R$) are:

- true in classical logic (as show by their truth tables);
- not provable but all equivalent in intuitionistic logic.

$P \vee \neg P$	excluded middle, *tertium non datur*, *tiers exclu*
$\neg\neg P \Rightarrow P$	double negation elimination
$(\neg P \Rightarrow P) \Rightarrow P$	*mirabile consequentia*, Clavius' law
$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	Peirce's law
$(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$	counterexample principle
$P \vee (P \Rightarrow Q)$	Tarski's formula
$(P \Rightarrow Q) \vee (Q \Rightarrow R)$	linearity principle

A vision: classical logic = intuitionistic logic + (one of) those laws.

Classical, intuitionistic, minimal logics



Classical laws

In minimal logic (without *ex falso quodlibet*, $\bot \Rightarrow P$), those classical laws are not all equivalent:

double negation elimination $\neg\neg P \Rightarrow P$ $\Leftrightarrow$ counterexample principle $(\neg(P \Rightarrow Q)) \Rightarrow P \wedge \neg Q$

$\Downarrow$

Peirce's law $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$ $\Leftrightarrow$ Tarski's formula $P \vee (P \Rightarrow Q)$ $\Leftrightarrow$ linearity $(P \Rightarrow Q) \vee (Q \Rightarrow R)$

$\Downarrow$

excluded middle $P \vee \neg P$ $\Leftrightarrow$ Clavius' law $(\neg P \Rightarrow P) \Rightarrow P$

$\Downarrow$

weak excluded middle $\neg P \vee \neg\neg P$ $\Leftrightarrow$ de Morgan's law $\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$

(Diener and McKubre-Jordens, *Classifying Material Implications over Minimal Logic*, 2018)

de Morgan's laws and duality

The laws connecting "and", "or", "not" (de Morgan's laws) do not all hold in intuitionistic logic:

Classical	Intuitionistic
$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$	$\Leftarrow$
$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$	$\Leftrightarrow$
$\neg(\neg P \wedge \neg Q) \Leftrightarrow P \vee Q$	$\Leftarrow$
$\neg(\neg P \vee \neg Q) \Leftrightarrow P \wedge Q$	$\Leftarrow$

A vision: classical logic = duality between $\wedge$ *and* $\vee$ *via* $\neg$

Translations classical logic $\longrightarrow$ intuitionistic logic

A vision: intuitionistic logic as a mean to study classical logic.

An approach initiated by Gödel circa 1933 to show the consistency of classical arithmetic (Peano's arithmetic):

- Define a translation $[\![\cdot]\!]$ for logic formulas φ such that:
  if $C.L. \vdash \varphi$ then $I.L. \vdash [\![\varphi]\!]$;
  if φ is a contradiction then $[\![\varphi]\!]$ is a contradiction.
- Prove consistency of intuitionistic arithmetic (Heyting's arithmetic) using BHK-style interpretations.

The $[\![\cdot]\!]$ translations are generally based on double negation.

Double negation

Double negation $\neg\neg P$ reads as "P is not wrong".

In classical logic, $\neg\neg P$ is equivalent to P,
and "it's not wrong" is equivalent to "it's true".

In intuitionistic logic, P implies $\neg\neg P$ but the converse does not hold (no double negation elimination). Hence, "it's not wrong" is weaker than "it's true".

However, the intuitionistic "it's not wrong" behaves much like the classical "it's true". In particular, excluded middle is not wrong!

Double negation

Theorem (Excluded middle is not wrong)

$\neg\neg(P \vee \neg P)$ *holds in intuitionistic logic.*

Proof.

Assume $\neg(P \vee \neg P)$ and show absurdity $\bot$.
By contraposition (if $A \Rightarrow B$ then $\neg B \Rightarrow \neg A$) and by $P \Rightarrow P \vee \neg P$, we derive $\neg P$.
By contraposition and by $\neg P \Rightarrow P \vee \neg P$ we derive $\neg\neg P$.
Absurdity follows. $\square$

Negative translation: the propositional case

Theorem (Glivenko, 1929)

Let Φ be a propositional formula.
$C.L. \vdash \Phi$ if and only if $I.L. \vdash \neg\neg\Phi$.

Proof.

If $I.L. \vdash \neg\neg\Phi$, then $C.L. \vdash \neg\neg\Phi$, hence, classically, $C.L. \vdash \varphi$.

Conversely, assume $C.L. \vdash \Phi$. Let $P_1, \ldots, P_n$ be the variables of Φ. Using the truth table technique,

$$\Psi \stackrel{def}{=} P_1 \vee \neg P_1 \Rightarrow \cdots \Rightarrow P_n \vee \neg P_n \Rightarrow \Phi$$

is true in I.L., hence its double negation $\neg\neg\Psi$ is true as well. Yet,

$$\neg\neg\Psi \iff (\neg\neg(P_1 \vee \neg P_1) \Rightarrow \cdots \Rightarrow \neg\neg(P_n \vee \neg P_n) \Rightarrow \neg\neg\Phi)$$

The premises $\neg\neg(P_i \vee \neg P_i)$ being true in I.L., we have $I.L. \vdash \neg\neg\Phi$. □

Negative translation: handling quantifiers

With the quantifiers $\forall$, $\exists$, adding $\neg\neg$ at the head of the formula is not enough. Kolmogorov (1925) adds $\neg\neg$ to every sub-formula.

$$[\![A]\!] = \neg\neg A \quad \text{if } A \text{ is atomic} \qquad [\![P \Rightarrow Q]\!] = \neg\neg([\![P]\!] \Rightarrow [\![Q]\!])$$
$$[\![P \wedge Q]\!] = \neg\neg([\![P]\!] \wedge [\![Q]\!]) \qquad [\![P \vee Q]\!] = \neg\neg([\![P]\!] \vee [\![Q]\!])$$
$$[\![\forall x.\, P]\!] = \neg\neg\forall x.\, [\![P]\!] \qquad [\![\exists x.\, P]\!] = \neg\neg\exists x.\, [\![P]\!]$$

Theorem

If $C.L. \vdash P$ then $I.L. \vdash [\![P]\!]$.

