5th Asian-Pacific Summer School on Formal Methods
August 5-10, 2013, Tsinghua University, Beijing, China

Functions

jean-jacques.levy@inria.fr
2013-8-6

http://sts.thss.tsinghua.edu.cn/Coqschool2013

Notes adapted from
Assia Mahboubi
(coq school 2010, Paris) and
Benjamin Pierce (software
foundations course, UPenn)

Functions in Coq

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

Plan

- notation for functions in Coq
- λ-notation
- λ-calculus
- enumerated types
- pattern-matching on constructors

definitions (1/3)

three equivalent definitions:

```
Definition plusOne (x: nat) : nat := x + 1.
Check plusOne.

Definition plusOne := fun (x: nat) => x + 1.
Check plusOne.

Definition plusOne := fun x => x + 1.
Check plusOne.

Compute (fun x:nat => x + 1) 3.
```

higher-order definitions:

```
Definition plusTwo (x: nat) : nat := x + 2.

Definition twice := fun f => fun (x:nat) => f (f x).

Compute twice plusTwo 3.
```

lambda-terms (2/3)

- Coq tries to guess the type, but could fail.
`(type inference)`

- but always possible to give explicit types.

- Types can be higher-order
`(see later with polymorphic functions)`

- Types can also depend on values
`(see later the constructor cases)`

Recap

- Coq commands / keywords:

  - `Definition` for functions definitions
  - `Check` to show types
  - `Compute` to show values

lambda-terms (3/3)

- Coq treats with an extention of the λ-calculus with inductive data types. It's a small programming language.

- the typed λ-calculus is used as a trick to make a correspondence between proofs and λ-terms and propositions and types for constructive logics (see other lectures).
`(Curry-Howard correspondence)`

constructive logic

constructive logic

- An example of a non constructive proof:

Theorem
There exists 2 irrational numbers a and b such that a^b is rational.

Proof
We know that $\sqrt{2}$ is not rational. Take $a = b = \sqrt{2}$.

- a^b is rational. OK!

- a^b is irrational. Then let $c = a^b$.
Then $c^b = (a^b)^b = a^{b \times b} = a^2 = 2$. Done!

QED

constructive logic

- Coq is constructive logic

Propositions always exist with their (witness) proofs.
$h : P$ in environment means h is witness proof of P.

constructive logic

- An example of a non constructive proof:

Theorem
There exists 2 irrational numbers a and b such that a^b is rational.

Proof
We know that $\sqrt{2}$ is not rational. Take $a = b = \sqrt{2}$.

- a^b is rational. OK!

- a^b is irrational. Then let $c = a^b$.
Then $c^b = (a^b)^b = a^{b \times b} = a^2 = 2$. Done!

QED

- Coq is constructive logic

Propositions always exist with their (witness) proofs.
$h : P$ in environment means h is witness proof of P.

λ-notation

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

Functional calculus (1/4)

$(\lambda x.\, x+1)3 \longrightarrow 3+1 \longrightarrow 4$

$(\lambda x.\, 2*x+2)4 \longrightarrow 2*4+2 \longrightarrow 8+2 \longrightarrow 10$

$(\lambda f.f3)(\lambda x.\, x+2) \longrightarrow (\lambda x.\, x+2)3 \longrightarrow 3+2 \longrightarrow 5$

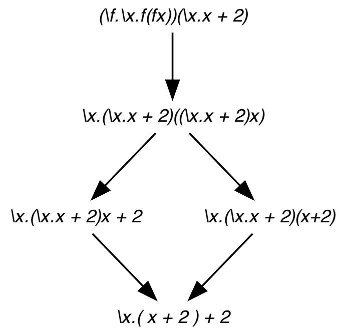
$(\lambda x.\lambda y.x+y)3\ 2 =$

$((\lambda x.\lambda y.x+y)3)2 \longrightarrow (\lambda y.3+y)2 \longrightarrow 3+2 \longrightarrow 5$

