CSC_51051_EP: Simply typed λ-calculus

Samuel Mimram

2024

École polytechnique

Part I

Introduction

Putting it all together

What we have done so far.

1. We have seen that types in OCaml could intuitively be interpreted as formulas.
2. We have formally defined what is a formula and a proof.
3. We have formally defined the core of a functional language (λ-calculus).

and now we put it all together:

4. We define a typing system for λ-calculus and show that it corresponds precisely to building proofs.

Putting it all together

In other words,

$$\text{PROGRAM} = \text{PROOF}$$

Putting it all together

In other words,

$$\text{PROGRAM} = \text{PROOF}$$

Or, more precisely, there is a bijection between

- types and formulas,
- programs of type A and proofs of A,

Putting it all together

In other words,

$$\text{PROGRAM} = \text{PROOF}$$

Or, more precisely, there is a bijection between

- types and formulas,
- programs of type A and proofs of A,
- reductions of programs and cut elimination.

The core

In order to have things as simple as possible, we will first focus on functions.

But, we will see that it extends to more realistic programming languages.

Part II

Simply typed λ-calculus

Simple types

The **simple types** are generated by the grammar

$$A, B \quad ::= \quad X \mid A \to B$$

where X is a variable.

For instance, we have a type

$$(X \to Y) \to X$$

which roughly corresponds to OCaml's

```
('a -> 'b) -> 'a
```

Simple types

The **simple types** are generated by the grammar

$$A, B \quad ::= \quad X \mid A \to B$$

where X is a variable.

For instance, we have a type

$$(X \to Y) \to X$$

which roughly corresponds to OCaml's

```
('a -> 'b) -> 'a
```

By convention, arrows are associated on the *right*:

$$X \to Y \to Z \qquad = \qquad X \to (Y \to Z)$$

Terms

The programs we considers are λ-**terms** generated by the grammar

$$t, u \quad ::= \quad x \mid t\,u \mid \lambda x^A.t$$

where x is a variable and A is a type.

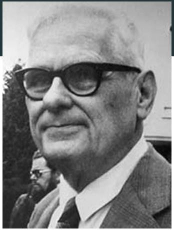
All the abstractions carry the type of the abstracted variable:

$$\lambda x^{\texttt{int}}.x$$

corresponds to OCaml's

```
fun (x : int) -> x
```

Terms



The programs we considers are λ-**terms** generated by the grammar

$$t, u \quad ::= \quad x \mid t\, u \mid \lambda x^A.t$$

where x is a variable and A is a type.

All the abstractions carry the type of the abstracted variable:

$$\lambda x^{\texttt{int}}.x$$

corresponds to OCaml's

```
fun (x : int) -> x
```

This is called **Church style**.

Terms

The programs we considers are λ-**terms** generated by the grammar

$$t, u \quad ::= \quad x \mid t\,u \mid \lambda x \quad .t$$

where x is a variable and A is a type.

All the abstractions carry the type of the abstracted variable:

$$\lambda x \quad .x$$

corresponds to OCaml's

```
fun  x         -> x
```

This is called **Church style** (the other one being *Curry style*).

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	

As usual, we will formulate this by using inference rules on sequents.

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	$A \to A$

As usual, we will formulate this by using inference rules on sequents.

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	$A \to A$
$\lambda f^{A \to A}.\lambda x^A.f(fx)$	

As usual, we will formulate this by using inference rules on sequents.

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	$A \to A$
$\lambda f^{A\to A}.\lambda x^A.f(fx)$	$(A \to A) \to A \to A$

As usual, we will formulate this by using inference rules on sequents.

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	$A \to A$
$\lambda f^{A\to A}.\lambda x^A.f(fx)$	$(A \to A) \to A \to A$
$\lambda x^A.xx$	

As usual, we will formulate this by using inference rules on sequents.

Typing

We are now going assign types to terms. For instance, the type of

term	type
$\lambda x^A.x$	$A \to A$
$\lambda f^{A\to A}.\lambda x^A.f(fx)$	$(A \to A) \to A \to A$
$\lambda x^A.xx$	not well-typed!

As usual, we will formulate this by using inference rules on sequents.

Contexts

A **context** Γ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable x_i and a type A_i.

It can be read as "I assume that the variable x_i has type A_i for every index i".

Contexts

A **context** Γ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable x_i and a type A_i.

It can be read as "I assume that the variable x_i has type A_i for every index i".

The **domain** of the context Γ is $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.

Contexts

A **context** Γ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable x_i and a type A_i.

It can be read as "I assume that the variable x_i has type A_i for every index i".

The **domain** of the context Γ is $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.

Given $x \in \mathrm{dom}(\Gamma)$ we write $\Gamma(x)$ for the type of x:

$$(\Gamma, x : A)(x) = A \qquad\qquad (\Gamma, y : A)(x) = \Gamma(x)$$

Contexts

A **context** Γ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable x_i and a type A_i.

It can be read as "I assume that the variable x_i has type A_i for every index i".

The **domain** of the context Γ is $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.

Given $x \in \mathrm{dom}(\Gamma)$ we write $\Gamma(x)$ for the type of x:

$$(\Gamma, x : A)(x) = A \qquad\qquad (\Gamma, y : A)(x) = \Gamma(x)$$

(note that a variable might occur multiple times).

Sequents

A **sequent** is a triple noted

$$\Gamma \vdash t : A$$

where

- Γ is a context,
- t is a term,
- A is a type.

Read as "under the typing assumptions for the variables in Γ, the term t has type A".

The **typing rules** are

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

with $x \in \mathrm{dom}(\Gamma)$ for (ax).

Typing rules

The **typing rules** are

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ (\to_{\mathrm{E}})$$

with $x \in \mathrm{dom}(\Gamma)$ for (ax).

Note that depending on the term only one rule applies.

Typing

We say that *a term* t *has type* A *in context* Γ when $\Gamma \vdash t : A$ is derivable.

Typing

We say that *a term* t *has type* A *in context* Γ when $\Gamma \vdash t : A$ is derivable.

We say that *a term* t *has type* A when $\vdash t : A$ is derivable.

An example of typing derivation

For instance we claim that

$$\lambda f^{A \to A}.\lambda x^A.f(fx) \qquad \text{has type} \qquad (A \to A) \to A \to A$$

which means that

$$\vdash \lambda f^{A \to A}.\lambda x^A.f(fx) : (A \to A) \to A \to A$$

is derivable.

An example of typing derivation

$$\vdash \lambda f^{A \to A}.\lambda x^{A}.f(fx) : (A \to A) \to A \to A$$

An example of typing derivation

$$\dfrac{f : A \to A \vdash \lambda x^A.f(fx) : A \to A}{\vdash \lambda f^{A\to A}.\lambda x^A.f(fx) : (A \to A) \to A \to A}\ (\to_{\mathsf{I}})$$

An example of typing derivation

$$\dfrac{\dfrac{f : A \to A, x : A \vdash f(fx) : A}{f : A \to A \vdash \lambda x^A.f(fx) : A \to A}\,(\to_I)}{\vdash \lambda f^{A\to A}.\lambda x^A.f(fx) : (A \to A) \to A \to A}\,(\to_I)$$

An example of typing derivation

$$
\dfrac{\dfrac{\dfrac{\Gamma \vdash f : A \to A \qquad\qquad \Gamma \vdash fx : A}{f : A \to A, x : A \vdash f(fx) : A}\,(\to_{\mathsf{E}})}{f : A \to A \vdash \lambda x^{A}.f(fx) : A \to A}\,(\to_{\mathsf{I}})}{\vdash \lambda f^{A \to A}.\lambda x^{A}.f(fx) : (A \to A) \to A \to A}\,(\to_{\mathsf{I}})
$$

An example of typing derivation

$$
\dfrac{\dfrac{}{\Gamma \vdash f : A \to A}\,(\mathrm{ax}) \qquad \Gamma \vdash fx : A}{\dfrac{f : A \to A, x : A \vdash f(fx) : A}{\dfrac{f : A \to A \vdash \lambda x^{A}.f(fx) : A \to A}{\vdash \lambda f^{A \to A}.\lambda x^{A}.f(fx) : (A \to A) \to A \to A}\,(\to_{\mathrm{I}})}\,(\to_{\mathrm{I}})}\,(\to_{\mathrm{E}})
$$

An example of typing derivation

$$
\dfrac{
  \dfrac{
    \dfrac{}{\Gamma \vdash f : A \to A}\,(\mathrm{ax})
    \qquad
    \dfrac{\Gamma \vdash f : A \to A \qquad \Gamma \vdash x : A}{\Gamma \vdash fx : A}\,(\to_{\mathsf{E}})
  }{
    f : A \to A, x : A \vdash f(fx) : A
  }\,(\to_{\mathsf{E}})
}{
  \dfrac{f : A \to A \vdash \lambda x^{A}.f(fx) : A \to A}{\vdash \lambda f^{A \to A}.\lambda x^{A}.f(fx) : (A \to A) \to A \to A}\,(\to_{\mathsf{I}})
}\,(\to_{\mathsf{I}})
$$

An example of typing derivation

$$\dfrac{\dfrac{}{\Gamma \vdash f : A \to A}\ (\text{ax}) \qquad \dfrac{\dfrac{}{\Gamma \vdash f : A \to A}\ (\text{ax}) \qquad \Gamma \vdash x : A}{\Gamma \vdash fx : A}\ (\to_{\text{E}})}{\dfrac{f : A \to A, x : A \vdash f(fx) : A}{\dfrac{f : A \to A \vdash \lambda x^A.f(fx) : A \to A}{\vdash \lambda f^{A \to A}.\lambda x^A.f(fx) : (A \to A) \to A \to A}\ (\to_{\text{I}})}\ (\to_{\text{I}})}\ (\to_{\text{E}})$$

An example of typing derivation

$$
\dfrac{\dfrac{}{\Gamma \vdash f : A \to A}\,(\mathsf{ax}) \qquad \dfrac{\dfrac{}{\Gamma \vdash f : A \to A}\,(\mathsf{ax}) \qquad \dfrac{}{\Gamma \vdash x : A}\,(\mathsf{ax})}{\Gamma \vdash fx : A}\,(\to_{\mathsf{E}})}{\dfrac{f : A \to A, x : A \vdash f(fx) : A}{\dfrac{f : A \to A \vdash \lambda x^A.f(fx) : A \to A}{\vdash \lambda f^{A \to A}.\lambda x^A.f(fx) : (A \to A) \to A \to A}\,(\to_{\mathsf{I}})}\,(\to_{\mathsf{I}})}\,(\to_{\mathsf{E}})
$$

Typing and α-conversion

Lemma
Two α-convertible terms have the same type.

Lemma
Two α-convertible terms have the same type.

This is the reason why we defined $\Gamma(x)$ to be the type of the *rightmost* occurrence of x:

$$\dfrac{\dfrac{\dfrac{}{x : A, y : B \vdash y : B}\,(\mathrm{ax})}{x : A \vdash \lambda y^B.y : B \to B}\,(\to_\mathrm{I})}{\vdash \lambda x^A.\lambda y^B.y : A \to B \to B}\,(\to_\mathrm{I})$$

Typing and α-conversion

Lemma
Two α-convertible terms have the same type.

This is the reason why we defined $\Gamma(x)$ to be the type of the *rightmost* occurrence of x:

$$\dfrac{\dfrac{\dfrac{}{x : A, x : B \vdash x : B}\ (\mathrm{ax})}{x : A \vdash \lambda x^B.x : B \to B}\ (\to_\mathrm{I})}{\vdash \lambda x^A.\lambda x^B.x : A \to B \to B}\ (\to_\mathrm{I})$$

Weakening

The typing system satisfies the usual structural properties.

For instance, the **weakening rule** is admissible:

Proposition
If $\Gamma \vdash t : A$ *is derivable then* $\Gamma, \Delta \vdash t : A$ *is also derivable, provided that* $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Gamma) = \emptyset$*.*

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ and $\Gamma \vdash t : A'$ are derivable then $A = A'$ (and the two proofs are the same!).

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : A'$ *are derivable then* $A = A'$ *(and the two proofs are the same!).*

Proof.
By induction on the term t.

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ and $\Gamma \vdash t : A'$ are derivable then $A = A'$ (and the two proofs are the same!).

Proof.
By induction on the term t.

We recall that the typing rules are:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\ (\to_{\mathrm{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ (\to_{\mathrm{E}})$$

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : A'$ *are derivable then* $A = A'$ *(and the two proofs are the same!).*

Proof.
By induction on the term t.

- If $t = x$ then the two derivations are necessarily

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax})$$

  and $A = A' = \Gamma(x)$

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ and $\Gamma \vdash t : A'$ are derivable then $A = A'$ (and the two proofs are the same!).

Proof.
By induction on the term t.

- If $t = \lambda x^B.u$ then the two derivations are necessarily of the form

$$\frac{\Gamma, x : B \vdash t : C}{\Gamma \vdash \lambda x^B.t : B \to C}\,(\to_\mathrm{I}) \qquad\qquad \frac{\Gamma, x : B \vdash t : C'}{\Gamma \vdash \lambda x^B.t : B \to C'}\,(\to_\mathrm{I})$$

  by induction hypothesis we have $C = C'$
  and thus $A = (B \to C) = (B \to C') = A'$.

Uniqueness of typing

A term admits at most one type:

Theorem
If $\Gamma \vdash t : A$ *and* $\Gamma \vdash t : A'$ *are derivable then* $A = A'$ *(and the two proofs are the same!).*

Proof.
By induction on the term t.

- If $t = u\,v$ then the two derivations are necessarily of the form

$$\dfrac{\Gamma \vdash t : B \to A \qquad \Gamma \vdash u : B}{\Gamma \vdash t\,u : A}\,(\to_E) \qquad\qquad \dfrac{\Gamma \vdash t : B' \to A' \qquad \Gamma \vdash u : B'}{\Gamma \vdash t\,u : A'}\,(\to_E)$$

  by induction hypothesis we have $(B \to A) = (B' \to A')$
  and thus $A = A'$. $\square$

Uniqueness of typing

The fact that we used Church style is important here!

Uniqueness of typing

The fact that we used Church style is important here!

In Curry style, this is a small variant:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\text{(ax)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}(\to_{\mathrm{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}(\to_{\mathrm{E}})$$

Uniqueness of typing

The fact that we used Church style is important here!

In Curry style, this is a small variant:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

but types are not unique anymore,

Uniqueness of typing

The fact that we used Church style is important here!

In Curry style, this is a small variant:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\,(\text{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,(\to_{\text{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\,(\to_{\text{E}})$$

but types are not unique anymore, e.g. $\lambda x.x$ has types

$$A \to A \qquad (A \to B) \to (A \to B) \qquad \text{etc.}$$

Uniqueness of typing

In fact, we have more.

We observed earlier that one rule applies on a given term:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B}\ (\to_{\mathrm{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

Theorem
Given a derivable judgment $\Gamma \vdash t : A$*, there is exactly one way to derive it.*

Typing problems

If a term admits a type, there is only one and we can compute it!

As a consequence, the three following problems are decidable: given Γ and t,

- *type checking*: determine whether $\Gamma \vdash t : A$ is derivable,
- *typability*: determine whether there exists an A such that $\Gamma \vdash t : A$ is derivable,
- *type inference*: construct an A such that $\Gamma \vdash t : A$ is derivable.