Since $\neg\neg\bot \Rightarrow \bot$, it follows that $C.L. \vdash \bot$ implies $I.L. \vdash \bot$.
Therefore, if I.L. is consistent, C.L. is consistent.

Negative translation: variants

Gödel (1933) and Gentzen (1936) use a more "thrifty" translation, $\neg\neg$ being unnecessary in front of $\wedge$, $\Rightarrow$ and $\forall$.

$$\begin{aligned} [\![A]\!] &= \neg\neg A & [\![P \Rightarrow Q]\!] &= [\![P]\!] \Rightarrow [\![Q]\!] \\ [\![P \wedge Q]\!] &= [\![P]\!] \wedge [\![Q]\!] & [\![P \vee Q]\!] &= \neg\neg([\![P]\!] \vee [\![Q]\!]) \\ [\![\forall x.\, P]\!] &= \forall x.\, [\![P]\!] & [\![\exists x.\, P]\!] &= \neg\neg\exists x.\, [\![P]\!] \end{aligned}$$

Kuroda (1951) is even more "thrifty": $[\![P]\!] = \neg\neg P^*$, where

$$\begin{aligned} A^* &= A & (P \Rightarrow Q)^* &= P^* \Rightarrow Q^* \\ (P \wedge Q)^* &= P^* \wedge Q^* & (P \vee Q)^* &= P^* \vee Q^* \\ (\forall x.\, P)^* &= \forall x.\, \neg\neg P^* & (\exists x.\, P)^* &= \exists x.\, P^* \end{aligned}$$

(G. Ferreira, P. Oliva. *On Various Negative Translations*. EPTCS 47, 2011.)

Relative negation

As introduced by H. Friedman:

Replace absurdity $\bot$ by a proposition R of our choosing.
Replace negation $\neg P \stackrel{def}{=} P \Rightarrow \bot$ by relative negation

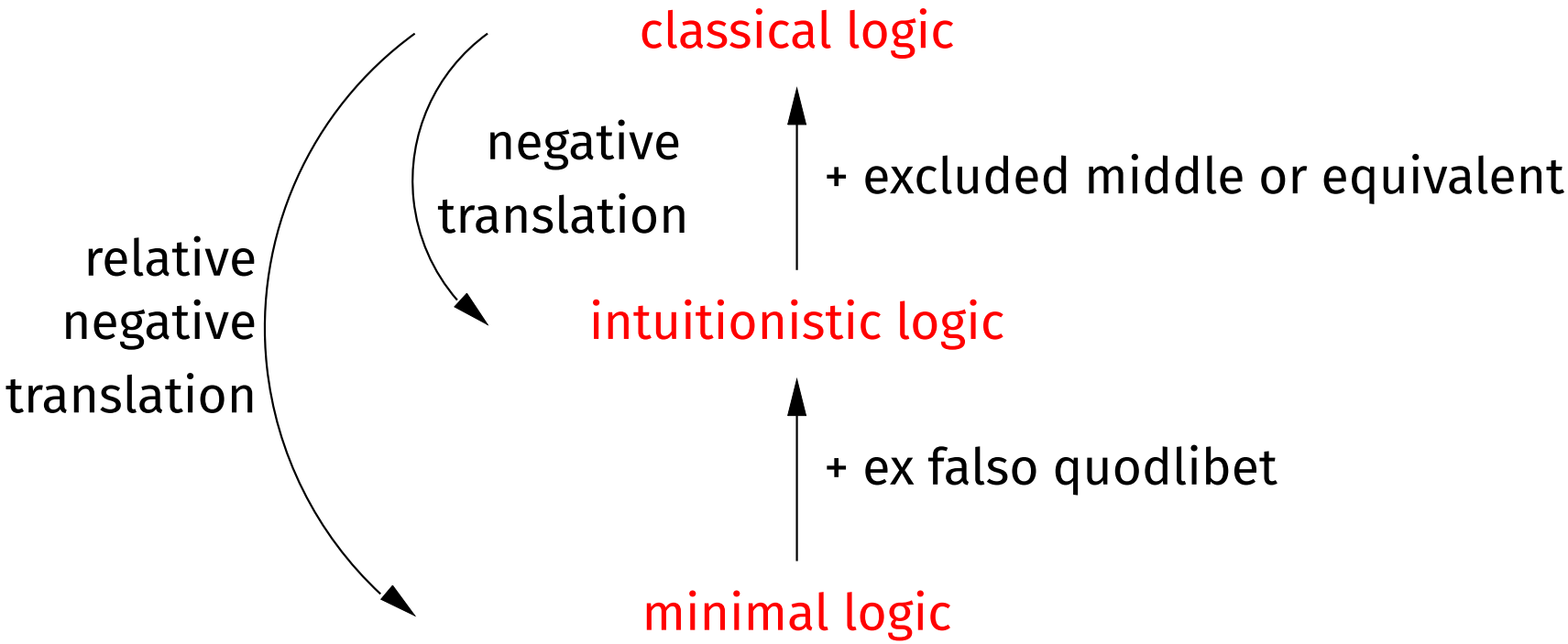
$$\neg_R P \stackrel{def}{=} P \Rightarrow R$$

This gives rise to a relative negative translation:

$$\begin{aligned}
[\![A]\!]_R &= \neg_R \neg_R A & [\![P \Rightarrow Q]\!]_R &= \neg_R \neg_R ([\![P]\!]_R \Rightarrow [\![Q]\!]_R) \\
[\![P \wedge Q]\!]_R &= \neg_R \neg_R ([\![P]\!]_R \wedge [\![Q]\!]_R) & [\![P \vee Q]\!]_R &= \neg_R \neg_R ([\![P]\!]_R \vee [\![Q]\!]_R) \\
[\![\forall x.\, P]\!]_R &= \neg_R \neg_R \forall x.\, [\![P]\!]_R & [\![\exists x.\, P]\!]_R &= \neg_R \neg_R \exists x.\, [\![P]\!]_R
\end{aligned}$$

It is apparent that we now target minimal logic
(without the $\bot$ symbol nor its rule ex falso quodlibet).