$(\lambda f.\lambda x.f(f\,x))(\lambda x.x+2) \longrightarrow \ldots$

Functional calculus (2/4)

$(\lambda f.\lambda x.f(f\,x))(\lambda x.\, x+2) \longrightarrow \ldots$

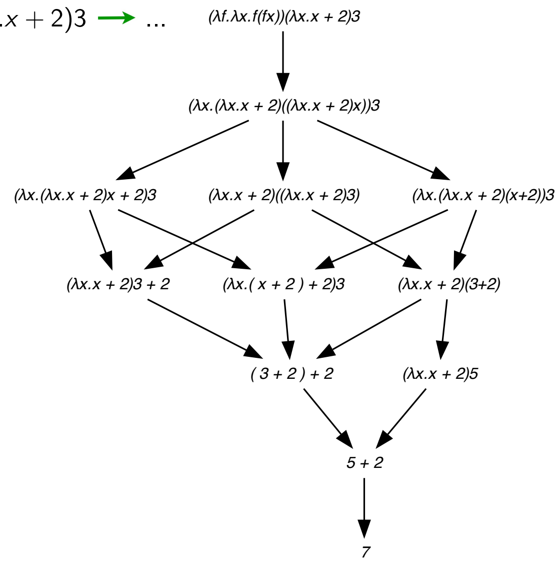


Functional calculus (2/4)

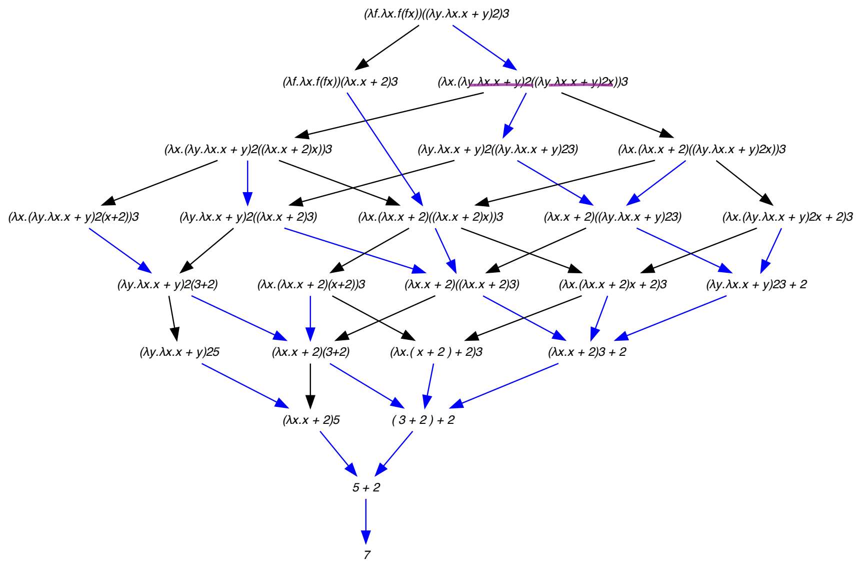
$(\lambda f.\lambda x.f(f\,x))(\lambda x.\, x+2) \longrightarrow \ldots$