Type inference and checking

```
(** Types. *)
```

Type inference and checking

```
(** Types. *)

type ty =
  | TVar of string
  | Arr  of ty * ty
```

Type inference and checking

```
(** Types. *)

type ty =
  | TVar of string
  | Arr  of ty * ty

(** Terms. *)
```

Type inference and checking

```
(** Types. *)

type ty =
  | TVar of string
  | Arr  of ty * ty

(** Terms. *)

type term =
  | Var  of string
  | App  of term * term
  | Abs  of string * ty * term
```

Type inference and checking

```
(** Types. *)

type ty =
  | TVar of string
  | Arr  of ty * ty

(** Terms. *)

type term =
  | Var  of string
  | App  of term * term
  | Abs  of string * ty * term

(** Environments. *)
```

Type inference and checking

```
(** Types. *)

type ty =
  | TVar of string
  | Arr  of ty * ty

(** Terms. *)

type term =
  | Var  of string
  | App  of term * term
  | Abs  of string * ty * term

(** Environments. *)

type context = (string * ty) list
```

Type inference and checking

```
exception Type_error

(** Type inference. *)
```

Type inference and checking

```
exception Type_error

(** Type inference. *)

let rec infer env t =
  match t with
```

Type inference and checking

```
exception Type_error

(** Type inference. *)

let rec infer env t =
  match t with
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \text{(ax)}$$

Type inference and checking

```
exception Type_error


(** Type inference. *)

let rec infer env t =
  match t with
  | Var x -> (try List.assoc x env with Not_found -> raise Type_error)
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax})$$

Type inference and checking

```
exception Type_error

(** Type inference. *)

let rec infer env t =
  match t with
  | Var x -> (try List.assoc x env with Not_found -> raise Type_error)
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B}\ (\to_\mathrm{I})$$

Type inference and checking

```
exception Type_error

(** Type inference. *)

let rec infer env t =
  match t with
  | Var x -> (try List.assoc x env with Not_found -> raise Type_error)
  | Abs (x, a, t) -> Arr (a, infer ((x,a)::env) t)
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\,(\text{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\,(\to_{\mathrm{I}})$$

Type inference and checking

```
exception Type_error

(** Type inference. *)

let rec infer env t =
  match t with
  | Var x -> (try List.assoc x env with Not_found -> raise Type_error)
  | Abs (x, a, t) -> Arr (a, infer ((x,a)::env) t)
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\ (\to_{\mathrm{I}}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ (\to_{\mathrm{E}})$$

Type inference and checking

```
exception Type_error


(** Type inference. *)

let rec infer env t =
  match t with
  | Var x -> (try List.assoc x env with Not_found -> raise Type_error)
  | Abs (x, a, t) -> Arr (a, infer ((x,a)::env) t)
  | App (t, u) ->
     match infer env t with
     | Arr (a, b) -> check u a; b
     | _ -> raise Type_error
```

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B}\ (\to_\mathrm{I}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ (\to_\mathrm{E})$$

Type inference and checking

```
(** Type checking. *)
```

Type inference and checking

```
(** Type checking. *)

and check env t a =
  if infer env t <> a then raise Type_error
```

Type inference and checking

```
(** Type checking. *)

and check env t a =
  if infer env t <> a then raise Type_error

(** Typability. *)
```

Type inference and checking

```
(** Type checking. *)

and check env t a =
  if infer env t <> a then raise Type_error

(** Typability. *)

let typable env t =
  try  let _ = infer env t in true
  with Type_error -> false
```

Part III

The Curry-Howard correspondence

The Curry-Howard correspondence

The **Curry-Howard correspondence** is the observation that

- a type is the same as a formula in the implicative fragment of logic:

$$(A \to B) \to A \to B \qquad \text{corresponds to} \qquad (A \Rightarrow B) \Rightarrow A \Rightarrow B$$

- a typing derivation for simply typed λ-calculus is the same as a proof in NJ (implicative fragment).

The Curry-Howard correspondence

typing	logic
$\dfrac{}{\Gamma, x : A, \Gamma' \vdash x : A}\ (\mathrm{ax})$	$\dfrac{}{\Gamma, A, \Gamma' \vdash A}\ (\mathrm{ax})$
$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B}\ (\to_{\mathrm{I}})$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ (\Rightarrow_{\mathrm{I}})$
$\dfrac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$	$\dfrac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ (\Rightarrow_{\mathrm{E}})$

The Curry-Howard correspondence

The "*term-erasing procedure*" consists, starting from a typing derivation, in removing all the variables and terms (and replacing $\to$ by $\Rightarrow$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash x : A}\,(\mathrm{ax})}{f : A \to B, x : A \vdash fx : B}\,(\to_{\mathrm{E}})}{f : A \to B \vdash \lambda x^{A}.fx : A \to B}\,(\to_{\mathrm{I}})}{\vdash \lambda f^{A \to B}.\lambda x^{A}.fx : (A \to B) \to A \to B}\,(\to_{\mathrm{I}})$$

The Curry-Howard correspondence

The "*term-erasing procedure*" consists, starting from a typing derivation, in removing all the variables and terms (and replacing $\to$ by $\Rightarrow$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow B,\quad A \vdash \quad A \Rightarrow B}\,(\mathsf{ax}) \qquad \dfrac{}{A \Rightarrow B,\quad A \vdash \quad A}\,(\mathsf{ax})}{A \Rightarrow B,\quad A \vdash \quad B}\,(\Rightarrow_{\mathsf{E}})}{A \Rightarrow B \vdash \quad A \Rightarrow B}\,(\Rightarrow_{\mathsf{I}})}{\vdash \quad (A \Rightarrow B) \Rightarrow A \Rightarrow B}\,(\Rightarrow_{\mathsf{I}})$$

The "*term-erasing procedure*" consists, starting from a typing derivation, in removing all the variables and terms (and replacing $\to$ by $\Rightarrow$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow B, \quad A \vdash \quad A \Rightarrow B}\,(\mathrm{ax}) \qquad \dfrac{}{A \Rightarrow B, \quad A \vdash \quad A}\,(\mathrm{ax})}{A \Rightarrow B, \quad A \vdash \quad B}\,(\Rightarrow_{\mathrm{E}})}{A \Rightarrow B \vdash \quad A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}})}{\vdash \quad (A \Rightarrow B) \Rightarrow A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}})$$

Lemma
Given a typing derivation, its term-erasure is a valid proof in NJ.

Proof.
Immediate induction. □

The Curry-Howard correspondence

Lemma
Conversely, given a proof π of $A_1, \ldots, A_n \vdash A$ in NJ, we can construct a typing derivation of $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$, for some term t, whose term-erasure is π.

Proof.
By induction on π.

The Curry-Howard correspondence

Lemma
Conversely, given a proof π of $A_1, \ldots, A_n \vdash A$ in NJ, we can construct a typing derivation of $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$, for some term t, whose term-erasure is π.

Proof.
It the last rule is

$$\frac{}{\Gamma, A, \Gamma' \vdash A}(\mathrm{ax})$$

then we construct

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}(\mathrm{ax})$$

The Curry-Howard correspondence

Lemma
Conversely, given a proof π *of* $A_1, \ldots, A_n \vdash A$ *in NJ, we can construct a typing derivation of* $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$*, for some term* t*, whose term-erasure is* π*.*

Proof.

If the last rule is $$\dfrac{\dfrac{\pi}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}(\Rightarrow_{\mathsf{E}})$$

then, by induction hypothesis, we have $\dfrac{\vdots}{\Gamma \vdash t : A \to B}$ and $\dfrac{\vdots}{\Gamma \vdash u : A}$

and we construct $$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : A \to B} \qquad \dfrac{\vdots}{\Gamma \vdash u : A}}{\Gamma \vdash t\, u : B}(\to_{\mathsf{I}}).$$

Lemma
Conversely, given a proof π *of* $A_1, \ldots, A_n \vdash A$ *in NJ, we can construct a typing derivation of* $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$*, for some term* t*, whose term-erasure is* π*.*

Proof.

If the last rule is $\dfrac{\dfrac{\pi}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}(\Rightarrow_{\mathrm{I}})$

then by induction hypothesis we have $\dfrac{\vdots}{\Gamma, x : A \vdash t : B}$

and we construct $\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A . t : A \to B}(\to_{\mathrm{I}})$. $\square$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$
\dfrac{\dfrac{\dfrac{}{A \Rightarrow B, \quad A \vdash \quad A \Rightarrow B}\,(\mathsf{ax}) \qquad \dfrac{}{A \Rightarrow B, \quad A \vdash \quad A}\,(\mathsf{ax})}{\dfrac{A \Rightarrow B, \quad A \vdash \quad B}{A \Rightarrow B \vdash \qquad A \Rightarrow B}\,(\Rightarrow_{\mathsf{I}})}\,(\Rightarrow_{\mathsf{E}})}{\vdash \qquad (A \Rightarrow B) \Rightarrow A \Rightarrow B}\,(\Rightarrow_{\mathsf{I}})
$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\begin{prooftree}
\infer0[(ax)]{A \to B,\quad A \vdash \quad A \to B}
\infer0[(ax)]{A \to B,\quad A \vdash \quad A}
\infer2[($\to_E$)]{A \to B,\quad A \vdash \quad B}
\infer1[($\to_I$)]{A \to B \vdash \quad A \to B}
\infer1[($\to_I$)]{\vdash \quad (A \to B) \to A \to B}
\end{prooftree}$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B,\quad A \vdash \quad A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B,\quad A \vdash \quad A}\,(\mathrm{ax})}{f : A \to B,\quad A \vdash \quad B}\,(\to_{\mathrm{E}})}{f : A \to B \vdash \qquad A \to B}\,(\to_{\mathrm{I}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\mathrm{I}})$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash \quad A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash \quad A}\,(\mathrm{ax})}{f : A \to B, x : A \vdash \quad B}\,(\to_{\mathrm{E}})}{f : A \to B \vdash \qquad A \to B}\,(\to_{\mathrm{I}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\mathrm{I}})
$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash \quad A}\,(\mathrm{ax})}{f : A \to B, x : A \vdash \quad B}\,(\to_{\mathrm{E}})}{f : A \to B \vdash \qquad A \to B}\,(\to_{\mathrm{I}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\mathrm{I}})$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash x : A}\,(\mathrm{ax})}{f : A \to B, x : A \vdash \qquad B}\,(\to_{\mathrm{E}})}{f : A \to B \vdash \qquad A \to B}\,(\to_{\mathrm{I}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\mathrm{I}})
$$

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash x : A}\,(\mathrm{ax})}{\dfrac{f : A \to B, x : A \vdash fx : B}{f : A \to B \vdash \qquad A \to B}\,(\to_{\mathrm{I}})}\,(\to_{\mathrm{E}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\mathrm{I}})$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\text{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash x : A}\,(\text{ax})}{f : A \to B, x : A \vdash fx : B}\,(\to_{\text{E}})}{f : A \to B \vdash \lambda x^A.fx : A \to B}\,(\to_{\text{I}})}{\vdash \qquad (A \to B) \to A \to B}\,(\to_{\text{I}})$$

The Curry-Howard correspondence

In the previous proof we did not have any choice for the terms (up to α-conversion):

$$\dfrac{\dfrac{\dfrac{}{f : A \to B, x : A \vdash f : A \to B}\,(\mathrm{ax}) \qquad \dfrac{}{f : A \to B, x : A \vdash x : A}\,(\mathrm{ax})}{\dfrac{f : A \to B, x : A \vdash fx : B}{f : A \to B \vdash \lambda x^{A}.fx : A \to B}\,(\to_{\mathrm{I}})}\,(\to_{\mathrm{E}})}{\vdash \lambda f^{A \to B}.\lambda x^{A}.fx : (A \to B) \to A \to B}\,(\to_{\mathrm{I}})$$

The Curry-Howard correspondence

Theorem
There is a bijection between

- *typable λ-terms (up to α-conversion),*
- *typing derivations of λ-terms,*
- *proofs in the implicative fragment of NJ.*

The Curry-Howard correspondence

Theorem
There is a bijection between

- *typable λ-terms (up to α-conversion),*
- *typing derivations of λ-terms,*
- *proofs in the implicative fragment of NJ.*

In other words,

$$\text{PROGRAM} = \text{PROOF}$$

The Curry-Howard correspondence

In particular, λ-terms can be considered as *proof witnesses*:

— *you*: Hey, the formula A is true!

The Curry-Howard correspondence

In particular, λ-terms can be considered as *proof witnesses*:

— *you*: Hey, the formula A is true!

— *me*: Why should I believe you?

The Curry-Howard correspondence

In particular, λ-terms can be considered as *proof witnesses*:

— *you*: Hey, the formula A is true!

— *me*: Why should I believe you?

— *you*: Here is a term t witnessing for that.

The Curry-Howard correspondence

In particular, λ-terms can be considered as *proof witnesses*:

— *you*: Hey, the formula A is true!

— *me*: Why should I believe you?

— *you*: Here is a term t witnessing for that.

— *me*: Let me typecheck that...

The Curry-Howard correspondence

In particular, λ-terms can be considered as *proof witnesses*:

— *you*: Hey, the formula A is true!

— *me*: Why should I believe you?

— *you*: Here is a term t witnessing for that.

— *me*: Let me typecheck that...

— *me*: Ok, now I believe you!

Part IV

Other connectives

Other connectives

The correspondence extends to other logical connectives too!

The general idea is that for each connective we can introduce in λ-calculus

- constructions to create a value of this type (= introduction rules)
- constructions to use a value of this type (= elimination rules)

with appropriate reduction rules (= cut elimination).

Products

We extend the syntax of λ-terms with

Products

We extend the syntax of λ-terms with

$$t, u \quad ::= \quad \dots \mid \langle t, u \rangle \mid \pi_{\mathsf{l}}(t) \mid \pi_{\mathsf{r}}(t)$$

together with the reduction rules

Products

We extend the syntax of λ-terms with

$$t, u \quad ::= \quad \dots \mid \langle t, u \rangle \mid \pi_{\mathsf{l}}(t) \mid \pi_{\mathsf{r}}(t)$$

together with the reduction rules

$$\pi_{\mathsf{l}}\left(\langle t, u \rangle\right) \longrightarrow t \qquad\qquad \pi_{\mathsf{r}}\left(\langle t, u \rangle\right) \longrightarrow u$$

Products

```
(** Terms. *)
type term =
  | Var  of string
  | App  of term * term
  | Abs  of string * ty * term
```

Products

```
(** Terms. *)
type term =
  | Var  of string
  | App  of term * term
  | Abs  of string * ty * term
  | Pair of term * term
  | Fst of term
  | Snd of term
```

Products

We extends the syntax of types with

$$A, B \quad ::= \quad \dots \mid A \times B$$

and add the typing rules

Products

We extends the syntax of types with

$$A, B \quad ::= \quad \ldots \mid A \times B$$

and add the typing rules

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} \ (\times_{\mathsf{I}})$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A} \ (\times_{\mathsf{E}}^{\mathsf{l}}) \qquad\qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_{\mathsf{r}}(t) : B} \ (\times_{\mathsf{E}}^{\mathsf{r}})$$

Products

We extends the syntax of types with

$$A, B \quad ::= \quad \dots \mid A \times B$$

and add the typing rules

$$\frac{\Gamma \vdash \quad A \qquad \Gamma \vdash \quad B}{\Gamma \vdash \quad A \wedge B} \; (\wedge_{\mathsf{I}})$$

$$\frac{\Gamma \vdash \quad A \wedge B}{\Gamma \vdash \quad A} \; (\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \frac{\Gamma \vdash \quad A \wedge B}{\Gamma \vdash \quad B} \; (\wedge_{\mathsf{E}}^{\mathsf{r}})$$

Unit

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \langle\rangle$$

(no reduction rule), extend the syntax of types with

Unit

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \langle\rangle$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \dots \mid 1$$

and add the typing rule

Unit

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \langle\rangle$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \dots \mid 1$$

and add the typing rule

$$\frac{}{\Gamma \vdash \langle\rangle : 1}\ (1_\mathsf{I})$$

Unit

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \langle\rangle$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \dots \mid 1$$

and add the typing rule

$$\frac{}{\Gamma \vdash \quad \top}\ (\top_{\text{I}})$$

Coproducts

We now want to add coproducts types $A + B$, which corresponds to the formula $A \vee B$.

Recall that in OCaml the corresponding type is implemented with

Coproducts

We now want to add coproducts types $A + B$, which corresponds to the formula $A \vee B$.

Recall that in OCaml the corresponding type is implemented with

```
type ('a,'b) coprod =
  | Left  of 'a
  | Right of 'b
```

and a typical program using those is of the form

Coproducts

We now want to add coproducts types $A + B$, which corresponds to the formula $A \vee B$.

Recall that in OCaml the corresponding type is implemented with

```
type ('a,'b) coprod =
  | Left  of 'a
  | Right of 'b
```

and a typical program using those is of the form

```
match t with
| Left  x -> u
| Right y -> v
```

Coproducts

If we do not want to use matching, we can program once for all the function