A panorama of negative translations



Relative double negation

Preserves the nice properties of double negation:
(these properties are provable in minimal logic)

$$
\begin{array}{rcl}
P & \Rightarrow & \neg_R \neg_R P \\
\neg_R \neg_R \neg_R \neg_R P & \Rightarrow & \neg_R \neg_R P \\
\neg_R \neg_R P \Rightarrow \neg_R \neg_R Q & \Leftrightarrow & \neg_R \neg_R (P \Rightarrow Q) \\
\neg_R \neg_R P \wedge \neg_R \neg_R Q & \Leftrightarrow & \neg_R \neg_R (P \wedge Q) \\
\neg_R \neg_R P \vee \neg_R \neg_R Q & \Rightarrow & \neg_R \neg_R (P \vee Q)
\end{array}
$$

Validates “relative excluded middle”: $\neg_R \neg_R (P \vee \neg_R P)$.

Validates elimination of the double negation of R: $\neg_R \neg_R R \Rightarrow R$.

Conservativity

Theorem

If C.L. $\vdash P$ *then I.L.* $\vdash [\![P]\!]_R$.

Corollary

If $[\![R]\!]_R$ *implies R, and if C.L.* $\vdash R$*, then I.L.* $\vdash R$.

The hypothesis "$[\![R]\!]_R$ implies R" holds in many cases:

- R is an atomic formula A of the shape $f(x_1, \ldots, x_n) = 0$
- R is $A_1 \vee A_2$
- R is $\exists x_1, \ldots, x_n . A$ (Σ^0_1 formula)
- R is $\forall x_1, \ldots x_n . \exists y_1 \ldots y_m . A$ (Π^0_2 formula).

Theorem (Friedman)

Any Π^0_2 *formula provable in classical arithmetic is provable in intuitionistic arithmetic.*

Conservativity

This is an impressive result, yet it has limitations:

Atomic formulas in arithmetic are all decidable because equivalent to $f(x_1, \ldots, x_n) = 0$ where f is a primitive recursive function. That's why we do not really need excluded middle to reason over Π^0_2 formulas.

In the usual example of a statement where excluded middle is essential:

$$\forall m : TM,\ \texttt{terminates}(m) \vee \neg\texttt{terminates}(m)$$

$\texttt{terminates}(m)$ is not an atomic formula, and the statement is not Π^0_2. Indeed:

$$\texttt{terminates}(m) \stackrel{def}{=} \exists n, \texttt{exec}(m, n) = \texttt{halted}$$
$$\neg\texttt{terminates}(m) \Leftrightarrow \forall n, \texttt{exec}(m, n) \neq \texttt{halted}$$

where $\texttt{exec}(m, n)$ = run machine m for n steps.

II

Continuations and control operators

The concept of continuation

Given a functional program p and a subexpression a of p, the continuation of a is the sequence of computations that remain to be executed, once a is evaluated, to obtain the value of p.

It can be viewed as a function (value of a) $\mapsto$ (value of p).

Example

Consider the program $p = (1 + 2) \times (3 + 4)$, evaluated left-to-right.

The continuation of $a = (1 + 2)$ is $\lambda v.\, v \times (3 + 4)$.

The continuation of $a' = (3 + 4)$ is $\lambda v.\, 3 \times v$.

(But not $\lambda v.\, (1 + 2) \times v$, because $1 + 2$ has already been evaluated to 3.)

Continuations and evaluation strategies

In a semantics for a programming language, making continuations explicit enables us to define precisely the evaluation strategy, that is, the order in which computations are performed.

For instance, in functional languages we have two popular evaluation strategies:

- Call by value (CBV):
  The argument N to a function call $(\lambda x.M)\ N$ is reduced to a value (lambda-abstraction or constant) before being bound to parameter x.
- Call by name (CBN):
  The argument N is bound to x unevaluated. N will be evaluated when (and every time) the value of x is needed.

The continuation passing style (CPS) transformation

We can impose an evaluation strategy for a lambda-term M by transforming it into a term $[\![M]\!]$ in continuation passing style. This term $[\![M]\!]$

- expects one argument k representing M's continuation;
- reduces M to a value v;
- and finally applies k to v.

Call by name

$$
\begin{aligned}
[\![cst]\!]_n &= \lambda k.\, k\ cst \\
[\![\lambda x.\, M]\!]_n &= \lambda k.\, k\ (\lambda x.\, [\![M]\!]_n) \\
[\![x]\!]_n &= \lambda k.\, x\ k \\
[\![M\ N]\!]_n &= \lambda k.\, [\![M]\!]_n\ (\lambda f.\, f\ [\![N]\!]_n\ k)
\end{aligned}
$$

The continuation passing style (CPS) transformation

We can impose an evaluation strategy for a lambda-term M by transforming it into a term $[\![M]\!]$ in continuation passing style. This term $[\![M]\!]$

- expects one argument k representing M's continuation;
- reduces M to a value v;
- and finally applies k to v.

Call by name

$$
\begin{aligned}
[\![cst]\!]_n &= \lambda k.\, k\ cst \\
[\![\lambda x.\, M]\!]_n &= \lambda k.\, k\ (\lambda x.\, [\![M]\!]_n) \\
[\![x]\!]_n &= \lambda k.\, x\ k \\
[\![M\ N]\!]_n &= \lambda k.\, [\![M]\!]_n\ (\lambda f.\, f\ [\![N]\!]_n\ k)
\end{aligned}
$$

Call by value

$$
\begin{aligned}
[\![cst]\!]_v &= \lambda k.\, k\ cst \\
[\![\lambda x.\, M]\!]_v &= \lambda k.\, k\ (\lambda x.\, [\![M]\!]_v) \\
[\![x]\!]_v &= \lambda k.\, k\ x \\
[\![M\ N]\!]_v &= \lambda k.\, [\![M]\!]_v\ (\lambda f.\, \\
&\qquad [\![N]\!]_v\ (\lambda a.\, \\
&\qquad\quad f\ a\ k))
\end{aligned}
$$

The CPS transformation

The CPS transformation preserves the expected semantics, while being indifferent to evaluation order:

- $M \xrightarrow{*} cst$ in call by value iff $[\![M]\!]_v\ (\lambda x.\, x) \xrightarrow{*} cst$ in any strategy.
- $M \xrightarrow{*} cst$ in call by name iff $[\![M]\!]_n\ (\lambda x.\, x) \xrightarrow{*} cst$ in any strategy.