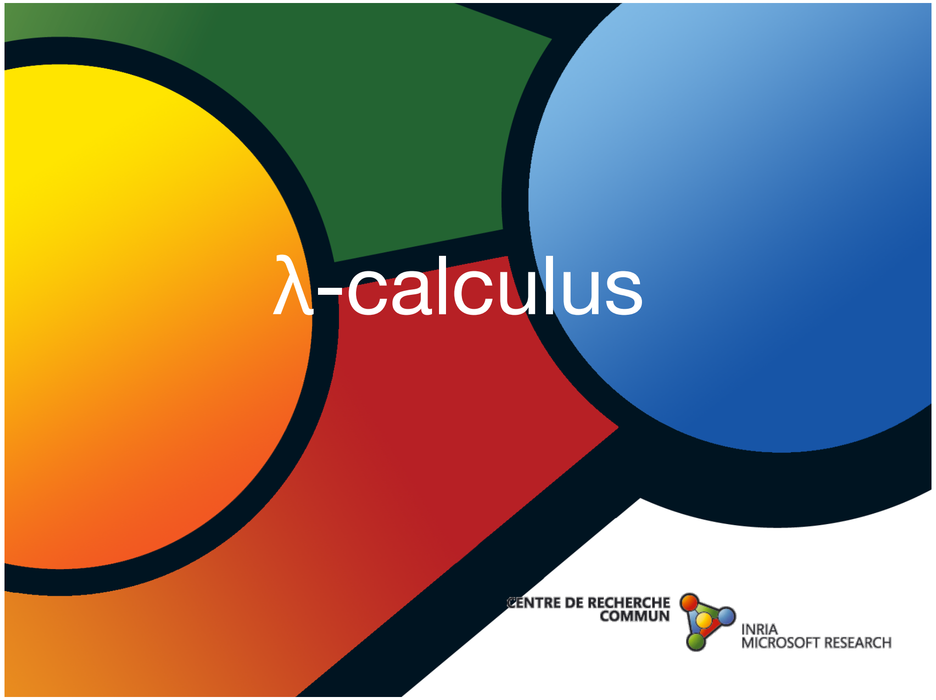
Functional calculus (3/4)

$(\lambda f.\lambda x.f(f\,x))(\lambda x.x+2)3 \longrightarrow \ldots$



$$(\lambda f.\lambda x.f(f\,x))((\lambda y.\lambda x.x+y)2)3 \longrightarrow \ldots$$





Functional calculus (4/4)

- computing with functions may be long and complex

- but yield a unique result
(Church-Rosser property)

Thought of Tuesday 2013-8-6

- computer science = programs = texts in ASCII

```
#define _ -F<OO || --F-OO--;
int F=OO,OO=OO;
main(){F_OO();printf("%1.3f\n", 4.*-F/OO/OO);}F_OO()
{
            _-_-_-_
       _-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_
       _-_-_-_-_-_-_-_-_
            _-_-_-_
 }
```

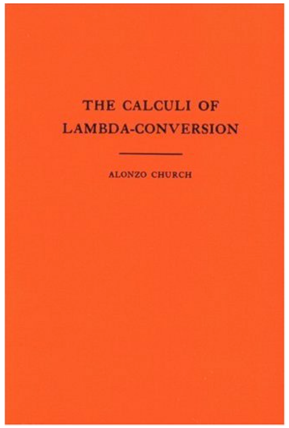
- mathematics
  = greek letters
  + symbols

(λ η σ ρ π α β γ δ Δ -
+ / ⊆ ∩ ⊦ ⊧)

Pure lambda-calculus

- lambda-terms

M, N, P	$::=$	$x, y, z, \ldots$	(variables)
	$\mid$	$\lambda x.M$	(M as function of x)
	$\mid$	$M\,(\,N\,)$	(M applied to N)



- Computations "reductions"

$$(\lambda x.M)(N) \longrightarrow M\{x := N\}$$

Examples of reductions (1/2)

- Examples

$$(\lambda x.x)N \longrightarrow N$$

$$(\lambda f.f\,N)(\lambda x.x) \longrightarrow (\lambda x.x)N \longrightarrow N$$

$$(\lambda x.x\,N)(\lambda y.y) \longrightarrow (\lambda y.y)N \longrightarrow N$$ (name of bound variable is meaningless)

$$(\lambda x.\,x\,x)(\lambda x.xN) \longrightarrow (\lambda x.xN)(\lambda x.xN) \longrightarrow (\lambda x.xN)N \longrightarrow NN$$

$$(\lambda x.x)(\lambda x.x) \longrightarrow \lambda x.x$$

Let $I = \lambda x.x$, we have $I(x) = x$ for all x.
Therefore $I(I) = I$. [Church 41]

Examples of reductions (2/2)

- Examples

$$(\lambda x.\,x\,x)(\lambda x.xN) \longrightarrow (\lambda x.xN)(\lambda x.xN) \longrightarrow (\lambda x.xN)N \longrightarrow NN$$