```
let case t u v =
  match t with
  | Left  x -> u x
  | Right y -> v y
```

which is the *eliminator* for coproducts.

Coproducts

We extend the syntax of λ-terms with

Coproducts

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \iota_{\mathsf{l}}(t) \mid \iota_{\mathsf{r}}(t) \mid \mathsf{case}(t, x \mapsto u, y \mapsto v)$$

together with the reduction rules

Coproducts

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \iota_{\mathsf{l}}(t) \mid \iota_{\mathsf{r}}(t) \mid \mathrm{case}(t, x \mapsto u, y \mapsto v)$$

together with the reduction rules

$$\mathrm{case}(\iota_{\mathsf{l}}(t), x \mapsto u, y \mapsto v) \longrightarrow u[t/x]$$
$$\mathrm{case}(\iota_{\mathsf{r}}(t), x \mapsto u, y \mapsto v) \longrightarrow v[t/y]$$

Note: the reduction rules thus say that

```
match Left t with
| Left  x -> u
| Right y -> v
```

reduces to

Coproducts

Note: the reduction rules thus say that

```
match Left t with
| Left  x -> u
| Right y -> v
```

reduces to

$$u[t/x]$$

Coproducts

Note: the reduction rules thus say that

```
match Left t with
| Left  x -> u
| Right y -> v
```

reduces to

$$\texttt{u}[\texttt{t}/\texttt{x}]$$

and similarly for `Right`.

Coproducts

We extend the syntax of types with

$$A, B \quad ::= \quad \dots \mid A + B$$

and add the typing rules

Coproducts

We extend the syntax of types with

$$A, B \quad ::= \quad \dots \mid A + B$$

and add the typing rules

$$\frac{\Gamma \vdash t : A + B \qquad \Gamma, x : A \vdash u : C \qquad \Gamma, y : B \vdash v : C}{\Gamma \vdash \mathrm{case}(t, x \mapsto u, y \mapsto v) : C} \, (+_\mathrm{E})$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_\mathrm{l}(t) : A + B} \, (+^\mathrm{l}_\mathrm{I}) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_\mathrm{r}(t) : A + B} \, (+^\mathrm{r}_\mathrm{I})$$

Coproducts

We extend the syntax of types with

$$A, B \quad ::= \quad \dots \mid A + B$$

and add the typing rules

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C} \; (\vee_{\mathrm{E}})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \; (\vee_{\mathrm{I}}^{\mathrm{l}}) \qquad\qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \; (\vee_{\mathrm{I}}^{\mathrm{r}})$$

Coproducts

Note that in OCaml, the type of our function

```
let case t u v =
  match t with
  | Left  x -> u x
  | Right y -> v y
```

is

Coproducts

Note that in OCaml, the type of our function

```
let case t u v =
  match t with
  | Left  x -> u x
  | Right y -> v y
```

is

```
('a, 'b) coprod -> ('a -> 'c) -> ('b -> 'c) -> 'c
```

which can be read logically as

Coproducts

Note that in OCaml, the type of our function

```
let case t u v =
  match t with
  | Left  x -> u x
  | Right y -> v y
```

is

```
('a, 'b) coprod -> ('a -> 'c) -> ('b -> 'c) -> 'c
```

which can be read logically as

$$(A \vee B) \Rightarrow (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$$

Coproducts

There is a slight problem: what's wrong if we try to perform type inference?

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_{\mathsf{l}}(t) : A + B}\ (+^{\mathsf{l}}_{\mathsf{I}}) \qquad\qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_{\mathsf{r}}(t) : A + B}\ (+^{\mathsf{r}}_{\mathsf{I}})$$

Coproducts

There is a slight problem: what's wrong if we try to perform type inference?

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_{\mathsf{l}}(t) : A + B}\ (+^{\mathsf{l}}_{\mathsf{I}}) \qquad\qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_{\mathsf{r}}(t) : A + B}\ (+^{\mathsf{r}}_{\mathsf{I}})$$

For instance, what is the type of $\iota_{\mathsf{l}}(\lambda x^A.x)$?

Coproducts

There is a slight problem: what's wrong if we try to perform type inference?

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_\mathsf{l}(t) : A + B}\ (+^\mathsf{l}_\mathsf{I}) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_\mathsf{r}(t) : A + B}\ (+^\mathsf{r}_\mathsf{I})$$

For instance, what is the type of $\iota_\mathsf{l}(\lambda x^A.x)$?

It is like Church vs Curry, we need more typing information:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_\mathsf{l}^B(t) : A + B}\ (+^\mathsf{l}_\mathsf{I}) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_\mathsf{r}^A(t) : A + B}\ (+^\mathsf{r}_\mathsf{I})$$

Coproducts

Note that a term

$$\mathsf{case}(t, x \mapsto u, y \mapsto v)$$

should be considered up to α-conversion (we can rename x and y), which means extra care when implementing substitution.

Coproducts

Note that a term

$$\mathrm{case}(t, x \mapsto u, y \mapsto v)$$

should be considered up to α-conversion (we can rename x and y), which means extra care when implementing substitution.

Instead, it is often easier to implement the variant with actual functions

$$\mathrm{case}(t, u, v)$$

Coproducts

Note that a term

$$\mathsf{case}(t, x \mapsto u, y \mapsto v)$$

should be considered up to α-conversion (we can rename x and y), which means extra care when implementing substitution.

Instead, it is often easier to implement the variant with actual functions

$$\mathsf{case}(t, u, v)$$

which is typed as

$$\frac{\Gamma \vdash t : A + B \qquad \Gamma \vdash u : A \to C \qquad \Gamma \vdash v : B \to C}{\Gamma \vdash \mathsf{case}(t, u, v) : C} \ (+_{\mathrm{E}})$$

Coproducts

Note that a term

$$\mathrm{case}(t, x \mapsto u, y \mapsto v)$$

should be considered up to α-conversion (we can rename x and y), which means extra care when implementing substitution.

Instead, it is often easier to implement the variant with actual functions

$$\mathrm{case}(t, u, v)$$

which is typed as

$$\frac{\Gamma \vdash t : A + B \qquad \Gamma \vdash u : A \to C \qquad \Gamma \vdash v : B \to C}{\Gamma \vdash \mathrm{case}(t, u, v) : C} \, (+_{\mathrm{E}})$$

instead of

$$\frac{\Gamma \vdash t : A + B \qquad \Gamma, x : A \vdash u : C \qquad \Gamma, y : B \vdash v : C}{\Gamma \vdash \mathrm{case}(t, x \mapsto u, y \mapsto v) : C} \, (+_{\mathrm{E}})$$

Coproducts

In the previous slide we agreed that, instead of writing

$$\mathrm{case}(t, x \mapsto u, y \mapsto v)$$

we should write

$$\mathrm{case}(t, \lambda x.u, \lambda y.v)$$

There is however a problem:

Coproducts

In the previous slide we agreed that, instead of writing

$$\mathsf{case}(t, x \mapsto u, y \mapsto v)$$

we should write

$$\mathsf{case}(t, \lambda x.u, \lambda y.v)$$

There is however a problem: we do not have a type for the abstracted variables x and y!

Coproducts

In the previous slide we agreed that, instead of writing

$$\mathsf{case}(t, x \mapsto u, y \mapsto v)$$

we should write

$$\mathsf{case}(t, \lambda x.u, \lambda y.v)$$

There is however a problem: we do not have a type for the abstracted variables x and y!

However, it is fine to allow λ-terms without types (à la Curry) here
because we can guess them when typing:

$$\dfrac{\Gamma \vdash t : A + B \qquad \dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash u : C}}{\Gamma \vdash \lambda x.u : A \to C}\,(\to_{\mathrm{I}}) \qquad \dfrac{\dfrac{\vdots}{\Gamma, y : A \vdash v : C}}{\Gamma \vdash \lambda y.v : B \to C}\,(\to_{\mathrm{I}})}{\Gamma \vdash \mathsf{case}(t, \lambda x.u, \lambda y.v) : C}\,(+_{\mathrm{E}})$$

Coproducts

This is easily implemented using two (mutually recursive) functions:

- **inference**
  e.g. we can infer that $\lambda x^A.t$ has type $A \to B$
- **checking**
  e.g. we can check that $\lambda x.t$ has type $A \to B$

Empty type

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \mathsf{case}(t)$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \dots \mid 0$$

and add the typing rule

Empty type

We extend the syntax of λ-terms with

$$t \quad ::= \quad \dots \mid \mathsf{case}(t)$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \dots \mid 0$$

and add the typing rule

$$\frac{\Gamma \vdash t : 0}{\Gamma \vdash \mathsf{case}(t) : A}\ (0_{\mathsf{E}})$$

Empty type

We extend the syntax of λ-terms with

$$t \quad ::= \quad \ldots \mid \mathsf{case}(t)$$

(no reduction rule), extend the syntax of types with

$$A \quad ::= \quad \ldots \mid 0$$

and add the typing rule

$$\frac{\Gamma \vdash \quad \bot}{\Gamma \vdash \qquad\qquad A}\,(\bot_{\mathrm{E}})$$

All together

If we add them all together, we want more reduction rules:

$$\mathrm{case}^{A\to B}(t)\,u \longrightarrow \mathrm{case}^{B}(t)$$

$$\pi_{\mathsf{l}}(\mathrm{case}^{A\times B}(t)) \longrightarrow \mathrm{case}^{A}(t)$$

$$\pi_{\mathsf{r}}(\mathrm{case}^{A\times B}(t)) \longrightarrow \mathrm{case}^{B}(t)$$

$$\mathrm{case}(\mathrm{case}^{A+B}(t), x \mapsto u, y \mapsto v) \longrightarrow \mathrm{case}^{C}(t)$$

$$\mathrm{case}^{A}(\mathrm{case}^{0}(t)) \longrightarrow \mathrm{case}^{A}(t)$$

$$\mathrm{case}(t, x \mapsto u, y \mapsto v)\,w \longrightarrow \mathrm{case}(t, x \mapsto uw, y \mapsto vw)$$

$$\pi_{\mathsf{l}}(\mathrm{case}(t, x \mapsto u, y \mapsto v)) \longrightarrow \mathrm{case}(t, x \mapsto \pi_{\mathsf{l}}(u), y \mapsto \pi_{\mathsf{l}}(v))$$

$$\pi_{\mathsf{r}}(\mathrm{case}(t, x \mapsto u, y \mapsto v)) \longrightarrow \mathrm{case}(t, x \mapsto \pi_{\mathsf{r}}(u), y \mapsto \pi_{\mathsf{r}}(v))$$

$$\mathrm{case}^{C}(\mathrm{case}(t, x \mapsto u, y \mapsto v)) \longrightarrow \mathrm{case}(t, x \mapsto \mathrm{case}^{C}(u), y \mapsto \mathrm{case}^{C}(v))$$

$$\mathrm{case}(\mathrm{case}(t, x \mapsto u, y \mapsto v), x' \mapsto u', y' \mapsto v') \longrightarrow \mathrm{case}(t, x \mapsto \mathrm{case}(u, x' \mapsto u', y' \mapsto v'), y \mapsto$$

Natural numbers

In OCaml, natural numbers can be defined as

```
type nat =
  | Zero
  | Succ of nat
```

so that factorial can be implemented with

Natural numbers

In OCaml, natural numbers can be defined as

```
type nat =
  | Zero
  | Succ of nat
```

so that factorial can be implemented with

```
let rec fact n =
  match n with
  | Zero   -> Succ Zero
  | Succ n -> mult (Succ n) (fact n)
```

Natural numbers

The “recurrence principle” / eliminator can then be defined as

Natural numbers

The “recurrence principle” / eliminator can then be defined as

```
let rec recursor n z s =
  match n with
  | Zero   -> z
  | Succ n -> s n (recursor n z s)
```

of type

```
val recursor : nat -> 'a -> (nat -> 'a -> 'a) -> 'a = <fun>
```

Natural numbers

The “recurrence principle” / eliminator can then be defined as

```
let rec recursor n z s =
  match n with
  | Zero   -> z
  | Succ n -> s n (recursor n z s)
```

of type

```
val recursor : nat -> 'a -> (nat -> 'a -> 'a) -> 'a = <fun>
```

so that factorial can be programmed as

Natural numbers

The “recurrence principle” / eliminator can then be defined as

```
let rec recursor n z s =
  match n with
  | Zero   -> z
  | Succ n -> s n (recursor n z s)
```

of type

```
val recursor : nat -> 'a -> (nat -> 'a -> 'a) -> 'a = <fun>
```

so that factorial can be programmed as