Control operators

Control operators equip functional languages with the ability to reify continuations as first-class values, enabling programs to manipulate their own continuations.

The oldest control operator is Landin's `J`
(*A generalization of Jumps and Labels*, 1965):

$$f = \lambda x.\, \texttt{let}\ g = \texttt{J}(\lambda y.\, N)\ \texttt{in}\ M$$

The `J` operator modifies the local function $g = \lambda y.\, N$ in such a way that when g is called inside M, it returns directly to the caller of f.

The best known control operator is `call/cc` (*call with current continuation*) in the Scheme language.

The `callcc` operator

The expression `callcc`$(\lambda k.\, M)$ evaluates as follows:

- The continuation of this expression is bound to variable k.
- M is evaluated; its value is the value of `callcc`$(\lambda k.\, M)$.
- If, during the evaluation of M or at any later time, k is applied to a value v, evaluation continues as if `callcc`$(\lambda k.\, M)$ returned value v.

  In other words, the continuation of the `callcc` expression is reinstated and restarted with v as the result for the expression.

Example of use

Libraries for lists, sets, and other collections often provide an imperative iterator `iter`, such as

```
(* list_iter: ('a -> unit) -> 'a list -> unit *)

let rec list_iter f l =
    match l with
    | [] -> ()
    | head :: tail -> f head; list_iter f tail
```

Example of use

Using a control operator, an imperative iterator can be reused as a function that returns the first element of a collection satisfying the predicate `pred`.

```
let find pred lst =
    callcc (λk.
        list_iter
            (λx. if pred x then k (Some x) else ())
            lst;
        None)
```

If iteration hits an element `x` such that `pred x = true`,
the application of `k` causes `Some x` to be immediately returned as the result of `find pred lst`.

If no such element `x` exists, `list_iter` terminates normally and `None` is returned.

Example of use

The previous example can be implemented with exceptions. However, `callcc` adds the ability to restart the search.

```
let find pred lst =
    callcc (λk.
        list_iter
            (λx. if pred x
                 then callcc (λk'. k (Some(x, k')))
                 else ())
            lst;
        None)
```

When we find `x` such that `pred x = true`, `find` returns not just `x` but also a continuation `k'` that can *backtrack* the search, restarting it on the element that follows `x`.

Example of use

Iterating `find` as follows, we can print all elements of the collection that satisfy the predicate:

```
let printall pred lst =
    match find pred list with
    | None -> ()
    | Some(x, k) -> print_string x; k ()
```

The application `k ()` restarts `find pred list` where it stopped last.

Semantics of `callcc`

The CPS transformation extends easily to control operators like `callcc`. This gives a semantics to these operators, but also provides an implementation for them, in a compiler or via manual CPS transformation.

$$[\![\texttt{callcc}\ M]\!]_v = \lambda k.\ [\![M]\!]_v\ (\lambda f.\ f\ (\lambda v.\lambda k'.\ k\ v)\ k)$$

In `callcc` M, the function value f of M receives the current continuation k both as argument (wrapped in $\lambda v.\lambda k'.\ k\ v$) and as continuation.

When the captured continuation k is restarted on a value v, the current continuation k' at that time is ignored.

Other control operators

Introduced by Felleisen circa 1986, the operator $\mathcal{C}$ is a simplified version of `callcc` where the current continuation, once captured, is replaced by the initial continuation. We have

$$\texttt{callcc}(\lambda k.\, M) = \mathcal{C}(\lambda k.\, k\, M)$$

The operators $\mathcal{F}$ / # (Felleisen et al, 1987), `shift` / `reset` (Danvy and Filinski, 1990), `cupto` (Gunter et al, 1995), etc, provide ways to capture delimited continuations

value of an expression $\mapsto$ value of an enclosing expression

instead of full continuations

value of an expression $\mapsto$ value of the whole program

III

Correspondences between classical logic and control operators

CPS transformation and negative translation

(Chetan Murthy, *Extracting Constructive Content from Classical Proofs*, PhD, Cornell, 1990.)

In his PhD, Chetan Murthy demonstrates a correspondence (in the style of Curry-Howard) between

- call-by-name CPS transformation;
- Kolmogorov's negative translation, relativized by Friedman.

More precisely: the negative translation describes the effect of the CPS transformation over types.

Typing call-by-name CPS transformation

Let r be the type of the whole program. We define the transformation of simple types:

$$[\![t]\!] = (t^* \to r) \to r \qquad \iota^* = \iota \qquad (t \to s)^* = [\![t]\!] \to [\![s]\!]$$

Intuition: a term of type t becomes a function $\lambda k \ldots$
The continuation k expects a value of type t^* and produces the program result, with type r. Hence, $k : t^* \to r$ and the transformed term has type $(t^* \to r) \to r$.

Theorem (CPS transformation preserves typing)

If $\ldots\ x_i : t_i\ \ldots \vdash M : t$, *then* $\ldots\ x_i : [\![t_i]\!]\ \ldots \vdash [\![M]\!]_n : [\![t]\!]$.