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \longrightarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \longrightarrow \cdots$$

- Possible to loop inside applications of functions ...

$$Y_f = (\lambda x.f(xx))(\lambda x.f(xx)) \longrightarrow f((\lambda x.f(xx))(\lambda x.f(xx))) = f(Y_f)$$

$$f(Y_f) \longrightarrow f(f(Y_f)) \longrightarrow \cdots \longrightarrow f^n(Y_f) \longrightarrow \cdots$$

- Every computable function can be computed by a λ-term

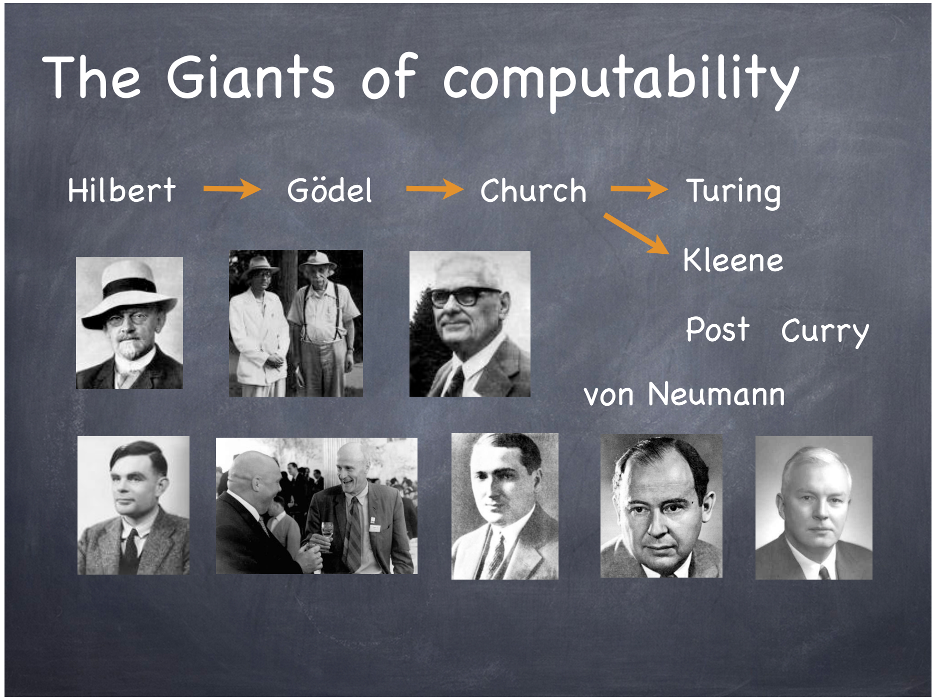
→ Church's thesis. [Church 41]