```
let fact n =
  recursor n (Succ Zero) (fun n r -> mult (Succ n) r)
```

Natural numbers

We extend the syntax of λ-terms with

$$t \quad ::= \quad \ldots \mid$$

Natural numbers

We extend the syntax of λ-terms with

$$t \quad ::= \quad \ldots \mid \mathsf{Z} \mid \mathsf{S}(t) \mid \mathsf{rec}(t, u, xy \mapsto v)$$

Natural numbers

We extend the syntax of λ-terms with

$$t \quad ::= \quad \ldots \mid \mathsf{Z} \mid \mathsf{S}(t) \mid \mathsf{rec}(t, u, xy \mapsto v)$$

and add the reduction rules

Natural numbers

We extend the syntax of λ-terms with

$$t \quad ::= \quad \ldots \mid \mathsf{Z} \mid \mathsf{S}(t) \mid \mathsf{rec}(t, u, xy \mapsto v)$$

and add the reduction rules

$$\begin{aligned} \mathsf{rec}(\mathsf{Z}, z, xy \mapsto s) &\longrightarrow z \\ \mathsf{rec}(\mathsf{S}(n), z, xy \mapsto s) &\longrightarrow s[n/x, \mathsf{rec}(n, z, xy \mapsto s)/y] \end{aligned}$$

Natural numbers

We extend the syntax of types with

$$A, B \quad ::= \quad \ldots \mid \mathsf{Nat}$$

Natural numbers

We extend the syntax of types with

$$A, B \quad ::= \quad \ldots \mid \mathsf{Nat}$$

and add the typing rules

Natural numbers

We extend the syntax of types with

$$A, B \quad ::= \quad \ldots \mid \mathsf{Nat}$$

and add the typing rules

$$\frac{}{\Gamma \vdash \mathsf{Z} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{S}(t) : \mathsf{Nat}}$$

$$\frac{\Gamma \vdash n : \mathsf{Nat} \qquad \Gamma \vdash z : A \qquad \Gamma, x : \mathsf{Nat}, y : A \vdash s : A}{\Gamma \vdash \mathsf{rec}(n, z, xy \mapsto s) : A}$$

Natural numbers

We extend the syntax of types with

$$A, B \quad ::= \quad \ldots \mid \mathsf{Nat}$$

and add the typing rules

$$\frac{}{\Gamma \vdash \qquad \mathsf{Nat}} \qquad \frac{\Gamma \vdash \qquad \mathsf{Nat}}{\Gamma \vdash \qquad \mathsf{Nat}}$$

$$\frac{\Gamma \vdash \qquad \mathsf{Nat} \qquad \Gamma \vdash \qquad A \qquad \Gamma, \qquad \mathsf{Nat}, \qquad A \vdash \qquad A}{\Gamma \vdash \qquad A}$$

Part V

Dynamics of Curry-Howard

Cut elimination

Remember that a **cut** in a proof is an elimination rule whose principal premise is an introduction rule.

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash A}\,(\wedge^{\mathrm{l}}_{\mathrm{E}}) \qquad\qquad \dfrac{\dfrac{\dfrac{\pi}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}\,(\Rightarrow_{\mathrm{E}})$$

Such a proof is intuitively doing “useless work” and we have seen that we could gradually remove all the cuts from a proof.

Cut elimination: conjunction

For instance, the cuts related to conjunction can be eliminated with

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_\mathsf{I})}{\Gamma \vdash A}\,(\wedge^\mathsf{l}_\mathsf{E}) \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_\mathsf{I})}{\Gamma \vdash B}\,(\wedge^\mathsf{r}_\mathsf{E}) \quad \rightsquigarrow$$

Cut elimination: conjunction

For instance, the cuts related to conjunction can be eliminated with

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash A}\,(\wedge_{\mathrm{E}}^{\mathrm{l}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash A}$$

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash B}\,(\wedge_{\mathrm{E}}^{\mathrm{r}}) \quad \rightsquigarrow \quad \dfrac{\pi'}{\Gamma \vdash B}$$

Cut elimination: conjunction

For instance, the cuts related to conjunction can be eliminated with

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash A}\,(\wedge^{\mathrm{l}}_{\mathrm{E}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash A}$$

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash B}\,(\wedge^{\mathrm{r}}_{\mathrm{E}}) \quad \rightsquigarrow \quad \dfrac{\pi'}{\Gamma \vdash B}$$

What it the computational contents of this transformation?

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A} \qquad \dfrac{\pi'}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,(\wedge_\mathrm{I})}{\Gamma \vdash A}\,(\wedge_\mathrm{E}^\mathrm{l}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash A}$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash \quad B}}{\Gamma \vdash \qquad A \wedge B}\,(\wedge_{\mathsf{I}})}{\Gamma \vdash \qquad A}\,(\wedge_{\mathsf{E}}^{\mathsf{l}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash \quad A}$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$
\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash u : B}}{\Gamma \vdash \quad A \wedge B}\,(\wedge_\mathrm{I})}{\Gamma \vdash \quad A}\,(\wedge_\mathrm{E}^\mathrm{l}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash \quad A}
$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash u : B}}{\Gamma \vdash \langle t, u \rangle : A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash \qquad\qquad A}\,(\wedge_{\mathrm{E}}^{\mathrm{l}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A}$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash u : B}}{\Gamma \vdash \langle t, u \rangle : A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash \pi_{\mathrm{l}}\, \langle t, u \rangle : A}\,(\wedge^{\mathrm{l}}_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A}$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash u : B}}{\Gamma \vdash \langle t, u \rangle : A \wedge B}\,(\wedge_{\mathrm{I}})}{\Gamma \vdash \pi_{\mathrm{l}}\,\langle t, u \rangle : A}\,(\wedge_{\mathrm{E}}^{\mathrm{l}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash t : A}$$

Cut elimination: conjunction

One of the cut-elimination rules is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A} \qquad \dfrac{\pi'}{\Gamma \vdash u : B}}{\Gamma \vdash \langle t, u \rangle : A \wedge B}(\wedge_{\mathsf{I}})}{\Gamma \vdash \pi_{\mathsf{l}} \langle t, u \rangle : A}(\wedge_{\mathsf{E}}^{\mathsf{l}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash t : A}$$

In other words, it transforms a subterm

$$\pi_{\mathsf{l}} \langle t, u \rangle \quad \longrightarrow_{\beta} \quad t$$

which is the reduction rule!

Cut elimination: implication

The cut elimination rule for $\Rightarrow$ is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_\mathrm{I}) \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}\,(\Rightarrow_\mathrm{E}) \qquad \rightsquigarrow$$

Cut elimination: implication

The cut elimination rule for $\Rightarrow$ is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash B}$$

where $\pi[\pi'/A]$ is π where we have replaced all axioms on A

$$\dfrac{}{\Gamma, A, \Gamma' \vdash A}\,(\mathrm{ax}) \qquad \text{by} \qquad \dfrac{w(\pi')}{\Gamma, \Gamma' \vdash A}$$

where $w(\pi')$ is an appropriate weakening of π.

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\,(\text{ax})}{\Gamma \vdash A \Rightarrow A}\,(\Rightarrow_{\text{I}}) \qquad \dfrac{\pi}{\Gamma \vdash A}}{\Gamma \vdash A}\,(\Rightarrow_{\text{E}}) \qquad \rightsquigarrow$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\,(\mathrm{ax})}{\Gamma \vdash A \Rightarrow A}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi}{\Gamma \vdash A}}{\Gamma \vdash A}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash A}$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\overline{\Gamma, x : A \vdash A}\ (\mathrm{ax})}{\Gamma \vdash A \Rightarrow A}\ (\Rightarrow_\mathrm{I}) \qquad \dfrac{\pi}{\Gamma \vdash A}}{\Gamma \vdash A}\ (\Rightarrow_\mathrm{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash A}$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\,(\text{ax})}{\Gamma \vdash A \Rightarrow A}\,(\Rightarrow_{\text{I}}) \qquad \dfrac{\pi}{\Gamma \vdash A}}{\Gamma \vdash A}\,(\Rightarrow_{\text{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash A}$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\,(\text{ax})}{\Gamma \vdash \lambda x^A.x : A \Rightarrow A}\,(\Rightarrow_\text{I}) \qquad \dfrac{\pi}{\Gamma \vdash \quad A}}{\Gamma \vdash \qquad\qquad A}\,(\Rightarrow_\text{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A}$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\,(\text{ax})}{\Gamma \vdash \lambda x^A.x : A \Rightarrow A}\,(\Rightarrow_{\text{I}}) \qquad \dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash \qquad A}\,(\Rightarrow_{\text{E}}) \qquad\rightsquigarrow\qquad \dfrac{\pi}{\Gamma \vdash \quad A}$$

Cut elimination: implication

For instance, we can eliminate the cut

$$
\dfrac{\dfrac{\overline{\Gamma, x : A \vdash x : A}\ (\text{ax})}{\Gamma \vdash \lambda x^A.x : A \Rightarrow A}\ (\Rightarrow_\text{I}) \qquad \dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash (\lambda x^A.x)t : A}\ (\Rightarrow_\text{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A}
$$

Cut elimination: implication

For instance, we can eliminate the cut

$$
\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\ (\mathrm{ax})}{\Gamma \vdash \lambda x^A.x : A \Rightarrow A}\ (\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash (\lambda x^A.x)t : A}\ (\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A}
$$

Cut elimination: implication

For instance, we can eliminate the cut

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma \vdash \lambda x^A.x : A \Rightarrow A}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash (\lambda x^A.x)t : A}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A}$$

In other words,

$$(\lambda x^A.x)t \qquad \longrightarrow_\beta \qquad t$$

which is the β-reduction rule!

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash B}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash B}$$

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash A \Rightarrow B}(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash A}}{\Gamma \vdash B}(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash B}$$

Cut elimination: implication

More generally, we have

$$
\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\,(\Rightarrow_\mathrm{I}) \qquad \dfrac{\pi'}{\Gamma \vdash \quad A}}{\Gamma \vdash \qquad\qquad B}\,(\Rightarrow_\mathrm{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash \qquad\qquad B}
$$

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A . t : A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash u : A}}{\Gamma \vdash \qquad\qquad B}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash \qquad B}$$

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A . t : A \Rightarrow B}\,(\Rightarrow_\mathrm{I}) \qquad \dfrac{\pi'}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x^A . t) u : B}\,(\Rightarrow_\mathrm{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash \qquad\qquad B}$$

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\ (\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x^A.t)u : B}\ (\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash t[u/x] : B}$$

Cut elimination: implication

More generally, we have

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x^A . t : A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \dfrac{\pi'}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x^A . t) u : B}\,(\Rightarrow_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi[\pi'/A]}{\Gamma \vdash t[u/x] : B}$$

In other words,

$$(\lambda x^A . t) u \qquad \longrightarrow_\beta \qquad t[u/x]$$

which is the β-reduction rule!

Cut elimination and reduction

Suppose given a term t of type A in a context Γ, its typing derivation

$$\frac{\pi}{\Gamma \vdash t : A}$$

can be seen as a proof in NJ. We have shown that

Cut elimination and reduction

Suppose given a term t of type A in a context Γ, its typing derivation

$$\frac{\pi}{\Gamma \vdash t : A}$$

can be seen as a proof in NJ. We have shown that

Theorem
The cut elimination steps of π are in correspondence with the β-reduction steps of t.

This means that for every β-reduction $t \longrightarrow_\beta t'$ there is a derivation π' of $\Gamma \vdash t' : A$

$$\frac{\pi}{\Gamma \vdash t : A} \quad \rightsquigarrow \quad \frac{\pi'}{\Gamma \vdash t' : A}$$

which is obtained by a cut elimination step from π, and conversely every cut-elimination step from π is of this form.

In particular, we have shown the **subject reduction** property:

typing is compatible with β-reduction.

Theorem

If $\Gamma \vdash t : A$ is derivable and $t \longrightarrow_\beta t'$ then $\Gamma \vdash t' : A$ is also derivable.

Subject reduction

In particular, we have shown the **subject reduction** property:
typing is compatible with β-reduction.

Theorem
If $\Gamma \vdash t : A$ is derivable and $t \longrightarrow_\beta t'$ then $\Gamma \vdash t' : A$ is also derivable.

For instance,

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma \vdash \lambda x^A.x : A \to A}\,(\to_\mathrm{I}) \qquad \dfrac{\pi}{\Gamma \vdash t : B}}{\vdash (\lambda x^A.x)t : B}\,(\to_\mathrm{E}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : B}$$

The Curry-Howard correspondence

We can add a third level to the correspondence:

Theorem
There is a bijection between

1. *types and formulas,*
2. *λ-terms of type A and proofs of A in NJ,*

We can add a third level to the correspondence:

Theorem
There is a bijection between

1. *types and formulas,*
2. λ*-terms of type* A *and proofs of* A *in NJ,*
3. *reduction steps and cut elimination steps.*

We can add a third level to the correspondence:

Theorem
There is a bijection between

1. *types and formulas,*
2. λ*-terms of type* A *and proofs of* A *in NJ,*
3. *reduction steps and cut elimination steps.*

Using a function in programming is the same as using a lemma in mathematics!

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A \Rightarrow B}}{\Gamma, A \vdash A \Rightarrow B}\,(\mathsf{wk}) \qquad \dfrac{}{\Gamma, A \vdash A}\,(\mathsf{ax})}{\Gamma, A \vdash B}\,(\Rightarrow_{\mathsf{E}})}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathsf{I}}) \qquad \rightsquigarrow
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A \Rightarrow B}}{\Gamma, A \vdash A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, A \vdash A}\,(\mathrm{ax})}{\Gamma, A \vdash B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash A \Rightarrow B}
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A \Rightarrow B}}{\Gamma, x : A \vdash A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash A}\,(\mathrm{ax})}{\Gamma, x : A \vdash B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash A \Rightarrow B}$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash \quad A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash \quad A}\,(\mathrm{ax})}{\Gamma, x : A \vdash \quad B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \quad A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}})
\qquad \rightsquigarrow \qquad
\dfrac{\pi}{\Gamma \vdash \quad A \Rightarrow B}
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash \quad A}\,(\mathrm{ax})}{\Gamma, x : A \vdash \qquad B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \qquad\qquad A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}})
\qquad \rightsquigarrow \qquad
\dfrac{\pi}{\Gamma \vdash \quad A \Rightarrow B}
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma, x : A \vdash \qquad B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \qquad\qquad A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \qquad A \Rightarrow B}$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma, x : A \vdash tx : B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \qquad\qquad A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}})
\quad \rightsquigarrow \quad
\dfrac{\pi}{\Gamma \vdash \quad A \Rightarrow B}
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma, x : A \vdash tx : B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \lambda x^A.tx : A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \qquad A \Rightarrow B}
$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma, x : A \vdash tx : B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \lambda x^A . tx : A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}$$

A variant of cuts

A cut is an elimination of an introduction of some connective.

What if we try to do the converse (introduction of an elimination)?

For implication,

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}}{\Gamma, x : A \vdash t : A \Rightarrow B}\,(\mathrm{wk}) \qquad \dfrac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{ax})}{\Gamma, x : A \vdash tx : B}\,(\Rightarrow_{\mathrm{E}})}{\Gamma \vdash \lambda x^A.tx : A \Rightarrow B}\,(\Rightarrow_{\mathrm{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A \Rightarrow B}$$

In other words, we recover η-**reduction**:

$$\lambda x^A.tx \qquad \longrightarrow_\eta \qquad t$$

A variant of cuts

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A}\,(\wedge_E^l) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash A \wedge B}}{\Gamma \vdash B}\,(\wedge_E^r)}{\Gamma \vdash A \wedge B}\,(\wedge_I) \quad \rightsquigarrow \quad \dfrac{\pi}{\Gamma \vdash A \wedge B}$$

A variant of cuts

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \qquad A}(\wedge_E^l) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \qquad B}(\wedge_E^r)}{\Gamma \vdash \qquad A \wedge B}(\wedge_I) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A \wedge B}$$

A variant of cuts

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A}\,(\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \qquad\qquad B}\,(\wedge_{\mathsf{E}}^{\mathsf{r}})}{\Gamma \vdash \qquad\qquad\qquad\qquad A \wedge B}\,(\wedge_{\mathsf{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \qquad A \wedge B}$$

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A}\,(\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{r}}(t) : B}\,(\wedge_{\mathsf{E}}^{\mathsf{r}})}{\Gamma \vdash \qquad\qquad A \wedge B}\,(\wedge_{\mathsf{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \qquad A \wedge B}$$

A variant of cuts

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A}\,(\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{r}}(t) : B}\,(\wedge_{\mathsf{E}}^{\mathsf{r}})}{\Gamma \vdash \langle \pi_{\mathsf{l}}(t), \pi_{\mathsf{r}}(t) \rangle : A \wedge B}\,(\wedge_{\mathsf{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash \quad A \wedge B}$$

A variant of cuts

This also works for other connectives:

$$
\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A}\,(\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{r}}(t) : B}\,(\wedge_{\mathsf{E}}^{\mathsf{r}})}{\Gamma \vdash \langle \pi_{\mathsf{l}}(t), \pi_{\mathsf{r}}(t) \rangle : A \wedge B}\,(\wedge_{\mathsf{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A \wedge B}
$$

A variant of cuts

This also works for other connectives:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{l}}(t) : A}(\wedge_{\mathsf{E}}^{\mathsf{l}}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A \wedge B}}{\Gamma \vdash \pi_{\mathsf{r}}(t) : B}(\wedge_{\mathsf{E}}^{\mathsf{r}})}{\Gamma \vdash \langle \pi_{\mathsf{l}}(t), \pi_{\mathsf{r}}(t) \rangle : A \wedge B}(\wedge_{\mathsf{I}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A \wedge B}$$

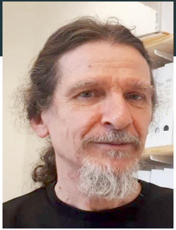
In other words, the η-reduction rule for products is

$$\langle \pi_{\mathsf{l}}(t), \pi_{\mathsf{r}}(t) \rangle \qquad \longrightarrow_{\eta} \qquad t$$

Part VI

Classical logic

Classical logic



For a long time, people thought that this correspondence could not be extended to classical logic.

It turns out that it actually does (Parigot's $\lambda\mu$-calculus): classical logic is **constructive**!

This gives rise to strange languages, relying heavily on a variant of exceptions.

Classical logic

We have seen that classical logic could be obtained from intuitionistic one by adding the principle of elimination of double negation:

$$\neg\neg A \Rightarrow A$$

Classical logic

We have seen that classical logic could be obtained from intuitionistic one by adding the principle of elimination of double negation:

$$\neg\neg A \Rightarrow A$$

This can be equivalently implemented by adding the rule

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ (\neg\neg_{\mathrm{E}})$$

Classical logic

We have seen that classical logic could be obtained from intuitionistic one by adding the principle of elimination of double negation:

$$\neg\neg A \Rightarrow A$$

This can be equivalently implemented by adding the rule

$$\frac{\Gamma \vdash t : \neg\neg A}{\Gamma \vdash \mathcal{C}(t) : A}\ (\neg\neg_{\mathrm{E}})$$

This suggests that we should add a new construction $\mathcal{C}$.

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

Proof.
Suppose that we have a proof π of $\Gamma \vdash A$, then we have

$$\Gamma \vdash \neg\neg A$$

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

Proof.
Suppose that we have a proof π of $\Gamma \vdash A$, then we have

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash \neg\neg A}\ (\neg_{\mathrm{I}})$$

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

Proof.
Suppose that we have a proof π of $\Gamma \vdash A$, then we have

$$\dfrac{\dfrac{\Gamma, \neg A \vdash \neg A \qquad\qquad \Gamma, \neg A \vdash A}{\Gamma, \neg A \vdash \bot}\,(\neg_\mathrm{E})}{\Gamma \vdash \neg\neg A}\,(\neg_\mathrm{I})$$

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

Proof.
Suppose that we have a proof π of $\Gamma \vdash A$, then we have

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \neg A \vdash \neg A}\,(\mathrm{ax}) \qquad \Gamma, \neg A \vdash A}{\Gamma, \neg A \vdash \bot}\,(\neg_{\mathrm{E}})}{\Gamma \vdash \neg\neg A}\,(\neg_{\mathrm{I}})$$

For the introduction rule, recall that we have shown:

Lemma
The following rule is admissible

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$

Proof.
Suppose that we have a proof π of $\Gamma \vdash A$, then we have

$$\dfrac{\dfrac{}{\Gamma, \neg A \vdash \neg A}\,(\mathrm{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash A}}{\Gamma, \neg A \vdash A}\,(\mathrm{wk})}{\dfrac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash \neg\neg A}\,(\neg_{\mathrm{I}})}\,(\neg_{\mathrm{E}})$$

Classical logic

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, A \Rightarrow \bot \vdash A \Rightarrow \bot}\text{(ax)} \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash A}}{\Gamma, A \Rightarrow \bot \vdash A}\text{(wk)}}{\dfrac{\Gamma, A \Rightarrow \bot \vdash \bot}{\Gamma \vdash (A \Rightarrow \bot) \Rightarrow \bot}(\Rightarrow_{\mathrm{I}})}(\Rightarrow_{\mathrm{E}})$$

Classical logic

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, k : A \Rightarrow \bot \vdash \quad A \Rightarrow \bot}\,(\text{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma, k : A \Rightarrow \bot \vdash \quad A}\,(\text{wk})}{\dfrac{\Gamma, k : A \Rightarrow \bot \vdash \quad \bot}{\Gamma \vdash \quad (A \Rightarrow \bot) \Rightarrow \bot}\,(\Rightarrow_{\mathrm{I}})}\,(\Rightarrow_{\mathrm{E}})$$

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, k : A \Rightarrow \bot \vdash k : A \Rightarrow \bot}\,(\text{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma, k : A \Rightarrow \bot \vdash \quad A}\,(\text{wk})}{\dfrac{\Gamma, k : A \Rightarrow \bot \vdash \qquad \bot}{\Gamma \vdash \qquad (A \Rightarrow \bot) \Rightarrow \bot}\,(\Rightarrow_{\text{I}})}\,(\Rightarrow_{\text{E}})$$

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, k : A \Rightarrow \bot \vdash k : A \Rightarrow \bot}\,(\text{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma, k : A \Rightarrow \bot \vdash t : A}\,(\text{wk})}{\dfrac{\Gamma, k : A \Rightarrow \bot \vdash \qquad \bot}{\Gamma \vdash \qquad (A \Rightarrow \bot) \Rightarrow \bot}\,(\Rightarrow_{\text{I}})}\,(\Rightarrow_{\text{E}})$$

Classical logic

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, k : A \Rightarrow \bot \vdash k : A \Rightarrow \bot}\,(\mathrm{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma, k : A \Rightarrow \bot \vdash t : A}\,(\mathrm{wk})}{\dfrac{\Gamma, k : A \Rightarrow \bot \vdash k\, t : \bot}{\Gamma \vdash \qquad (A \Rightarrow \bot) \Rightarrow \bot}\,(\Rightarrow_{\mathrm{I}})}\,(\Rightarrow_{\mathrm{E}})$$

Classical logic

Remembering that

$$\neg A \quad = \quad A \Rightarrow \bot$$

we already have an introduction rule for double negation:

$$\dfrac{\dfrac{}{\Gamma, k : A \Rightarrow \bot \vdash k : A \Rightarrow \bot}\;(\text{ax}) \qquad \dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma, k : A \Rightarrow \bot \vdash t : A}\;(\text{wk})}{\dfrac{\Gamma, k : A \Rightarrow \bot \vdash k\,t : \bot}{\Gamma \vdash \lambda k^{A \Rightarrow \bot}.k\,t : (A \Rightarrow \bot) \Rightarrow \bot}\;(\Rightarrow_\text{I})}\;(\Rightarrow_\text{E})$$

Classical logic: reduction

The cut elimination procedure should give

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash A}}{\Gamma \vdash \neg\neg A}\,(\neg\neg_{\mathsf{I}})}{\Gamma \vdash A}\,(\neg\neg_{\mathsf{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash A}$$

Classical logic: reduction

The cut elimination procedure should give

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash \neg\neg A}\,(\neg\neg_{\mathrm{I}})}{\Gamma \vdash A}\,(\neg\neg_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A}$$

Classical logic: reduction

The cut elimination procedure should give

$$
\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash \lambda k^{\neg A}.k\,t : \neg\neg A}\,(\neg\neg_{\mathsf{I}})}{\Gamma \vdash \qquad\qquad\qquad A}\,(\neg\neg_{\mathsf{E}})
\qquad \rightsquigarrow \qquad
\dfrac{\pi}{\Gamma \vdash t : A}
$$

The cut elimination procedure should give

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash \lambda k^{\neg A}.k\,t : \neg\neg A}\ (\neg\neg_{\mathrm{I}})}{\Gamma \vdash \mathcal{C}(\lambda k^{\neg A}.k\,t) : A}\ (\neg\neg_{\mathrm{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A}$$

Classical logic: reduction

The cut elimination procedure should give

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash t : A}}{\Gamma \vdash \lambda k^{\neg A}.k\,t : \neg\neg A}\,(\neg\neg_{\mathsf{I}})}{\Gamma \vdash \mathcal{C}(\lambda k^{\neg A}.k\,t) : A}\,(\neg\neg_{\mathsf{E}}) \qquad \rightsquigarrow \qquad \dfrac{\pi}{\Gamma \vdash t : A}$$

In other words,

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \qquad \longrightarrow_{\beta} \qquad t$$

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \qquad \longrightarrow_\beta \qquad t$$

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\, t) \qquad \longrightarrow_{\beta} \qquad t$$

When we apply this function k to some argument t the function $\mathcal{C}$ will discard the computation and return the argument t,

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \qquad \longrightarrow_\beta \qquad t$$

When we apply this function k to some argument t the function $\mathcal{C}$ will discard the computation and return the argument t, which only makes sense when $k \notin \mathsf{FV}(t)$!

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\, t) \qquad \longrightarrow_\beta \qquad t$$

When we apply this function k to some argument t the function $\mathcal{C}$ will discard the computation and return the argument t, which only makes sense when $k \notin \mathsf{FV}(t)$!

Generally, reduction looks like this:

$$\mathcal{C}(\lambda k^{\neg A}.u) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_1) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_2) \longrightarrow_\beta \dots \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.k\, t) \longrightarrow_\beta t$$

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \qquad \longrightarrow_\beta \qquad t$$

When we apply this function k to some argument t the function $\mathcal{C}$ will discard the computation and return the argument t, which only makes sense when $k \notin \mathsf{FV}(t)$!

Generally, reduction looks like this:

$$\mathcal{C}(\lambda k^{\neg A}.u) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_1) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_2) \longrightarrow_\beta \ldots \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.k\,t) \longrightarrow_\beta t$$

We can read

$$\mathcal{C}(\ldots) = \texttt{try ... catch x -> x} \qquad\qquad k = \texttt{raise}$$

Classical logic: reduction

The reduction rule is

$$\mathcal{C}(\lambda k^{\neg A}.k\, t) \qquad \longrightarrow_\beta \qquad t$$

When we apply this function k to some argument t the function $\mathcal{C}$ will discard the computation and return the argument t, which only makes sense when $k \notin \mathsf{FV}(t)$!

Generally, reduction looks like this:

$$\mathcal{C}(\lambda k^{\neg A}.u) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_1) \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.u_2) \longrightarrow_\beta \ldots \longrightarrow_\beta \mathcal{C}(\lambda k^{\neg A}.k\, t) \longrightarrow_\beta t$$

We can read

$$\mathcal{C}(\ldots) = \texttt{try ... catch x -> x} \qquad\qquad k = \texttt{raise}$$

Each time we catch a different raise function is created.

Classical logic: reduction

In order for things to work properly, three rules are actually needed:

- the previous catch / raise reduction: for $k \notin \mathrm{FV}(t)$,

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \longrightarrow_\beta t$$

Classical logic: reduction

In order for things to work properly, three rules are actually needed:

- the previous catch / raise reduction: for $k \notin \mathrm{FV}(t)$,

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \longrightarrow_\beta t$$

- re-raising is the same as raising:

$$\mathcal{C}(\lambda k^{\neg A}.k\,\mathcal{C}(\lambda k'^{\neg A}.t)) \longrightarrow_\beta \mathcal{C}(\lambda k''^{\neg A}.t[k''/k, k''/k'])$$

Classical logic: reduction

In order for things to work properly, three rules are actually needed:

- the previous catch / raise reduction: for $k \notin \mathrm{FV}(t)$,

$$\mathcal{C}(\lambda k^{\neg A}.k\,t) \longrightarrow_\beta t$$

- re-raising is the same as raising:

$$\mathcal{C}(\lambda k^{\neg A}.k\,\mathcal{C}(\lambda k'^{\neg A}.t)) \longrightarrow_\beta \mathcal{C}(\lambda k''^{\neg A}.t[k''/k, k''/k'])$$

- application goes through catch:

$$\mathcal{C}(\lambda k^{\neg(A \to B)}.t)\,u \longrightarrow_\beta \mathcal{C}(\lambda k'^{\neg B}.t[\lambda f^{A \to B}.k\,(f\,u)/k])$$

Call-cc

The operator $\mathcal{C}$ is due to Felleisen.

A well-known variant is **call-cc** cc (for *call with current continuation*) which is typed as

$$\mathsf{cc} : (\neg A \to A) \to A$$

Classical logic: excluded middle

A proof for excluded middle is

$$\vdash \neg A \vee A$$

Classical logic: excluded middle

A proof for excluded middle is

$$\dfrac{\vdash \neg\neg(\neg A \vee A)}{\vdash \neg A \vee A}\ (\neg\neg_{\mathsf{E}})$$

Classical logic: excluded middle

A proof for excluded middle is

$$\dfrac{\dfrac{\neg(\neg A \vee A) \vdash \bot}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathsf{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathsf{E}})$$

Classical logic: excluded middle

A proof for excluded middle is

$$\dfrac{\dfrac{\dfrac{\neg(\neg A \vee A) \vdash \neg A \vee A}{\neg(\neg A \vee A) \vdash \bot}\,(\neg_{\mathrm{E}})}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathrm{E}})$$

Classical logic: excluded middle

A proof for excluded middle is

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg(\neg A \vee A) \vdash \neg A}{\neg(\neg A \vee A) \vdash \neg A \vee A}\,(\vee^{\mathsf{l}}_{\mathsf{I}})}{\neg(\neg A \vee A) \vdash \bot}\,(\neg_{\mathsf{E}})}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathsf{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathsf{E}})$$

Classical logic: excluded middle

A proof for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg(\neg A \vee A), A \vdash \bot}{\neg(\neg A \vee A) \vdash \neg A}\,(\neg_{\mathsf{I}})}{\neg(\neg A \vee A) \vdash \neg A \vee A}\,(\vee_{\mathsf{I}}^{\mathsf{l}})}{\neg(\neg A \vee A) \vdash \bot}\,(\neg_{\mathsf{E}})}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathsf{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathsf{E}})
$$

Classical logic: excluded middle

A proof for excluded middle is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg(\neg A \vee A), A \vdash \neg A \vee A}{\neg(\neg A \vee A), A \vdash \bot}\,(\neg_{\mathsf{E}})}{\neg(\neg A \vee A) \vdash \neg A}\,(\neg_{\mathsf{I}})}{\neg(\neg A \vee A) \vdash \neg A \vee A}\,(\vee^{\mathsf{l}}_{\mathsf{I}})}{\neg(\neg A \vee A) \vdash \bot}\,(\neg_{\mathsf{E}})}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathsf{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathsf{E}})$$

Classical logic: excluded middle

A proof for excluded middle is

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\neg(\neg A \vee A), A \vdash A}{\neg(\neg A \vee A), A \vdash \neg A \vee A}\;(\vee_{\mathsf{I}}^{\mathsf{r}})}{\neg(\neg A \vee A), A \vdash \bot}\;(\neg_{\mathsf{E}})}{\neg(\neg A \vee A) \vdash \neg A}\;(\neg_{\mathsf{I}})}{\neg(\neg A \vee A) \vdash \neg A \vee A}\;(\vee_{\mathsf{I}}^{\mathsf{l}})}{\neg(\neg A \vee A) \vdash \bot}\;(\neg_{\mathsf{E}})}{\vdash \neg\neg(\neg A \vee A)}\;(\neg_{\mathsf{I}})}{\vdash \neg A \vee A}\;(\neg\neg_{\mathsf{E}})
$$

Classical logic: excluded middle

A proof for excluded middle is

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(ax)}
\UnaryInfC{$\neg(\neg A \lor A), A \vdash A$}
\RightLabel{$(\lor_{\mathrm{I}}^{\mathrm{r}})$}
\UnaryInfC{$\neg(\neg A \lor A), A \vdash \neg A \lor A$}
\RightLabel{$(\neg_{\mathrm{E}})$}
\UnaryInfC{$\neg(\neg A \lor A), A \vdash \bot$}
\RightLabel{$(\neg_{\mathrm{I}})$}
\UnaryInfC{$\neg(\neg A \lor A) \vdash \neg A$}
\RightLabel{$(\lor_{\mathrm{I}}^{\mathrm{l}})$}
\UnaryInfC{$\neg(\neg A \lor A) \vdash \neg A \lor A$}
\RightLabel{$(\neg_{\mathrm{E}})$}
\UnaryInfC{$\neg(\neg A \lor A) \vdash \bot$}
\RightLabel{$(\neg_{\mathrm{I}})$}
\UnaryInfC{$\vdash \neg\neg(\neg A \lor A)$}
\RightLabel{$(\neg\neg_{\mathrm{E}})$}
\UnaryInfC{$\vdash \neg A \lor A$}
\end{prooftree}
$$

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash A}\,(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash \bot}\,(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \neg A}\,(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash \bot}\,(\neg_{\mathrm{E}})}{\vdash \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \neg A \vee A}\,(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(ax)}
\UnaryInfC{$k : \neg(\neg A \vee A), a : A \vdash a : A$}
\RightLabel{($\vee_{\mathrm{I}}^{\mathrm{r}}$)}
\UnaryInfC{$k : \neg(\neg A \vee A), a : A \vdash \qquad \neg A \vee A$}
\RightLabel{($\neg_{\mathrm{E}}$)}
\UnaryInfC{$k : \neg(\neg A \vee A), a : A \vdash \qquad \bot$}
\RightLabel{($\neg_{\mathrm{I}}$)}
\UnaryInfC{$k : \neg(\neg A \vee A) \vdash \qquad \neg A$}
\RightLabel{($\vee_{\mathrm{I}}^{\mathrm{l}}$)}
\UnaryInfC{$k : \neg(\neg A \vee A) \vdash \qquad \neg A \vee A$}
\RightLabel{($\neg_{\mathrm{E}}$)}
\UnaryInfC{$k : \neg(\neg A \vee A) \vdash \qquad \bot$}
\RightLabel{($\neg_{\mathrm{I}}$)}
\UnaryInfC{$\vdash \qquad \neg\neg(\neg A \vee A)$}
\RightLabel{($\neg\neg_{\mathrm{E}}$)}
\UnaryInfC{$\vdash \qquad \neg A \vee A$}
\end{prooftree}
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\,(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash \quad \bot}\,(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \quad \neg A}\,(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \quad \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash \quad \bot}\,(\neg_{\mathrm{E}})}{\vdash \quad \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \quad \neg A \vee A}\,(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\;(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\;(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\, \iota_{\mathrm{r}}(a) : \bot}\;(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \qquad \neg A}\;(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \qquad \neg A \vee A}\;(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash \qquad \bot}\;(\neg_{\mathrm{E}})}{\vdash \qquad \neg\neg(\neg A \vee A)}\;(\neg_{\mathrm{I}})}{\vdash \qquad \neg A \vee A}\;(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\,(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\,\iota_{\mathrm{r}}(a) : \bot}\,(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \lambda a^A.k\,\iota_{\mathrm{r}}(a) : \neg A}\,(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \qquad \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash \qquad \bot}\,(\neg_{\mathrm{E}})}{\vdash \qquad \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \qquad \neg A \vee A}\,(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\,(\text{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathsf{r}}(a) : \neg A \vee A}\,(\vee_{\mathsf{I}}^{\mathsf{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\,\iota_{\mathsf{r}}(a) : \bot}\,(\neg_{\mathsf{E}})}{k : \neg(\neg A \vee A) \vdash \lambda a^{A}.k\,\iota_{\mathsf{r}}(a) : \neg A}\,(\neg_{\mathsf{I}})}{k : \neg(\neg A \vee A) \vdash \iota_{\mathsf{l}}(\lambda a^{A}.k\,\iota_{\mathsf{r}}(a)) : \neg A \vee A}\,(\vee_{\mathsf{I}}^{\mathsf{l}})}{k : \neg(\neg A \vee A) \vdash \qquad \bot}\,(\neg_{\mathsf{E}})}{\vdash \qquad \neg\neg(\neg A \vee A)}\,(\neg_{\mathsf{I}})}{\vdash \qquad \neg A \vee A}\,(\neg\neg_{\mathsf{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\,(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\,\iota_{\mathrm{r}}(a) : \bot}\,(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \lambda a^A . k\,\iota_{\mathrm{r}}(a) : \neg A}\,(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \iota_{\mathrm{l}}(\lambda a^A . k\,\iota_{\mathrm{r}}(a)) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash k\,\iota_{\mathrm{l}}(\lambda a^A . k\,\iota_{\mathrm{r}}(a)) : \bot}\,(\neg_{\mathrm{E}})}{\vdash \qquad\qquad \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \qquad\qquad \neg A \vee A}\,(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\,(\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\,\iota_{\mathrm{r}}(a) : \bot}\,(\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \lambda a^{A}.k\,\iota_{\mathrm{r}}(a) : \neg A}\,(\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \iota_{\mathrm{l}}(\lambda a^{A}.k\,\iota_{\mathrm{r}}(a)) : \neg A \vee A}\,(\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash k\,\iota_{\mathrm{l}}(\lambda a^{A}.k\,\iota_{\mathrm{r}}(a)) : \bot}\,(\neg_{\mathrm{E}})}{\vdash \lambda k^{\neg(\neg A \vee A)}.k\,\iota_{\mathrm{l}}(\lambda a^{A}.k\,\iota_{\mathrm{r}}(a)) : \neg\neg(\neg A \vee A)}\,(\neg_{\mathrm{I}})}{\vdash \qquad\qquad\qquad\qquad \neg A \vee A}\,(\neg\neg_{\mathrm{E}})
$$

Classical logic: excluded middle

A term for excluded middle is

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{k : \neg(\neg A \vee A), a : A \vdash a : A}\ (\mathrm{ax})}{k : \neg(\neg A \vee A), a : A \vdash \iota_{\mathrm{r}}(a) : \neg A \vee A}\ (\vee_{\mathrm{I}}^{\mathrm{r}})}{k : \neg(\neg A \vee A), a : A \vdash k\,\iota_{\mathrm{r}}(a) : \bot}\ (\neg_{\mathrm{E}})}{k : \neg(\neg A \vee A) \vdash \lambda a^A.k\,\iota_{\mathrm{r}}(a) : \neg A}\ (\neg_{\mathrm{I}})}{k : \neg(\neg A \vee A) \vdash \iota_{\mathrm{l}}(\lambda a^A.k\,\iota_{\mathrm{r}}(a)) : \neg A \vee A}\ (\vee_{\mathrm{I}}^{\mathrm{l}})}{k : \neg(\neg A \vee A) \vdash k\,\iota_{\mathrm{l}}(\lambda a^A.k\,\iota_{\mathrm{r}}(a)) : \bot}\ (\neg_{\mathrm{E}})}{\vdash \lambda k^{\neg(\neg A \vee A)}.k\,\iota_{\mathrm{l}}(\lambda a^A.k\,\iota_{\mathrm{r}}(a)) : \neg\neg(\neg A \vee A)}\ (\neg_{\mathrm{I}})}{\vdash \mathcal{C}(\lambda k^{\neg(\neg A \vee A)}.k\,\iota_{\mathrm{l}}(\lambda a^A.k\,\iota_{\mathrm{r}}(a))) : \neg A \vee A}\ (\neg\neg_{\mathrm{E}})
$$

Part VII

Strong normalization

Strong normalization

A term t is **strongly normalizing** (or SN, or **terminating**) if there is no infinite sequence of reductions starting from t:

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta t_3 \longrightarrow_\beta \dots$$

Theorem (Strong normalization)
The simply-typed λ-calculus is strongly normalizing: given a typable λ-term t, there is no infinite sequence of β-reductions starting from t.

Strong normalization

Theorem (Strong normalization)
The simply-typed λ-calculus is strongly normalizing: given a typable λ-term t, there is no infinite sequence of β-reductions starting from t.

For instance, the λ-term

$$(\lambda x.xx)(\lambda x.xx)$$

is <u>not</u> typable.

Strong normalization

Theorem (Strong normalization)
The simply-typed λ-calculus is strongly normalizing: given a typable λ-term t, there is no infinite sequence of β-reductions starting from t.

For instance, the λ-term

$$(\lambda x.xx)(\lambda x.xx)$$

is not typable.

