newsflash

Simon ! mice !

inductive predicates

logical verification

week 5

2004 10 06

recap in abstracto

what is an inductive type?

'minimal set such that **constructors** exist'

‘free’

all terms built with only constructors are different

inductive type = set of all ‘terms’

what do you get?

- a new type

  `nat`

- constructor functions

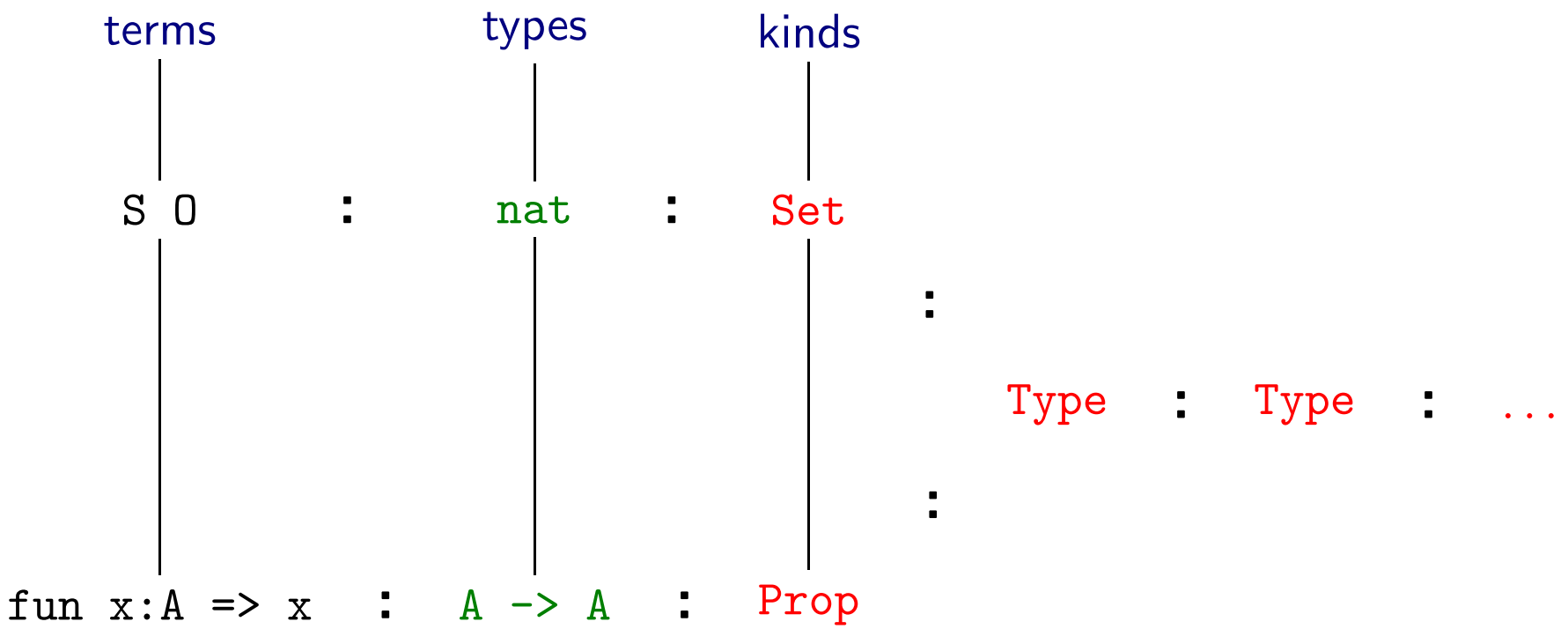
  `O` and `S`

- `match` + ι-reduction

  `match ... with O => ... | S m => ... end`

- induction principle

universes



inductive types can be defined in each universe

```
Inductive ... : Set :=
  ...

Inductive ... : Prop :=
  ...

Inductive ... : Type :=
  ...
```

inductive types for datatypes

booleans

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

enumeration types

```
Inductive color : Set :=
  | black : color
  | white : color
  | red : color
  | yellow : color
  | green : color
  | blue : color
  | brown : color
  | orange : color
  | purple : color
  | pink : color
  | gray : color.
```

singleton type

```
Inductive unit : Set :=
    tt : unit.
```

empty type

```
Inductive Empty_set : Set :=
    .
```

natural numbers

```
Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.
```

linear lists of natural numbers

```
Inductive natlist : Set :=
  | nil : natlist
  | cons : nat -> natlist -> natlist.
```