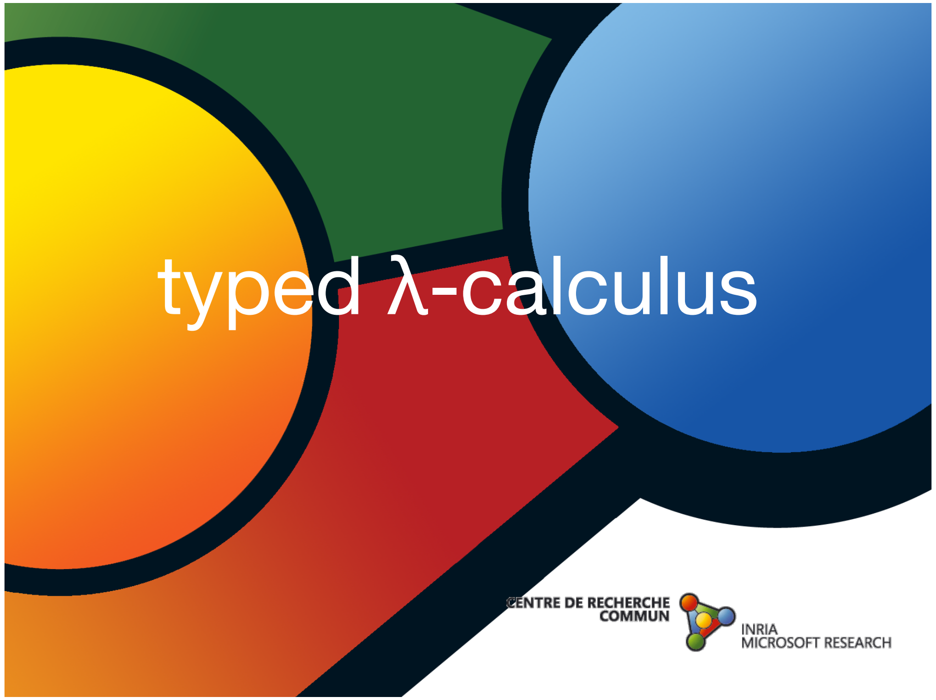
Fathers of computability

Alonzo Church

Stephen Kleene

CENTRE DE RECHERCHE COMMUN INRIA MICROSOFT RESEARCH





Typed lambda-calculus (1/5)

- In Coq, all λ-terms are typed
- In Coq, following λ-terms are typable

$(\lambda x.\, x+1)3 \longrightarrow 3+1 \longrightarrow 4$

$(\lambda x.\, 2*x+2)4 \longrightarrow 2*4+2 \longrightarrow 8+2 \longrightarrow 10$

$(\lambda f.f3)(\lambda x.\, x+2) \longrightarrow (\lambda x.\, x+2)3 \longrightarrow 3+2 \longrightarrow 5$

$(\lambda x.\lambda y.x+y)3\;2 =$

$((\lambda x.\lambda y.x+y)3)2 \longrightarrow (\lambda y.3+y)2 \longrightarrow (\lambda y.3+y)2 \longrightarrow 3+2 \longrightarrow 5$

$(\lambda f.\lambda x.f(f\,x))(\lambda x.x+2) \longrightarrow \ldots$

these terms are allowed

Typed lambda-calculus (2/5)

- In Coq, all λ-terms have only finite reductions
`(strong normalization property)`
- In Coq, all λ-terms have a (unique) normal form.
- In Coq, the following λ-terms are not typable

$(\lambda x.\, x\,x)(\lambda x.\, x\,x)$ $\qquad 2+(\lambda x.\, x+1)$ $\qquad 2(3)$

$(\lambda\, \mathtt{Fact}\,.\,\mathtt{Fact}(3))$

$(\,(\lambda Y.Y(\lambda f.\lambda x.\ \mathtt{ifz}\ x\ \mathtt{then}\ 1\ \mathtt{else}\ x*f(x-1)))$

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

these terms are not allowed

Typed lambda-calculus (2/5)

- In Coq, all λ-terms have only finite reductions
(strong normalization property)
- In Coq, all λ-terms have a (unique) normal form.

Typed lambda-calculus (3/5)

- The Coq laws for typing terms are quite complex
[Coquand-Huet 1985]
- They are almost the following (1st-order) rules:

Basic types: $\mathcal{N}$ (nat), $\mathcal{B}$ (bool), $\mathcal{Z}$ (int), ...

If M has type β when x has type α, then $(\lambda x.M)$ has type $\alpha \rightarrow \beta$

If M has type $\alpha \rightarrow \beta$ and if N has type α, then $M(N)$ has type β

Example

$1 : \texttt{nat}$

$x : \texttt{nat}$ implies $x + 1 : \texttt{nat}$

$(\lambda x.\, x + 1) : \texttt{nat} \rightarrow \texttt{nat}$

$3 : \texttt{nat}$

$(\lambda x.\, x + 1)3 : \texttt{nat}$

Typed lambda-calculus (4/5)