Correspondence with the negative translation

This transformation of simple types corresponds to Kolmogorov's negative translation, made relative, for the $\Rightarrow$ fragment:

$$\begin{aligned} [\![A]\!] &= \neg_R \neg_R A \quad \text{if } A \text{ is atomic} \\ [\![P \Rightarrow Q]\!] &= \neg_R \neg_R ([\![P]\!] \Rightarrow [\![Q]\!]) \end{aligned}$$

or, equivalently,

$$[\![P]\!] = \neg_R \neg_R P^* = (P^* \Rightarrow R) \Rightarrow R$$

$$\begin{aligned} A^* &= A \quad \text{if } A \text{ is atomic} \\ (P \Rightarrow Q)^* &= [\![P]\!] \Rightarrow [\![Q]\!] \end{aligned}$$

Correspondence with the negative translation

The correspondence extends to the other logical connectives ($\wedge, \vee, \bot, \forall, \exists$) just like CPS transformation extends to system F and to inductive types, and therefore to types $\times$, $+$, Σ, and to the empty type.

$$[\![\Lambda X.\, M]\!]_n = \lambda k.\, k\, (\Lambda X.\, [\![M]\!]_n)$$

$$[\![M[t]\,]\!]_n = [\![M]\!]_n\, (\lambda x.\, x[t]\; k)$$

$$[\![C\; M_1\; \cdots\; M_p]\!]_n = \lambda k.\, k\, (C\; [\![M_1]\!]_n\; \cdots\; [\![M_p]\!]_n)$$

$$[\![\texttt{match}\; M\; \texttt{with}\; \cdots \mid C_i\; \vec{x}_i \Rightarrow N_i \mid \cdots]\!]_n = \lambda k.\, M\, (\lambda x.\, \texttt{match}\; x\; \texttt{with}\; \cdots \mid C_i\; \vec{x}_i \Rightarrow [\![N_i]\!]_n\; k \mid \cdots)$$

Typing the call-by-value CPS transformation

In call-by-value, the type translation differs slightly:

$$[\![t]\!] = (t^* \to r) \to r \qquad \iota^* = \iota \qquad (t \to s)^* = t^* \to [\![s]\!]$$

A transformed function expects an argument that is already evaluated, hence of type t^*, instead of an argument that remains to be evaluated ($\lambda k \ldots$) which would have type $[\![t]\!]$.

The type preservation results extend to call-by-value CPS transformation.

Correspondence with the negative translation

This call-by-value transformation of simple types corresponds to a variant of Kuroda's "thrifty" translation:

$$[\![P]\!] = \neg_R \neg_R P^* = (P^* \Rightarrow R) \Rightarrow R$$

$$\begin{aligned} A^* &= A \quad \text{if } A \text{ atomic} \\ (P \Rightarrow Q)^* &= P^* \Rightarrow [\![Q]\!] \qquad \text{(call-by-value CPS)} \\ (P \Rightarrow Q)^* &= P^* \Rightarrow Q^* \qquad \text{(Kuroda)} \end{aligned}$$

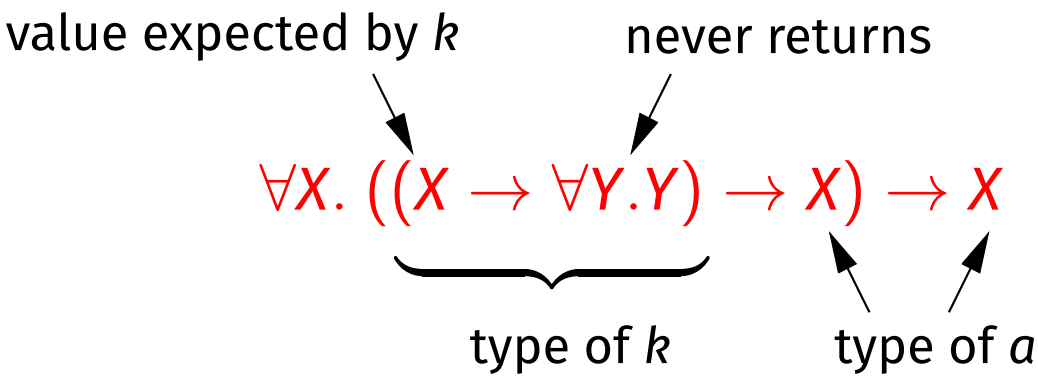
Viewed as program transformations, Kuroda's and Gödel-Gentzen's negative translations are not interesting: they preserve types but not dynamic semantics.

Control operators and classical laws

(Timothy Griffin, *A Formulae-as-Types Notion of Control*, POPL 1990.)

In 1989, Griffin observes that the control operators `callcc` and $\mathcal{C}$ have types that correspond to laws of classical logic.

For instance, the type of `callcc` is: (consider `callcc`$(\lambda k.\, a)$)

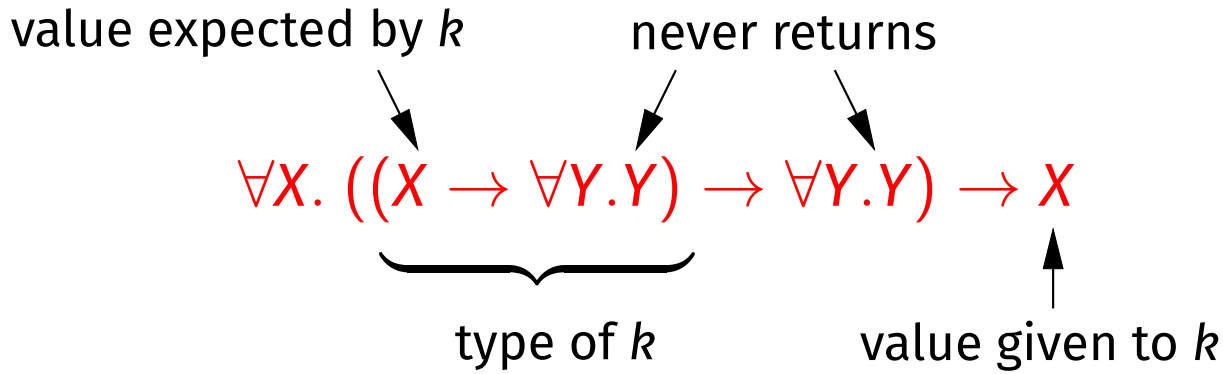


If we read $\forall Y.Y$ as $\bot$ and $X \to \forall Y.Y$ as $\neg X$,
this is Clavius' law, *mirabile consequentia*:

$$\forall P.\, (\neg P \Rightarrow P) \Rightarrow P$$

Control operators and classical laws

Likewise, operator $\mathcal{C}$ has type (consider $\mathcal{C}(\lambda k.\, a)$)



If we read $T \to \forall Y.Y$ as $\neg T$, this is double negation elimination:

$$\forall P.\, \neg\neg P \Rightarrow P$$

A “construction” of excluded middle

We saw that, in minimal logic, Clavius’ law implies excluded middle.