```
# let omega = (fun x -> x x) (fun x -> x x);;
                          ^
Error: This expression has type 'a -> 'b
       but an expression was expected of type 'a
       The type variable 'a occurs inside 'a -> 'b
```

Deciding β-equivalence

Recall that β-**equivalence** is the smallest equivalence relation generated by β-reduction.

This means that $t =_\beta u$ when there exists a sequence of reductions

$$t \xleftarrow{*} t_1 \xrightarrow{*} t_2 \xleftarrow{*} t_3 \xrightarrow{*} t_4 \xleftarrow{*} \dots \xrightarrow{*} u$$

How can we decide whether two terms are β-equivalent or not?

(remember this is undecidable for untyped λ-calculus)

The Church-Rosser theorem

A first simplification:

Theorem (Church-Rosser)
Two terms t and u are β-equivalent iff and only if there exists w such that

$$t \xrightarrow{*}_{\beta} w \;{}_{\beta}\!\xleftarrow{*} u$$

Proof.
The only if part is obvious. Suppose that we have

$$\begin{tikzcd}
& t_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_2 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_3 \arrow[dl, "*"'] & \cdots & t_n \arrow[dr, "*"] & \\
t = u_0 & & u_1 & & u_2 & & & & u_n = u
\end{tikzcd}$$

we show the result by induction on n. For $n = 0$, $t = u$ and the result is immediate. $\square$

The Church-Rosser theorem

A first simplification:

Theorem (Church-Rosser)
Two terms t and u are β-equivalent iff and only if there exists w such that

$$t \xrightarrow{*}_\beta w \, {}_\beta\!\xleftarrow{*} u$$

Proof.
The only if part is obvious. Suppose that we have

$$\begin{tikzcd}
& t_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_2 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_3 \arrow[dl, "*"'] & \dots & t_n \arrow[dr, "*"] & \\
t = u_0 & & u_1 & & u_2 & & & & u_n = u
\end{tikzcd}$$

Otherwise, $\square$

The Church-Rosser theorem

A first simplification:

Theorem (Church-Rosser)
Two terms t and u are β-equivalent iff and only if there exists w such that

$$t \xrightarrow{*}_{\beta} w \,{}_{\beta}\!\xleftarrow{*} u$$

Proof.
The only if part is obvious. Suppose that we have

$$\begin{tikzcd}
 & t_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_2 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_3 \arrow[dl, "*"'] & \ldots & t_n \arrow[dr, "*"] & \\
t = u_0 \arrow[dr, "*"'] & & u_1 \arrow[dl, "*"] & & u_2 & & & & u_n = u \\
 & w_1 & & & & & & &
\end{tikzcd}$$

Otherwise, by confluence $\square$

The Church-Rosser theorem

A first simplification:

Theorem (Church-Rosser)
Two terms t and u are β-equivalent iff and only if there exists w such that

$$t \xrightarrow{*}_{\beta} w \;{}_{\beta}\xleftarrow{*}\; u$$

Proof.
The only if part is obvious. Suppose that we have

$$\begin{tikzcd}
& t_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_2 \arrow[dl, "*"'] \arrow[dr, "*"] & & t_3 \arrow[dl, "*"'] & \dots & t_n \arrow[dr, "*"] & \\
t = u_0 \arrow[dr, "*"'] & & u_1 \arrow[dl, "*"] & & u_2 & & & & u_n = u \arrow[dl, "*"] \\
& w_1 \arrow[rrrrrr, "*"'] & & & & & & w &
\end{tikzcd}$$

Otherwise, by confluence and induction hypothesis. □

Joinability

We thus left with deciding whether two terms t and u are **joinable**, i.e. whether there exists w such that

$$t \xrightarrow{*}_{\beta} w \;{}_{\beta}\!\xleftarrow{*}\; u$$

Normal forms

A term t is a **normal form** when there is no t' such that $t \longrightarrow_\beta t'$.

Normal forms

A term t is a **normal form** when there is no t' such that $t \longrightarrow_\beta t'$.

Lemma
Every typable term t is β-equivalent to a normal form $\hat{t}$.

Proof.

Normal forms

A term t is a **normal form** when there is no t' such that $t \longrightarrow_\beta t'$.

Lemma
Every typable term t is β-equivalent to a normal form $\hat{t}$.

Proof.

Given a term t reduce it as much as possible:

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \cdots \longrightarrow_\beta t_n = \hat{t}$$

This process will stop because typable terms are strongly normalizing and t_n is a normal form. □

Normal forms

A term t is a **normal form** when there is no t' such that $t \longrightarrow_\beta t'$.

Lemma
Every typable term t is β-equivalent to a normal form $\hat{t}$.

Proof.
Given a term t reduce it as much as possible:

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \cdots \longrightarrow_\beta t_n = \hat{t}$$

This process will stop because typable terms are strongly normalizing and t_n is a normal form. $\square$

Strongly normalizing: any sequence of reductions will eventually lead to a normal form.

Lemma
Two normal forms t and u are β-equivalent iff they are equal.

Proof.

Normal forms

Lemma
Two normal forms t and u are β-equivalent iff they are equal.

Proof.

The only if part is obvious. For the if part, by the Church-Rosser theorem we have

$$t \xrightarrow{*}_{\beta} w \;{}_{\beta}\!\xleftarrow{*}\; u$$

but since t and u are normal forms, we actually have

$$t = w = u \qquad \square$$

Deciding β-equivalence

Suppose given two typable terms t and u. The following are equivalent

- $t =_{\beta} u$
- $\hat{t} =_{\beta} \hat{u}$
- $\hat{t} = \hat{u}$

Which can be pictured as

$$\begin{tikzcd} t \arrow[r, leftrightarrow, dotted, "?", "*"'] \arrow[d, "*"'] & u \arrow[d, "*"] \\ \hat{t} \arrow[r, equal, dotted, "?"'] & \hat{u} \end{tikzcd}$$

Extensions

This still holds for extensions of λ-calculus: products, coproducts, natural numbers, etc.

In particular, for natural numbers it is important that the recursive calls are performed on smaller numbers, which ensures termination.

Part VIII

Type inference à la Curry

Curry style λ-calculus

In Curry style, λ-terms are

$$t ::= x \mid t\,u \mid \lambda x.t$$

and the rules are

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\,(\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\,(\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,(\to_{\mathrm{I}})$$

Curry style λ-calculus

In Curry style, λ-terms are

$$t ::= x \mid t\,u \mid \lambda x.t$$

and the rules are

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\,(\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\,(\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,(\to_{\mathrm{I}})$$

A term can have multiple types:

$$\frac{\dfrac{}{x : A \vdash x : A}\,(\mathrm{ax})}{\vdash \lambda x.x : A \to A}\,(\to_{\mathrm{I}}) \qquad \frac{\dfrac{}{x : A \to A \vdash x : A \to A}\,(\mathrm{ax})}{\vdash \lambda x.x : (A \to A) \to (A \to A)}\,(\to_{\mathrm{I}})$$

Curry style λ-calculus

In Curry style, λ-terms are

$$t ::= x \mid t\,u \mid \lambda x.t$$

and the rules are

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}})$$

A term can have multiple types:

$$\dfrac{\dfrac{}{x : A \vdash x : A}\ (\mathrm{ax})}{\vdash \lambda x.x : A \to A}\ (\to_{\mathrm{I}}) \qquad \dfrac{\dfrac{}{x : A \to A \vdash x : A \to A}\ (\mathrm{ax})}{\vdash \lambda x.x : (A \to A) \to (A \to A)}\ (\to_{\mathrm{I}})$$

How do we compute all those types?

Substitutions

A **substitution** is a function which to type variables associate terms. For instance

$$\sigma(X) = A \to B \qquad\qquad \sigma(Y) = A$$

Substitutions

A **substitution** is a function which to type variables associate terms. For instance

$$\sigma(X) = A \to B \qquad\qquad \sigma(Y) = A$$

We write $A[\sigma]$ for the type A where variables have been replaced according to σ:

$$(X \to Y)[\sigma] = (A \to B) \to A$$

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

A substitution σ is a solution of E if

$$A_i[\sigma] = B_i[\sigma]$$

for every index i.

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

A substitution σ is a solution of E if

$$A_i[\sigma] = B_i[\sigma]$$

for every index i.

For instance, a solution of

$$\{(X \to Y) \stackrel{?}{=} (Z \to (Z \to Z))\}$$

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

A substitution σ is a solution of E if

$$A_i[\sigma] = B_i[\sigma]$$

for every index i.

For instance, a solution of

$$\{(X \to Y) \stackrel{?}{=} (Z \to (Z \to Z))\}$$

is $\sigma(X) = Z$, $\sigma(Y) = Z \to Z$.

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

A substitution σ is a solution of E if

$$A_i[\sigma] = B_i[\sigma]$$

for every index i.

For instance, a solution of

$$\{(X \to Y) \stackrel{?}{=} Y\}$$

Type equation systems

A type equation system is a finite set

$$E \quad = \quad \{A_1 \stackrel{?}{=} B_1, \ldots, A_n \stackrel{?}{=} B_n\}$$

of pairs of types A_i and B_i.

A substitution σ is a solution of E if

$$A_i[\sigma] = B_i[\sigma]$$

for every index i.

For instance, a solution of

$$\{(X \to Y) \stackrel{?}{=} Y\}$$

does not exist.

Typing as solving constraints

We are going to associate to each term t

- a type A_t
- an equation system E_t

such that

- t is typable iff E_t has a solution σ,
- in which case the $A_t[\sigma]$ are the possible types of t.

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}\ (\to_\text{E}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_\text{I})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$E_x = \qquad\qquad A_x =$$

$$E_{t\,u} = \qquad\qquad A_{t\,u} =$$

$$E_{\lambda x.t} = \qquad\qquad A_{\lambda x.t} =$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$E_x = \emptyset \qquad\qquad A_x =$$

$$E_{t\,u} = \qquad\qquad A_{t\,u} =$$

$$E_{\lambda x.t} = \qquad\qquad A_{\lambda x.t} =$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$E_x = \emptyset \qquad\qquad A_x = X_x$$

$$E_{t\,u} = \qquad\qquad A_{t\,u} =$$

$$E_{\lambda x.t} = \qquad\qquad A_{\lambda x.t} =$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \text{(ax)} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B} (\to_\mathrm{E}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} (\to_\mathrm{I})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$\begin{aligned} E_x &= \emptyset & A_x &= X_x \\ E_{t\,u} &= E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \to X)\} & A_{t\,u} &= \\ E_{\lambda x.t} &= & A_{\lambda x.t} &= \end{aligned}$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$E_x = \emptyset \qquad A_x = X_x$$

$$E_{t\,u} = E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \to X)\} \qquad A_{t\,u} = X \qquad \text{with } X \text{ fresh}$$

$$E_{\lambda x.t} = \qquad A_{\lambda x.t} =$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\text{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\text{I}})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$\begin{aligned} E_x &= \emptyset & A_x &= X_x & \\ E_{t\,u} &= E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \to X)\} & A_{t\,u} &= X & \text{with } X \text{ fresh} \\ E_{\lambda x.t} &= E_t & A_{\lambda x.t} &= & \end{aligned}$$

Typing as solving constraints

The rules

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_{\mathrm{I}})$$

suggest that

- to every term variable x, we associate a type variable X_x,
- to every term t, we associate a type A_t,
- to every term t, we associate an equation system E_t

by induction by

$$\begin{aligned}
E_x &= \emptyset & A_x &= X_x & \\
E_{t\,u} &= E_t \cup E_u \cup \{A_t \stackrel{?}{=} (A_u \to X)\} & A_{t\,u} &= X & \text{with } X \text{ fresh} \\
E_{\lambda x.t} &= E_t & A_{\lambda x.t} &= X_x \to A_t &
\end{aligned}$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_x = \emptyset \qquad\qquad A_x = X_x$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_x = \emptyset \qquad\qquad A_x = X_x$$

$$E_{\lambda x.x} = \emptyset \qquad\qquad A_{\lambda x.x} = X_x \to X_x$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_x = \emptyset \qquad A_x = X_x$$

$$E_{\lambda x.x} = \emptyset \qquad A_{\lambda x.x} = X_x \to X_x$$

$$E_{f(\lambda x.x)} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X\} \qquad A_{f(\lambda x.x)} = X$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$\begin{aligned} E_x &= \emptyset & A_x &= X_x \\ E_{\lambda x.x} &= \emptyset & A_{\lambda x.x} &= X_x \to X_x \\ E_{f(\lambda x.x)} &= \{X_f \stackrel{?}{=} (X_x \to X_x) \to X\} & A_{f(\lambda x.x)} &= X \\ E_{f(f(\lambda x.x))} &= \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} & A_{f(f(\lambda x.x))} &= Y \end{aligned}$$

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$\begin{aligned}
E_x &= \emptyset & A_x &= X_x \\
E_{\lambda x.x} &= \emptyset & A_{\lambda x.x} &= X_x \to X_x \\
E_{f(\lambda x.x)} &= \{X_f \stackrel{?}{=} (X_x \to X_x) \to X\} & A_{f(\lambda x.x)} &= X \\
E_{f(f(\lambda x.x))} &= \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} & A_{f(f(\lambda x.x))} &= Y \\
E_{\lambda f.f(f(\lambda x.x))} &= \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} & A_{\lambda f.f(f(\lambda x.x))} &= X_f \to Y
\end{aligned}$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A \quad \sigma(X) = A \to A$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A \quad \sigma(X) = A \to A \quad \sigma(Y) = A \to A$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A \quad \sigma(X) = A \to A \quad \sigma(Y) = A \to A \quad \sigma(X_f) = (A \to A) \to (A \to A)$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A \quad \sigma(X) = A \to A \quad \sigma(Y) = A \to A \quad \sigma(X_f) = (A \to A) \to (A \to A)$$

The resulting type is

$$(X_f \to Y)[\sigma] = ((A \to A) \to (A \to A)) \to A \to A$$

Typing as solving constraints

For instance, consider

$$t = \lambda f.f(f(\lambda x.x))$$

we have

$$E_{\lambda f.f(f(\lambda x.x))} = \{X_f \stackrel{?}{=} (X_x \to X_x) \to X, X_f \stackrel{?}{=} X \to Y\} \quad A_{\lambda f.f(f(\lambda x.x))} = X_f \to Y$$

A solution is

$$\sigma(X_x) = A \quad \sigma(X) = A \to A \quad \sigma(Y) = A \to A \quad \sigma(X_f) = (A \to A) \to (A \to A)$$

The resulting type is

$$(X_f \to Y)[\sigma] = ((A \to A) \to (A \to A)) \to A \to A$$

to be compared with