Example

$$x : \texttt{nat} \vdash x : \texttt{nat}$$

$$\frac{x : \texttt{nat} \vdash x : \texttt{nat} \qquad \vdash 1 : \texttt{nat}}{x : \texttt{nat} \vdash x + 1 : \texttt{nat}}$$

$$\frac{x : \texttt{nat} \vdash x + 1 : \texttt{nat}}{\vdash (\lambda x.\, x + 1) : \texttt{nat} \rightarrow \texttt{nat}}$$

$$\frac{\vdash (\lambda x.\, x + 1) : \texttt{nat} \rightarrow \texttt{nat} \qquad \vdash 3 : \texttt{nat}}{\vdash (\lambda x.\, x + 1)3 : \texttt{nat}}$$

Exercise Write it as a proof tree [aka Monin's lectures].

Typed lambda-calculus (5/5)

Example with currying and function as result

$$\frac{x : \texttt{nat} \vdash x : \texttt{nat}}{x : \texttt{nat}, y : \texttt{nat} \vdash x : \texttt{nat}} \qquad \frac{y : \texttt{nat} \vdash y : \texttt{nat}}{x : \texttt{nat}, y : \texttt{nat} \vdash y : \texttt{nat}}$$

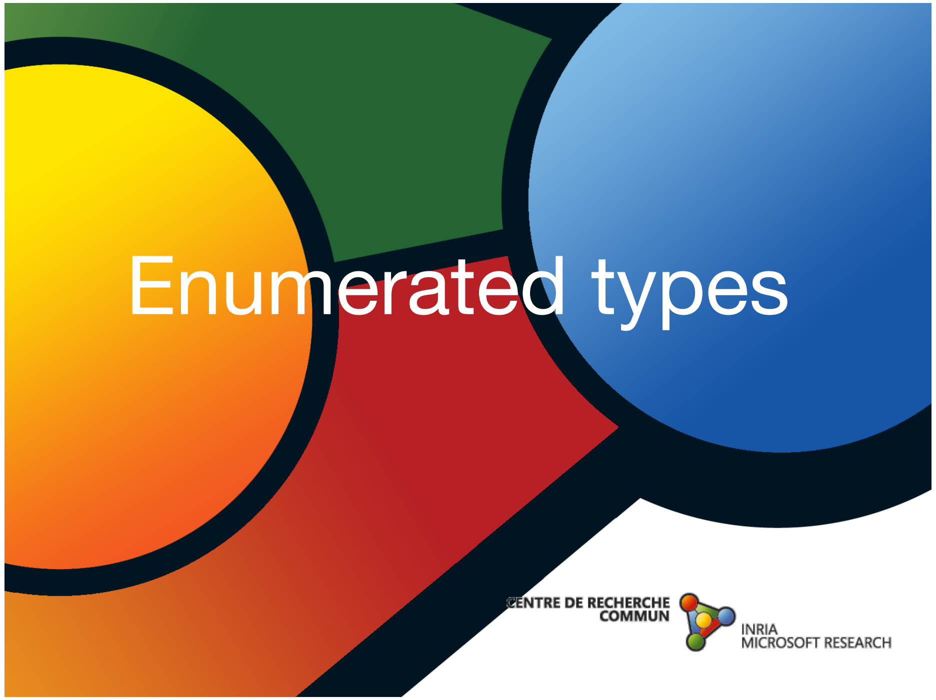
$$\frac{x : \texttt{nat}, y : \texttt{nat} \vdash x : \texttt{nat} \qquad x : \texttt{nat}, y : \texttt{nat} \vdash y : \texttt{nat}}{x : \texttt{nat}, y : \texttt{nat} \vdash x + y : \texttt{nat}}$$

$$\frac{x : \texttt{nat}, y : \texttt{nat} \vdash x + y : \texttt{nat}}{x : \texttt{nat} \vdash (\lambda y.x + y) : \texttt{nat} \rightarrow \texttt{nat}}$$

$$\frac{x : \texttt{nat} \vdash (\lambda y.x + y) : \texttt{nat} \rightarrow \texttt{nat}}{\vdash (\lambda x.\lambda y.x + y) : \texttt{nat} \rightarrow \texttt{nat} \rightarrow \texttt{nat}}$$