From the `callcc` operator whose type is Clavius’ law, we can therefore construct a term whose type is excluded middle:

$$\Lambda P.\ \texttt{callcc}\,(\lambda k : P \vee \neg P \Rightarrow \bot.\ \texttt{inj}_2\,(\lambda p : P.\ k\,(\texttt{inj}_1(p))))$$

$$: \forall P.\ P \vee \neg P$$

What is the dynamic behavior of this term??

A Faustian bargain

(from Philip Wadler, *Call-by-Value is Dual to Call-by-Name*, 2003)

THE DEVIL: Here is my offer. Either (a) I will give you one million euros, or (b) I will grant you any wish if you pay me one million euros.

FAUST: No other conditions? Do I need to sign over my soul?

THE DEVIL: Keep it. But I get to choose whether I offer (a) or (b).

FAUST: I accept. Do I get (a) or (b)?

THE DEVIL: I choose (b).

Many years later, Faust returns with one million euros and gives it to the devil.

FAUST: Grant me my wish!

THE DEVIL: Oh, did I say (b) before? I'm so sorry. I meant (a). It is my great pleasure to give you one million euros.

A “construction” of excluded middle

$$\texttt{callcc}\,(\lambda k : P \vee \neg P \Rightarrow \bot.\ \texttt{inj}_2\,(\lambda p : P.\ k\,(\texttt{inj}_1(p))))$$

The term returns $\texttt{inj}_2$, that is, it chooses the $\neg P$ case.

It also returns a “construction” of $\neg P$, that is, a function $\lambda p : P \ldots$ of type $P \to \bot$.

If the remainder of the proof does not use this claim $\neg P$, everything is fine.

If the remainder of the proof uses this claim, it is by “eliminating” the negation, that is, by passing a proof $p : P$ to the function $\lambda p : P \ldots$ so as to obtain a proof of $\bot$.

At that time, the continuation k is invoked with $\texttt{inj}_1(p)$: we backtrack to the choice point, we choose the P case, and we provide p as construction.

Exercise: which Faustian bargain is played by double negation elimination?

A “construction” of excluded middle

In the first lecture, we showed that there exists no construction (in the sense of the BHK interpretation) for excluded middle, because such a construction would decide the halting problem:

$$\forall m : TM,\ \texttt{terminates}(m) \vee \neg\texttt{terminates}(m)$$

What is going on here?

Logical inconsistency? No!

System F + `callcc` is normalizing.

A different notion of construction? Yes!

The term $\texttt{callcc}(\lambda k \ldots)$ has the right type but can reduce either to $\texttt{inj}_2(np)$ or to $\texttt{inj}_1(p)$ depending on the context. This puts this term outside of the BHK interpretation, where a construction for $P \vee Q$ determines once and for all which of the two cases P or Q is proved.

IV

Classical sequents and L-calculi

Duality lost

The vision “classical logic = intuitionistic logic + excluded middle”
destroys an important formal symmetry:
the duality between conjunction and disjunction via negation.

$$\neg(P \wedge Q) = \neg P \vee \neg Q$$
$$\neg(P \vee Q) = \neg P \wedge \neg Q$$

Also lost: the ability to define implication from the other connectives:
$P \Rightarrow Q = \neg P \vee Q = \neg(P \wedge \neg Q)$.

Finally: cut elimination for $\vee$ is much more difficult than for $\Rightarrow$ and $\wedge$.

Duality regained

The classical sequent calculus (Gentzen, 1934, 1935) is a formulation of classical logic that preserves duality between conjunction and disjunction.

A central notion: the classical sequent

$$A_1, \ldots, A_n \vdash B_1, \ldots, B_m \qquad \text{“if } A_1 \text{ and } \ldots \text{and } A_n \text{, then } B_1 \text{ or } \ldots \text{or } B_m\text{”}$$

Compare with intuitionistic sequents:

$$A_1, \ldots, A_n \vdash B \qquad \text{“if } A_1 \text{ and } \ldots \text{and } A_n \text{, then } B\text{”}$$

Each logical connective has “right” rules (if it appears in the conclusions B_i) and “left” rules (if it appears in the hypotheses A_i).

Classical sequent calculus

$$A \vdash A \ (\mathrm{Id}) \qquad \frac{\Gamma \vdash \Delta, A \quad A, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \ (\mathrm{cut})$$

$$\frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B} \ (\wedge R) \qquad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \ (\wedge L_1) \qquad \frac{B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \ (\wedge L_2)$$

$$\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \ (\vee R_1) \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B} \ (\vee R_2) \qquad \frac{A, \Gamma \vdash \Delta \quad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta} \ (\vee L)$$

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \ (\neg R) \qquad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \ (\neg L)$$

$$\frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \Rightarrow B} \ (\Rightarrow R) \qquad \frac{\Gamma \vdash \Delta, A \quad B, \Gamma' \vdash \Delta'}{A \Rightarrow B, \Gamma, \Gamma' \vdash \Delta, \Delta'} \ (\Rightarrow L)$$

(Plus: weakening, contraction, exchange.)