polymorphic linear lists

```
Inductive list (A : Set) : Set :=
  | nil : list A
  | cons : A -> list A -> list A.
```

unlabeled binary trees

```
Inductive bintree : Set :=
  | leaf : bintree
  | node : bintree -> bintree -> bintree.
```

polymorphic labeled binary trees

```
Inductive bintree (A B : Set) : Set :=
  | leaf : B -> bintree A B
  | node : A -> bintree A B -> bintree A B -> bintree A B.
```

option type

```
Inductive option (A : Set) : Set :=
  | Some : A -> option A
  | None : option A.
```

disjoint union of two types

```
Inductive sum (A B : Set) : Set :=
  | inl : A -> sum A B
  | inr : B -> sum A B.
```

cartesian product of two types

```
Inductive prod (A B : Set) : Set :=
    pair : A -> B -> prod A B.
```

Curry-Howard-de Bruijn isomorphism

proposition is true

$\Updownarrow$

type is inhabited

truth

```
Inductive True : Prop :=
    I : True.
```

falsum

```
Inductive False : Prop :=
    .
```

conjunction

```
Inductive and (A B : Prop) : Prop :=
    conj : A -> B -> and A B.

Notation "A /\ B" := (and A B) : type_scope.
```

disjunction

```
Inductive or (A B : Prop) : Prop :=
  | or_introl : A -> or A B
  | or_intror : B -> or A B
```

datatypes in Set $\longleftrightarrow$ logic in Prop

`unit`	$\longleftrightarrow$	`True`
`Empty_set`	$\longleftrightarrow$	`False`
`prod`	$\longleftrightarrow$	`and`
`sum`	$\longleftrightarrow$	`or`

Curry-Howard-de Bruijn isomorphism

predicate is true

$\Updownarrow$

inductive type is inhabited

even natural numbers

```
Inductive even : nat -> Prop :=
  | even_O : even O
  | even_S_S : forall n, even n -> even (S (S n)).
```

even and odd natural numbers

```
Inductive even : nat -> Prop :=
  | even_O : even O
  | even_S : forall n, odd n -> even (S n)
with odd : nat -> Prop :=
    odd_S : forall n, even n -> odd (S n).
```

$n \leq m$

```
Inductive le (n:nat) : nat -> Prop :=
  | le_n : le n n
  | le_S : forall m:nat, le n m -> le n (S m).
```

sorted lists of natural numbers

```
Inductive Sorted : natlist -> Prop :=
  | Sorted_nil : Sorted nil
  | Sorted_one : forall n : nat, Sorted (cons n nil)
  | Sorted_cons :
      forall (l : natlist) (n m : nat),
        Sorted (cons m l) -> le n m ->
          Sorted (cons n (cons m l)).
```

Leibniz equality

```
Inductive eq (A : Set) (x : A) : A -> Prop :=
    refl_equal : eq A x x.
```

applying the induction principle

- `elim`
- `induction`

inversion

```
Lemma not_even_1 : ~(even 1).
```

- `inversion`

what does inversion do?

abstract explanation

elimination proves universally quantified properties

$$\forall x_1, \ldots, x_l.\, \mathtt{I}(x_1, \ldots, x_l) \Rightarrow P(x_1, \ldots, x_l)$$

- what to do with $\mathtt{I}(t_1, \ldots, t_l) \Rightarrow P(t_1, \ldots, t_l)$ when t_i are not variables?
- generalize to $\forall x_1, \ldots, x_l.\, \mathtt{I}(x_1, \ldots, x_l) \Rightarrow P(x_1, \ldots, x_l)$
- what to do if the generalization does not hold?
- find another one which works:

$$\forall x_1, \ldots, x_l.\, \mathtt{I}(x_1, \ldots, x_l) \Rightarrow \underbrace{(x_1, \ldots, x_l) = (t_1, \ldots, t_l) \Rightarrow P(t_1, \ldots, t_l)}$$

example

constructors

$$\texttt{even\_O} : \texttt{even}\,(0) \qquad \texttt{even\_S\_S} : \forall n.\,\texttt{even}\,(n) \Rightarrow \texttt{even}\,(n+2)$$

induction principe

$$\frac{P(0) \qquad \forall n.\,\texttt{even}\,(n) \Rightarrow P(n) \Rightarrow P(n+2)}{\forall n.\,\texttt{even}\,(n) \Rightarrow P(n)}$$

how to prove $\texttt{even}\,(1) \Rightarrow \bot$?

take: $P(n) = n = 1 \Rightarrow \bot$