```
# fun f -> f (f (fun x -> x));;
- : (('a -> 'a) -> 'a -> 'a) -> 'a -> 'a = <fun>
```

Typing as solving constraints

Note that the previous solution works for which ever type A we choose.

Therefore there is an infinite number of solutions!

Typing as solving constraints

Theorem
We have

- *if* $\Gamma \vdash t : A$ *then there is a solution* σ *of* E_t *such that* $A = A_t[\sigma]$ *and* $\Gamma(x) = \sigma(X_x)$ *for every variable* $x \in \mathsf{FV}(t)$,
- *for every solution* σ *of* E_t, *if we write* Γ *for a context such that* $\Gamma(x) = \sigma(x)$ *for every free variable* $x \in \mathsf{FV}(t)$, *then* $\Gamma \vdash t : A_t[\sigma]$ *is derivable.*

Otherwise said, there is a bijection between

- solutions σ of E_t,
- pairs (Γ, A) such that $\Gamma \vdash t : A$ is derivable.

Unification

The **unification** algorithm takes a type equation system E and produces a solution σ when there exists one, in polynomial time.

Unification

The **unification** algorithm takes a type equation system E and produces a solution σ when there exists one, in polynomial time.

Moreover, this solution is the *most general one*: any other solution τ satisfies

$$\tau = \tau' \circ \sigma$$

Unification

The **unification** algorithm takes a type equation system E and produces a solution σ when there exists one, in polynomial time.

Moreover, this solution is the *most general one*: any other solution τ satisfies

$$\tau = \tau' \circ \sigma$$

We can therefore compute a most general type for Curry-style λ-terms in P-time!

Part IX

Bidirectional typechecking

Bidirectional typechecking

Looking closely at the operations we perform during type-checking there are two phases.

- **Type inference**: we guess the type of a term.
- **Type checking**: we check that a term has a given type.

For instance, consider the type inference of

$$\dfrac{\dfrac{\dfrac{}{x : \mathbb{N} \vdash x : \mathbb{N}}}{\vdash \lambda x^{\mathbb{N}}.x : \mathbb{N} \to \mathbb{N}} \qquad \dfrac{}{\vdash 5 : \mathbb{N}}}{\vdash (\lambda x^{\mathbb{N}}.x)5 : \mathbb{N}}$$

Bidirectional typechecking

Bidirectional typechecking formalizes this two phases, allowing to add type annotations (we could mix Church and Curry style).

Bidirectional typechecking

We consider Curry-style terms with type annotations:

$$t ::= x \mid t\,u \mid \lambda x.t \mid (x : A)$$

Bidirectional typechecking

We consider Curry-style terms with type annotations:

$$t ::= x \mid t\,u \mid \lambda x.t \mid (x : A)$$

We consider two kind of sequents:

- $\Gamma \vdash t \Rightarrow A$: the term t allows to synthesize the type A (type inference),
- $\Gamma \vdash t \Leftarrow A$: the term t allows checks against the type A (type checking).

Bidirectional typechecking

We consider Curry-style terms with type annotations:

$$t ::= x \mid t\,u \mid \lambda x.t \mid (x : A)$$

We consider two kind of sequents:

- $\Gamma \vdash t \Rightarrow A$: the term t allows to synthesize the type A (type inference),
- $\Gamma \vdash t \Leftarrow A$: the term t allows checks against the type A (type checking).

We can then orient the typing rules

$$\Gamma \vdash t : A \qquad \text{as} \qquad \Gamma \vdash t \Rightarrow A \qquad \text{or} \qquad \Gamma \vdash t :\Leftarrow A$$

Bidirectional typechecking

Orientation of the base rules

- variable: we already have the information in the context

$$\frac{}{\Gamma \vdash x \Rightarrow \Gamma(x)} \text{(ax)}$$

Bidirectional typechecking

Orientation of the base rules

- variable: we already have the information in the context

$$\frac{}{\Gamma \vdash x \Rightarrow \Gamma(x)}\ (\mathrm{ax})$$

- application: we cannot come up with B, we have to check the type of the argument

$$\frac{\Gamma \vdash t \Rightarrow A \to B \qquad \Gamma \vdash u \Leftarrow A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

Bidirectional typechecking

Orientation of the base rules

- variable: we already have the information in the context

$$\frac{}{\Gamma \vdash x \Rightarrow \Gamma(x)}\ (\mathrm{ax})$$

- application: we cannot come up with B, we have to check the type of the argument

$$\frac{\Gamma \vdash t \Rightarrow A \to B \qquad \Gamma \vdash u \Leftarrow A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

- abstraction: we cannot come up with A in Curry style (and typing is not unique)

$$\frac{\Gamma, x : A \vdash t \Leftarrow B}{\Gamma \vdash \lambda x.t \Leftarrow A \to B}\ (\to_{\mathrm{I}})$$

Bidirectional typechecking

We have two new rules:

- subsumption: if we can infer then we can check

$$\frac{\Gamma \vdash t \Rightarrow A}{\Gamma \vdash t \Leftarrow A}$$

Bidirectional typechecking

We have two new rules:

- subsumption: if we can infer then we can check

$$\frac{\Gamma \vdash t \Rightarrow A}{\Gamma \vdash t \Leftarrow A}$$

- casting:

$$\frac{\Gamma \vdash t \Leftarrow A}{\Gamma \vdash (t : A) \Rightarrow A}$$

Bidirectional typechecking

$$\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}$$

Bidirectional typechecking

$$\frac{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R} \qquad \Gamma \vdash 7 \Leftarrow \mathbb{R}}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}$$

Bidirectional typechecking

$$\frac{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R} \qquad \dfrac{\Gamma \vdash 7 \Rightarrow \mathbb{R}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}$$

Bidirectional typechecking

$$\dfrac{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R} \qquad \dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}$$

Bidirectional typechecking

$$
\dfrac{
\dfrac{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x) \Rightarrow \mathbb{R} \to \mathbb{R} \to \mathbb{R} \qquad \vdots}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R}}
\qquad
\dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}
}{
\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}
}
$$

Bidirectional typechecking

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\Gamma \vdash \mathsf{mean} \Rightarrow (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}}
      \qquad
      \Gamma \vdash \lambda x.x \Leftarrow \mathbb{R} \to \mathbb{R}
    }{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x) \Rightarrow \mathbb{R} \to \mathbb{R} \to \mathbb{R}}
    \qquad
    \vdots
  }{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R}}
  \qquad
  \dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}
}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}
$$

Bidirectional typechecking

$$
\dfrac{
  \dfrac{
    \dfrac{}{\Gamma \vdash \mathsf{mean} \Rightarrow (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}}
    \qquad
    \dfrac{\Gamma, x : \mathbb{R} \vdash x \Leftarrow \mathbb{R}}{\Gamma \vdash \lambda x.x \Leftarrow \mathbb{R} \to \mathbb{R}}
  }{
    \Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x) \Rightarrow \mathbb{R} \to \mathbb{R} \to \mathbb{R}
  }
  \quad \vdots
}{
  \Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R}
}
\qquad
\dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}
$$

$$
\overline{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}
$$

Bidirectional typechecking

$$
\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathsf{mean} \Rightarrow (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}} \quad \dfrac{\dfrac{\Gamma, x : \mathbb{R} \vdash x \Rightarrow \mathbb{R}}{\Gamma, x : \mathbb{R} \vdash x \Leftarrow \mathbb{R}}}{\Gamma \vdash \lambda x.x \Leftarrow \mathbb{R} \to \mathbb{R}}}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x) \Rightarrow \mathbb{R} \to \mathbb{R} \to \mathbb{R}} \quad \vdots}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R}} \quad \dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}}{\Gamma \vdash \mathsf{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{}{\Gamma \vdash \text{mean} \Rightarrow (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}}
    \qquad
    \dfrac{\dfrac{\dfrac{}{\Gamma, x : \mathbb{R} \vdash x \Rightarrow \mathbb{R}}}{\Gamma, x : \mathbb{R} \vdash x \Leftarrow \mathbb{R}}}{\Gamma \vdash \lambda x.x \Leftarrow \mathbb{R} \to \mathbb{R}}
  }{\Gamma \vdash \text{mean}\,(\lambda x.x \times x) \Rightarrow \mathbb{R} \to \mathbb{R} \to \mathbb{R}}
  \qquad \vdots
  }{\Gamma \vdash \text{mean}\,(\lambda x.x \times x)\,5 \Rightarrow \mathbb{R} \to \mathbb{R}}
  \qquad
  \dfrac{\dfrac{}{\Gamma \vdash 7 \Rightarrow \mathbb{R}}}{\Gamma \vdash 7 \Leftarrow \mathbb{R}}
}{\Gamma \vdash \text{mean}\,(\lambda x.x \times x)\,5\,7 \Rightarrow \mathbb{R}}
$$

If we try to define the mean function as

$$\lambda fxy.(f\,x + f\,y)/2$$

we cannot infer its type: we can only check the type of functions (i.e. when they are used as arguments of other functions).

Bidirectional typechecking

If we try to define the mean function as

$$\lambda fxy.(f\ x + f\ y)/2$$

we cannot infer its type: we can only check the type of functions (i.e. when they are used as arguments of other functions).

In order to define it, we have to provide its type

$$\text{mean} \quad = \quad (\lambda fxy.(f\ x + f\ y)/2 : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R})$$

If we try to define the mean function as

$$\lambda fxy.(f\,x + f\,y)/2$$

we cannot infer its type: we can only check the type of functions (i.e. when they are used as arguments of other functions).

In order to define it, we have to provide its type

$$\mathsf{mean} \quad = \quad (\lambda fxy.(f\,x + f\,y)/2 : (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R})$$

In Agda, the syntax for such definitions will be