The pleasures of a classical logic

$$\frac{A \vdash A}{\vdash A,\ \neg A}\ (\neg R)$$

From this rule we prove excluded middle and double negation elimination:

$$\dfrac{\dfrac{\dfrac{\vdash A,\ \neg A}{\vdash A,\ A \vee \neg A}\ (\vee R_1)}{\vdash A \vee \neg A,\ A \vee \neg A}\ (\vee R_2)}{\vdash A \vee \neg A}\ (\text{contraction})$$

$$\dfrac{\dfrac{\vdash A,\ \neg A}{\neg\neg A \vdash A}\ (\neg L)}{\vdash \neg\neg A \Rightarrow A}\ (\Rightarrow R)$$

Curry-Howard for classical sequents

Via Curry-Howard, typed lambda-calculus corresponds to intuitionistic sequent calculus.

What is the language that corresponds to classical sequent calculus?

Several proposals: $\lambda\mu$ by Parigot (1992, 1994); $\lambda\mu\tilde{\mu}$ by Curien and Herbelin (2000); the dual calculus by Wadler (2003); the L-calculus by Munch-Maccagnoni et al (2009); etc.

Two core ideas:

1. A computation $C = \langle M \mid K \rangle$ is the interaction between a term M and a context K (also called co-term, stack, or continuation).
2. To reflect the multiple conclusions of a classical sequent, contexts can have several "holes", identified by co-variables α.

A L-calculus for classical sequents

(Taken from Wadler, *Call-by-value is dual to call-by-name*, ICFP 2003.)

Computations: $C ::= \langle M \mid K \rangle$

Terms: $M, N ::= x \mid \ldots$

Contexts: $K, L ::= \alpha \mid \ldots$

Three typing judgments:

- $C : (x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m)$

  "if the x_i are bound to values of types A_i, executing C passes one of the continuations α_j a value of type B_j".

- $x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m \mid M : B$

  "if the x_i are bound to values of types A_i, evaluating M produces a value of type B or passes one of the continuations α_j a value of type B_j".

- $K : A \mid x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m$

  "if the x_i are bound to values of types A_i and if K is passed a value of type A, one of the continuations α_j is passed a value of type B_j".

Typing rules

$$x : A \vdash \ \mid x : A \quad \text{(IdR)} \qquad\qquad \alpha : A \mid \ \vdash \alpha : A \quad \text{(IdL)}$$

$$\frac{\Gamma \vdash \Delta \mid M : A \qquad K : A \mid \Gamma' \vdash \Delta'}{\langle M \mid K \rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')} \text{(cut)}$$

$$\frac{\Gamma \vdash \Delta \mid M : A \qquad \Gamma \vdash \Delta \mid N : B}{\Gamma \vdash \Delta \mid (M, N) : A \wedge B} (\wedge R) \qquad \frac{K : A \mid \Gamma \vdash \Delta}{\pi_1(K) : A \wedge B \mid \Gamma \vdash \Delta} (\wedge L_1) \qquad \frac{K : B \mid \Gamma \vdash \Delta}{\pi_2(K) : A \wedge B \mid \Gamma \vdash \Delta} (\wedge L_2)$$

$$\frac{\Gamma \vdash \Delta \mid M : A}{\Gamma \vdash \Delta \mid \mathtt{inj}_1(M) : A \vee B} (\vee R_1) \qquad \frac{\Gamma \vdash \Delta \mid N : B}{\Gamma \vdash \Delta \mid \mathtt{inj}_2(N) : A \vee B} (\vee R_2) \qquad \frac{K : A \mid \Gamma \vdash \Delta \qquad L : B \mid \Gamma \vdash \Delta}{(K \mid L) : A \vee B \mid \Gamma \vdash \Delta} (\vee L)$$

$$\frac{K : A \mid \Gamma \vdash \Delta}{\Gamma \vdash \Delta \mid \mathtt{not}(K) : \neg A} (\neg R) \qquad \frac{\Gamma \vdash \Delta \mid M : A}{\mathtt{not}(M) : \neg A \mid \Gamma \vdash \Delta} (\neg L)$$

$$\frac{x : A, \Gamma \vdash \Delta \mid N : B}{\Gamma \vdash \Delta \mid \lambda x.N : A \Rightarrow B} (\Rightarrow R) \qquad \frac{\Gamma \vdash \Delta \mid M : A \qquad K : B \mid \Gamma' \vdash \Delta'}{M \bullet K : A \Rightarrow B \mid \Gamma, \Gamma' \vdash \Delta, \Delta'} (\Rightarrow L)$$

$$\frac{C : (\Gamma \vdash \Delta, \alpha : A)}{\Gamma \vdash \Delta \mid \mu\alpha.\, C : A} \text{(actiR)} \qquad \frac{C : (x : A, \Gamma \vdash \Delta)}{\mu x.\, C : A \mid \Gamma \vdash \Delta} \text{(actiL)}$$

(Plus: weakening, contraction, exchange.)

Typing rules

$$A \vdash \mid A \ \text{(IdR)} \qquad A \mid \vdash A \ \text{(IdL)}$$

$$\frac{\Gamma \vdash \Delta \mid A \qquad A \mid \Gamma' \vdash \Delta'}{(\Gamma, \Gamma' \vdash \Delta, \Delta')} \ \text{(cut)}$$

$$\frac{\Gamma \vdash \Delta \mid A \qquad \Gamma \vdash \Delta \mid B}{\Gamma \vdash \Delta \mid A \wedge B} \ (\wedge R) \qquad \frac{A \mid \Gamma \vdash \Delta}{A \wedge B \mid \Gamma \vdash \Delta} \ (\wedge L_1) \qquad \frac{B \mid \Gamma \vdash \Delta}{A \wedge B \mid \Gamma \vdash \Delta} \ (\wedge L_2)$$

$$\frac{\Gamma \vdash \Delta \mid A}{\Gamma \vdash \Delta \mid A \vee B} \ (\vee R_1) \qquad \frac{\Gamma \vdash \Delta \mid B}{\Gamma \vdash \Delta \mid A \vee B} \ (\vee R_2) \qquad \frac{A \mid \Gamma \vdash \Delta \qquad B \mid \Gamma \vdash \Delta}{A \vee B \mid \Gamma \vdash \Delta} \ (\vee L)$$