$$\frac{\vdash (\lambda x.\lambda y.x + y) : \texttt{nat} \rightarrow \texttt{nat} \rightarrow \texttt{nat} \qquad \vdash 2 : \texttt{nat}}{\vdash ((\lambda x.\lambda y.x + y)2) : \texttt{nat} \rightarrow \texttt{nat}}$$

$$\frac{\vdash ((\lambda x.\lambda y.x + y)2) : \texttt{nat} \rightarrow \texttt{nat} \qquad \vdash 3 : \texttt{nat}}{\vdash ((\lambda x.\lambda y.x + y)2\ 3) : \texttt{nat}}$$



Enumeratives types (2/5)

Enumerated types are types which list and name exhaustively their inhabitants.

A new enumerated type:

```
Inductive day : Type :=
| monday | tuesday | wednesday |
| thursday | friday | saturday | sunday : day.
```

Enumeratives types (1/5)

Enumerated types are types which list and name exhaustively their inhabitants.

```
Inductive bool : Set := true : bool | false : bool.
```

Set is deprecated. Now use Type.

```
Inductive color : Type := black : color | white : color.
```

Enumeratives types (2/5)

Enumerated types are types which list and name exhaustively their inhabitants.

A new enumerated type:

```
Inductive day : Type :=
| monday | tuesday | wednesday |
| thursday | friday | saturday | sunday : day.
```

```
Check tuesday.
```

tuesday : day

Labels refer to distinct elements.

Enumeratives types (3/5)

Inspect the enumerated type inhabitants and assign values:

```
Definition negb (b : bool) :=
  match b with true => false | false => true end.
```


Enumeratives types (3/5)

Inspect the enumerated type inhabitants and assign values:

```
Definition negb (b : bool) :=
  match b with true => false | false => true end.
```


```
Definition next_weekday (d:day) : day :=
  match d with
  | monday => tuesday       | tuesday => wednesday
  | wednesday => thursday   | thursday => friday
  | friday | saturday | sunday => monday end.

Eval compute in (next_weekday friday).
    = monday
    : day
```


Enumeratives types (3/5)

Inspect the enumerated type inhabitants and assign values:

```
Definition negb (b : bool) :=
  match b with true => false | false => true end.

Definition next_weekday (d:day) : day :=
  match d with
  | monday => tuesday       | tuesday => wednesday
  | wednesday => thursday   | thursday => friday
  | friday | saturday | sunday => monday end.
```


Recap

- Coq commands / keywords:
  - `Definition` for functions definitions
  - `Check` to show types
  - `Compute` to show values
  - `Eval compute in` to show values
  - `Inductive` to define a new data type
  - `Type` set of all types
  - `match ... with` for case analysis on constructors

Enumeratives types (4/5)

```
Definition andb (b1:bool) (b2:bool) : bool :=
  match b1 with true => b2 | false => false  end.

Definition orb (b1:bool) (b2:bool) : bool :=
  match b1 with true => true | false => b2  end.
```

Recap

- Coq commands / keywords:
  - `Definition` for functions definitions
  - `Check` to show types
  - `Compute` to show values
  - `Eval compute in` to show values
  - `Inductive` to define a new data type
  - `match ... with` for case analysis on constructors
  - `Type` set of all types
  - `simpl` to compute normal form
  - `reflexivity` to conclude with trivial equality

Enumeratives types (4/5)

```
Definition andb (b1:bool) (b2:bool) : bool :=
  match b1 with true => b2 | false => false  end.

Definition orb (b1:bool) (b2:bool) : bool :=
  match b1 with true => true | false => b2  end.

Example test_orb1: (orb true false) = true.
  orb true false = true
Proof.
simpl.
  true = true
reflexivity.
Qed.
test_orb1 is defined
```

Enumeratives types (5/5)

Exercise Give definitions of predicates `work_day` and `weekend_day`.

Exercise Give definitions of function `black_if_workday` and white for weekends.