```
mean : (R → R) → R → R → R
mean f x y = (x + y) / 2
```

Implementation

The implementation is pretty direct.

We define types

```
type ty =
  | TVar of string
  | Arr  of ty * ty
```

Implementation

The implementation is pretty direct.

We define types

```
type ty =
  | TVar of string
  | Arr  of ty * ty
```

and terms:

```
type term =
  | Var  of string
  | App  of term   * term
  | Abs  of string * term
  | Cast of term   * ty
```

Implementation

```
(** Type inference. *)
let rec infer env = function
  | Var x ->
    (try List.assoc x env
     with Not_found ->
          raise Type_error)
  | App (t, u) ->
    (
      match infer env t with
      | Arr (a, b) -> check env u a; b
      | _ -> raise Type_error
    )
  | Abs (x, t) -> raise Cannot_infer
  | Cast (t, a) -> check env t a; a
```

Implementation

```
(** Type inference. *)
let rec infer env = function
  | Var x ->
    (try List.assoc x env
     with Not_found ->
          raise Type_error)
  | App (t, u) ->
    (
      match infer env t with
      | Arr (a, b) -> check env u a; b
      | _ -> raise Type_error
    )
  | Abs (x, t) -> raise Cannot_infer
  | Cast (t, a) -> check env t a; a
```