$$\frac{A \mid \Gamma \vdash \Delta}{\Gamma \vdash \Delta \mid \neg A} \ (\neg R) \qquad \frac{\Gamma \vdash \Delta \mid A}{\neg A \mid \Gamma \vdash \Delta} \ (\neg L)$$

$$\frac{A, \Gamma \vdash \Delta \mid B}{\Gamma \vdash \Delta \mid A \Rightarrow B} \ (\Rightarrow R) \qquad \frac{\Gamma \vdash \Delta \mid A \qquad B \mid \Gamma' \vdash \Delta'}{A \Rightarrow B \mid \Gamma, \Gamma' \vdash \Delta, \Delta'} \ (\Rightarrow L)$$

$$\frac{(\Gamma \vdash \Delta, A)}{\Gamma \vdash \Delta \mid A} \ \text{(actiR)} \qquad \frac{(A, \Gamma \vdash \Delta)}{A \mid \Gamma \vdash \Delta} \ \text{(actiL)}$$

(Plus: weakening, contraction, exchange.)

Syntax of terms

Deduced from the shape of the typing rules!

Computations:	C	$::= \langle M \mid K \rangle$	
Terms:	M, N	$::= x \mid \lambda x.\, M$	variables, function abstractions
		$\mid (M, N)$	product constructor
		$\mid \mathtt{inj}_1(M) \mid \mathtt{inj}_2(M)$	sum constructors
		$\mid \mathtt{not}(K)$	complement of a context
		$\mid \mu\alpha.C$	co-variable abstraction
Contexts:	K, L	$::= \alpha$	co-variable
		$\mid M \bullet K$	function application to M, followed by K
		$\mid \pi_1(K) \mid \pi_2(K)$	projections from a product
		$\mid (K \mid L)$	case analysis over a sum
		$\mid \mathtt{not}(M)$	complement of a term
		$\mid \mu x.C$	variable abstraction

Exercise: which term expresses double negation elimination?

Reduction rules

When a constructor (in the term) meets its destructor (in the context):

$$\langle (M, N) \mid \pi_1(K) \rangle \to \langle M \mid K \rangle \qquad \langle (M, N) \mid \pi_2(K) \rangle \to \langle N \mid K \rangle$$
$$\langle \mathtt{inj}_1(M) \mid (K \mid L) \rangle \to \langle M \mid K \rangle \qquad \langle \mathtt{inj}_2(M) \mid (K \mid L) \rangle \to \langle M \mid L \rangle$$
$$\langle \mathtt{not}(K) \mid \mathtt{not}(M) \rangle \to \langle M \mid K \rangle \qquad \langle \lambda x.\, M \mid N \bullet K \rangle \to \langle N \mid \mu x.\, \langle M \mid K \rangle \rangle$$

Plus: β-reductions for μ binders:

$$\langle M \mid \mu x.\, C \rangle \to C\{x \leftarrow M\} \qquad \langle \mu \alpha.\, C \mid K \rangle \to C\{\alpha \leftarrow K\}$$

Problem: reductions are not confluent!
(For example $\langle \mu \alpha.\, C \mid \mu x.\, C' \rangle$ reduces in two different ways.)

Reduction strategies

We must impose a reduction strategy. For example, we reduce $\langle M \mid K \rangle$

- only if M is a value (in particular: not $\mu\alpha.\, C$)
  $\Rightarrow$ a generalization of call by value
- or only if K is a "co-value" (in particular: not $\mu x.\, C$)
  $\Rightarrow$ a generalization of call by name.

(Other possible approach: by polarization.)

Theorem (Wadler 2003)

With the "by value" strategy, $\neg(A \wedge \neg B)$ behaves like $A \Rightarrow B$.
With the "by name" strategy, $\neg A \vee B$ behaves like $A \Rightarrow B$.

V

Concluding remarks

Does a classical proof has a computational content?

No, according to Girard, Lafont, Taylor (1989).
Cut elimination for classical sequent calculus is not confluent; therefore:

> *[A BHK interpretation] is not possible with classical logic: there is no sensible way of considering proofs as algorithms. In fact, classical logic has no denotational semantics, except the trivial one which identifies all the proofs of the same type. This is related to the nondeterministic behaviour of cut elimination.*

Yes, according to Murthy, Griffin, and the L-calculi people.
But we need at least:

- a fixed reduction strategy (call by name or call by value) to work around the lack of confluence;
- to accept that the "constructions" (proof terms) are not just pure lambda-calculus terms, but can also have effects.

Does a classical proof has a computational content?

No, according to Girard, Lafont, Taylor (1989).
Cut elimination for classical sequent calculus is not confluent; therefore:

> *[A BHK interpretation] is not possible with classical logic: there is no sensible way of considering proofs as algorithms. In fact, classical logic has no denotational semantics, except the trivial one which identifies all the proofs of the same type. This is related to the nondeterministic behaviour of cut elimination.*

Yes, according to Murthy, Griffin, and the L-calculi people.
But we need at least:

- a fixed reduction strategy (call by name or call by value) to work around the lack of confluence;
- to accept that the "constructions" (proof terms) are not just pure lambda-calculus terms, but can also have effects.

VI

Further reading

Further reading

- Programming with continuations and with `callcc`:
  Shriram Krishnamurthi. *Programming Languages: Application and Interpretation*. Chapter 14.
  `http://cs.brown.edu/courses/cs173/2012/book/index.html`

- Connections between classical logic and continuations:
  Morten Heine Sørensen, Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. 1998. Chapter 6.
  `https://disi.unitn.it/~bernardi/RSISE11/Papers/curry-howard.pdf`.

- Classical sequent calculus:
  Jean-Yves Girard.*The blind spot: Lectures on logic*. European Mathematical Society, 2011. Chapter 3.