```
(** Type checking. *)
and check env t a =
  match t , a with
  | Abs (x, t) , Arr (a, b) ->
    check ((x, a)::env) t b
  | _ -> if infer env t <> a then
              raise Type_error
```

Part X

Proof of strong normalization

We now want to prove

Theorem
*Typed λ-terms are **strongly normalizing**.*

Given a term t which is typable, there is no infinite sequence of reductions

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \ldots$$

Proof of strong normalization: first attempt

A naive try would be by induction on the derivation of $\Gamma \vdash t : A$.

If the last rule is

$$\frac{}{\Gamma \vdash x : A}\ (\mathrm{ax})$$

x is clearly strongly normalizing.

Proof of strong normalization: first attempt

A naive try would be by induction on the derivation of $\Gamma \vdash t : A$.

If the last rule is

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B} \ (\to_\mathrm{I})$$

A sequence of reductions is of the form

Proof of strong normalization: first attempt

A naive try would be by induction on the derivation of $\Gamma \vdash t : A$.

If the last rule is

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A . t : A \to B} \, (\to_\mathrm{I})$$

A sequence of reductions is of the form

$$\lambda x.t \longrightarrow_\beta \lambda x.t_1 \longrightarrow_\beta \lambda x.t_2 \longrightarrow_\beta \ldots$$

with

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \ldots$$

and we conclude by induction hypothesis.

Proof of strong normalization: first attempt

A naive try would be by induction on the derivation of $\Gamma \vdash t : A$.

If the last rule is

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\;(\to_{\mathrm{E}})$$

A reduction starting from $t\,u$ can be of the form

Proof of strong normalization: first attempt

A naive try would be by induction on the derivation of $\Gamma \vdash t : A$.

If the last rule is

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\,(\to_{\mathrm{E}})$$

A reduction starting from $t\,u$ can be of the form

$$t\,u \longrightarrow t'\,u \qquad \text{or} \qquad t\,u \longrightarrow t\,u'$$

but also

$$(\lambda x.t)u \longrightarrow t[u/x]$$

e.g. $\mathrm{I}\,\mathrm{I}$, for which we cannot say anything.

Reducibility candidates

Instead, we take an optimistic approach and defined, for each type A, a set

$$R_A$$

of terms, the **reducibility candidates** for the type A, such that

- for every term t of type A (in whichever context), we have $t \in R_A$,
- the terms of R_A are obviously terminating.

NB:

- the definition has to be carefully crafted in order to be able to reason by induction,
- the terms of R_A are not necessarily of type A (although we could).

Reducibility candidates

We define R_A by induction on A by

Reducibility candidates

We define R_A by induction on A by

- in the case of a variable,

$$R_X \quad = \quad \{t \mid t \text{ is strongly normalizing}\}$$

Reducibility candidates

We define R_A by induction on A by

- in the case of a variable,

$$R_X \quad = \quad \{t \mid t \text{ is strongly normalizing}\}$$

- in the case of an arrow,

$$R_{A \to B} \quad = \quad \{t \mid \text{for every } u \in R_A, \text{ we have } t\, u \in R_B\}$$

Reducibility candidates

We define R_A by induction on A by

- in the case of a variable,

$$R_X \quad = \quad \{t \mid t \text{ is strongly normalizing}\}$$

- in the case of an arrow,

$$R_{A \to B} \quad = \quad \{t \mid \text{for every } u \in R_A, \text{ we have } t\,u \in R_B\}$$

We still have to show that

- a term $t \in R_A$ is strongly normalizing,
- if $\Gamma \vdash t : A$ is derivable then $t \in R_A$.

Induction for SN terms

Proposition
Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$
\begin{tikzcd}
 & & t \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
t'_1 & t'_2 & t'_3 & \dots & t'_k
\end{tikzcd}
\qquad\qquad
\begin{array}{c}
P(t) \\
\Uparrow \\
P(t'_i) \text{ for every } i
\end{array}
$$

Then $P(t)$ holds for every SN term t.

Induction for SN terms

Proposition
Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$
\begin{tikzcd}
 & & t \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
t'_1 & t'_2 & t'_3 & \dots & t'_k
\end{tikzcd}
\qquad\qquad
\begin{array}{c} P(t) \\ \Uparrow \\ P(t'_i) \text{ for every } i \end{array}
$$

Then $P(t)$ holds for every SN term t.

Proof.
By contraposition. Suppose that $P(t)$ does not hold for some SN term t.

Induction for SN terms

Proposition
Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$\begin{array}{ccccc} & & t & & \\ \swarrow & \swarrow & \downarrow & & \searrow \\ t'_1 & t'_2 & t'_3 & \dots & t'_k \end{array} \qquad\qquad \begin{array}{c} P(t) \\ \Uparrow \\ P(t'_i) \text{ for every } i \end{array}$$

Then $P(t)$ holds for every SN term t.

Proof.
By contraposition. Suppose that $P(t)$ does not hold for some SN term t.

- then there exists t_1 with $t \longrightarrow_\beta t_1$ such that $P(t_1)$ does not hold,

Induction for SN terms

Proposition

Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$
\begin{tikzcd}
 & & t \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
t'_1 & t'_2 & t'_3 & \dots & t'_k
\end{tikzcd}
\qquad\qquad
\begin{array}{c}
P(t) \\
\Uparrow \\
P(t'_i) \text{ for every } i
\end{array}
$$

Then $P(t)$ holds for every SN term t.

Proof.

By contraposition. Suppose that $P(t)$ does not hold for some SN term t.

- then there exists t_1 with $t \longrightarrow_\beta t_1$ such that $P(t_1)$ does not hold,
- then there exists t_2 with $t_1 \longrightarrow_\beta t_2$ such that $P(t_2)$ does not hold,

Induction for SN terms

Proposition
Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$
\begin{tikzcd}
 & & t \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
t'_1 & t'_2 & t'_3 & \dots & t'_k
\end{tikzcd}
\qquad\qquad
\begin{array}{c}
P(t) \\
\Uparrow \\
P(t'_i) \text{ for every } i
\end{array}
$$

Then $P(t)$ holds for every SN term t.

Proof.
By contraposition. Suppose that $P(t)$ does not hold for some SN term t.

- then there exists t_1 with $t \longrightarrow_\beta t_1$ such that $P(t_1)$ does not hold,
- then there exists t_2 with $t_1 \longrightarrow_\beta t_2$ such that $P(t_2)$ does not hold,
- ...

Induction for SN terms

Proposition
Suppose given a property $P(t)$ on terms. Suppose that, for any term t, if $P(t')$ for every t' with $t \longrightarrow_\beta t'$, then $P(t)$:

$$
\begin{tikzcd}
 & & t \arrow[dll] \arrow[dl] \arrow[d] \arrow[drr] & & \\
t'_1 & t'_2 & t'_3 & \ldots & t'_k
\end{tikzcd}
\qquad\qquad
\begin{array}{c}
P(t) \\
\Uparrow \\
P(t'_i) \textit{ for every } i
\end{array}
$$

Then $P(t)$ holds for every SN term t.
Proof.
By contraposition. Suppose that $P(t)$ does not hold for some SN term t.

- then there exists t_1 with $t \longrightarrow_\beta t_1$ such that $P(t_1)$ does not hold,
- then there exists t_2 with $t_1 \longrightarrow_\beta t_2$ such that $P(t_2)$ does not hold,
- ...

Contradiction: we have an infinite sequence of reductions starting from t. $\square$

Neutral terms

A term t is **neutral** when it is not an abstraction:

$$t = t_1\, t_2 \qquad \text{or} \qquad t = x$$

Neutral terms

A term t is **neutral** when it is not an abstraction:

$$t = t_1\, t_2 \qquad\qquad \text{or} \qquad\qquad t = x$$

A neutral term does not interact with its context:

Lemma
Given terms t and u with t neutral, the only possible reductions of $t\,u$ are

- $t\,u \longrightarrow_\beta t'\,u$ *with* $t \longrightarrow_\beta t'$,
- $t\,u \longrightarrow_\beta t\,u'$ *with* $u \longrightarrow_\beta u'$.

Reducibility candidates

Lemma

(CR1) *If* $t \in R_A$ *then* t *is strongly normalizing.*

(CR2) *If* $t \in R_A$ *and* $t \longrightarrow_\beta t'$ *then* $t' \in R_A$.

(CR3) *If* t *is neutral, and for every* t' *such that* $t \longrightarrow_\beta t'$ *we have* $t' \in R_A$, *then* $t \in R_A$.

Reducibility candidates

Lemma

(CR1) *If* $t \in R_A$ *then* t *is strongly normalizing.*

(CR2) *If* $t \in R_A$ *and* $t \longrightarrow_\beta t'$ *then* $t' \in R_A$*.*

(CR3) *If* t *is neutral, and for every* t' *such that* $t \longrightarrow_\beta t'$ *we have* $t' \in R_A$*, then* $t \in R_A$*.*

Proof.
By induction on A. If $A = X$ is a variable then:

(CR1) is true by definition of R_X.

(CR2) If $t \longrightarrow_\beta t'$ and t is SN then t' is SN.

(CR3) If t reduces only in SN terms then it is SN.

Lemma

(CR1) *If $t \in R_A$ then t is strongly normalizing.*

(CR2) *If $t \in R_A$ and $t \longrightarrow_\beta t'$ then $t' \in R_A$.*

(CR3) *If t is neutral, and for every t' such that $t \longrightarrow_\beta t'$ we have $t' \in R_A$, then $t \in R_A$.*

Proof.
Consider the case $A \to B$.

(CR1) Fix $t \in R_{A \to B}$.
We have $x \in R_A$ by (CR3), therefore $t\,x \in R_B$ and is thus SN by (CR1).
Any infinite reduction $t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \dots$ would induce an infinite reduction $t\,x \longrightarrow_\beta t_1\,x \longrightarrow_\beta t_2\,x \longrightarrow_\beta \dots$ with $t\,x \in R_B$, which is impossible.
Thus t is SN.

Reducibility candidates

Lemma

(CR1) *If* $t \in R_A$ *then* t *is strongly normalizing.*

(CR2) *If* $t \in R_A$ *and* $t \longrightarrow_\beta t'$ *then* $t' \in R_A$.

(CR3) *If* t *is neutral, and for every* t' *such that* $t \longrightarrow_\beta t'$ *we have* $t' \in R_A$, *then* $t \in R_A$.

Proof.
Consider the case $A \to B$.

(CR2) Fix $t \in R_{A \to B}$ with $t \longrightarrow_\beta t'$.
Given $u \in R_A$, we have $t\,u \in R_B$ and $t\,u \longrightarrow_\beta t'\,u$, therefore $t'\,u \in R_B$ by (CR2).
Therefore $t' \in R_{A \to B}$ by definition of $R_{A \to B}$.

Lemma

(CR1) *If $t \in R_A$ then t is strongly normalizing.*

(CR2) *If $t \in R_A$ and $t \longrightarrow_\beta t'$ then $t' \in R_A$.*

(CR3) *If t is neutral, and for every t' such that $t \longrightarrow_\beta t'$ we have $t' \in R_A$, then $t \in R_A$.*

Proof.
Consider the case $A \to B$.

(CR3) Fix t neutral satisfying the property.

Given $u \in R_A$, u is SN by (CR1), and we can reason by induction on it.
Since t is neutral the term $t\,u$ reduces either to

- $t'\,u$ with $t \longrightarrow_\beta t'$: we have $t' \in R_{A \to B}$ and thus $t'\,u \in R_B$,
- $t\,u'$ with $u \longrightarrow_\beta u'$: we have $u' \in R_A$ by (CR2), and therefore $t\,u' \in R_B$ by IH.

The term $t\,u$ is neutral and is thus in R_B by (CR3) at type B. □

Reducibility candidates

Lemma
Given a term t and types A and B, if $t[u/x] \in R_B$ for every $u \in R_A$, then $\lambda x.t \in R_{A \to B}$.

Proof.
Since $x \in R_A$ by (CR3) at A, we have $t = t[x/x] \in R_B$ and thus $t \in R_B$ and is thus strongly normalizing by (CR1).

Given $u \in R_A$, we show that $(\lambda x.t)u \in R_B$ by induction on (t, u). The term $(\lambda x.t)u$ can reduce to

Lemma
Given a term t and types A and B, if $t[u/x] \in R_B$ for every $u \in R_A$, then $\lambda x.t \in R_{A \to B}$.

Proof.
Since $x \in R_A$ by (CR3) at A, we have $t = t[x/x] \in R_B$ and thus $t \in R_B$ and is thus strongly normalizing by (CR1).

Given $u \in R_A$, we show that $(\lambda x.t)u \in R_B$ by induction on (t, u). The term $(\lambda x.t)u$ can reduce to

- $t[u/x]$: in R_B by hypothesis,

Reducibility candidates

Lemma
Given a term t and types A and B, if $t[u/x] \in R_B$ for every $u \in R_A$, then $\lambda x.t \in R_{A \to B}$.

Proof.
Since $x \in R_A$ by (CR3) at A, we have $t = t[x/x] \in R_B$ and thus $t \in R_B$ and is thus strongly normalizing by (CR1).

Given $u \in R_A$, we show that $(\lambda x.t)u \in R_B$ by induction on (t, u). The term $(\lambda x.t)u$ can reduce to

- $t[u/x]$: in R_B by hypothesis,
- $(\lambda x.t')u$ with $t \longrightarrow_\beta t'$: in R_B by induction hypothesis,

Reducibility candidates

Lemma
Given a term t and types A and B, if $t[u/x] \in R_B$ for every $u \in R_A$, then $\lambda x.t \in R_{A \to B}$.

Proof.
Since $x \in R_A$ by (CR3) at A, we have $t = t[x/x] \in R_B$ and thus $t \in R_B$ and is thus strongly normalizing by (CR1).

Given $u \in R_A$, we show that $(\lambda x.t)u \in R_B$ by induction on (t, u). The term $(\lambda x.t)u$ can reduce to

- $t[u/x]$: in R_B by hypothesis,
- $(\lambda x.t')u$ with $t \longrightarrow_\beta t'$: in R_B by induction hypothesis,
- $(\lambda x.t)u'$ with $u \longrightarrow_\beta u'$: in R_B by induction hypothesis.

Reducibility candidates

Lemma
Given a term t and types A and B, if $t[u/x] \in R_B$ for every $u \in R_A$, then $\lambda x.t \in R_{A \to B}$.

Proof.
Since $x \in R_A$ by (CR3) at A, we have $t = t[x/x] \in R_B$ and thus $t \in R_B$ and is thus strongly normalizing by (CR1).

Given $u \in R_A$, we show that $(\lambda x.t)u \in R_B$ by induction on (t, u). The term $(\lambda x.t)u$ can reduce to

- $t[u/x]$: in R_B by hypothesis,
- $(\lambda x.t')u$ with $t \longrightarrow_\beta t'$: in R_B by induction hypothesis,
- $(\lambda x.t)u'$ with $u \longrightarrow_\beta u'$: in R_B by induction hypothesis.

The term $(\lambda x.t)u$ is neutral and reduces to terms in R_B.
By (CR3), it belongs to R_B. $\square$

Finally, we would like to show that

Theorem
Given t such that $\Gamma \vdash t : A$ is derivable, we have $t \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

Reducibility candidates

Finally, we would like to show that

Theorem
Given t such that $\Gamma \vdash t : A$ is derivable, we have $t \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{}{\Gamma \vdash x : A}\ (\mathrm{ax})$$

  Then x is neutral and reduces only to terms in R_A (it does not reduce).
  By (CR3) $x \in R_A$.

Reducibility candidates

Finally, we would like to show that

Theorem
Given t such that $\Gamma \vdash t : A$ is derivable, we have $t \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_{\mathrm{E}})$$

  By IH, we have $t \in R_{A \to B}$ and $u \in R_A$.

  Therefore $t\,u \in R_B$ by definition of $R_{A \to B}$.

Finally, we would like to show that

Theorem
Given t such that $\Gamma \vdash t : A$ is derivable, we have $t \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_\mathrm{I})$$

  By IH, we have $t \in R_A$. And... ???? □

Finally, we would like to show that

Theorem
Given t such that $\Gamma \vdash t : A$ is derivable, we have $t \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \ (\to_{\mathrm{I}})$$

  By IH, we have $t \in R_A$. And... ???? □

We actually need a stronger induction hypothesis!
(so that we can use previous lemma)

Reducibility candidates

Proposition
Given t such that $x_1 : A_1, \dots, x_n : A_n \vdash t : A$ is derivable, and for every terms $u_i \in R_{A_i}$, we have $t[\underline{u}/\underline{x}] = t[u_1/x_1, \dots, u_n/x_n] \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

Proposition
Given t such that $x_1 : A_1, \dots, x_n : A_n \vdash t : A$ is derivable, and for every terms $u_i \in R_{A_i}$, we have $t[\underline{u}/\underline{x}] = t[u_1/x_1, \dots, u_n/x_n] \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{}{\Gamma \vdash x_i : A}\ (\mathrm{ax})$$

The $x_i[\underline{u}/\underline{x}] = u_i \in R_{A_i}$.

Proposition
Given t *such that* $x_1 : A_1, \dots, x_n : A_n \vdash t : A$ *is derivable, and for every terms* $u_i \in R_{A_i}$*,* *we have* $t[\underline{u}/\underline{x}] = t[u_1/x_1, \dots, u_n/x_n] \in R_A$*.*

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : B}\ (\to_{\mathsf{E}})$$

By IH, we have $t_1[\underline{u}/\underline{x}] \in R_{A \to B}$ and $t_2[\underline{u}/\underline{x}] \in R_A$.
Therefore $(t_1\ t_2)[\underline{u}/\underline{x}] = (t_1[\underline{u}/\underline{x}])(t_2[\underline{u}/\underline{x}]) \in R_B$ by definition of $R_{A \to B}$.

Proposition
Given t such that $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ is derivable, and for every terms $u_i \in R_{A_i}$, we have $t[\underline{u}/\underline{x}] = t[u_1/x_1, \ldots, u_n/x_n] \in R_A$.

Proof.
By induction on the derivation of $\Gamma \vdash t : A$.

- If the last rule is

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_\mathrm{I})$$

By IH, we have $(t[\underline{u}/\underline{x}])[v/x] = t[\underline{u}/\underline{x}, v/x] \in R_B$ for every $v \in R_A$.
Therefore, $\lambda x.t \in R_{A \to B}$ by previous lemma. $\square$

Theorem
For every t *such that* $\Gamma \vdash t : A$ *is derivable, we have* $t \in R_A$.

Proof.
Suppose $\Gamma = x_1 : A_1, \ldots, x_n : A_n$.

By (CR3), we have $x_i \in R_{A_i}$.

By previous proposition, we have $t = t[x_1/x_1, \ldots, x_n/x_n] \in R_A$. $\square$

Strong normalizablility

Theorem
For every term t such that $\Gamma \vdash t : A$ is derivable, t is strongly normalizable.

Proof.
We have $t \in R_A$, and thus t is SN by (CR1). $\square$