

logical verification lecture 5

2011 04 15

inductive definitions

overview

- logic:
minimal, plus falsum, intuitionistic, classical
- lambda calculus:
simply typed
- Curry-Howard-de Bruijn isomorphism:
 $ML \sim \lambda^{\rightarrow}$
- inductive definitions:
for data-types (in Set) and for predicates (in Prop)

nat and natlist: definitions

a nat is either zero or successor of some nat

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

a natlist is either nil or cons of some nat and some natlist

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

nat and natlist: functions

pattern matching considers two cases for a nat

```
Fixpoint plus (n m : nat) {struct n} : nat :=  
  match n with  
  | 0 => m  
  | (S p) => S (plus p m)  
end .
```

pattern matching considers two cases for a natlist

```
Fixpoint append (k l : natlist) {struct k} : natlist :=  
  match k with  
  | nil => l  
  | (cons h t) => cons h (append t l)  
end .
```

nat and natlist: induction principles

induction on nat

```
nat_ind :  
  forall P : nat -> Prop,  
  P 0 ->  
  (forall n : nat, P n -> P (S n)) ->  
  forall n : nat, P n
```

induction on natlist

```
natlist_ind  
  : forall P : natlist -> Prop,  
  P nil ->  
  (forall (n : nat) (n0 : natlist), P n0 -> P (cons n n0)) ->  
  forall n : natlist, P n
```

nat and natlist: tactic induction

for some n in `nat`

induction `n` is roughly apply `nat_ind`

for some l in `natlist`

induction `l` is roughly apply `natlist_ind`

nat and natlist: proofs

NB recursion of plus in the first argument

Lemma plus_n_0 : forall n : nat, plus n 0 = n.

NB recursion of append in the first argument

Lemma append_l_nil : forall l : natlist , append l nil = l.

NB: sometimes the proof makes you reconsider your definitions

tactic discriminate

if a hypothesis has the form

$H : 0 = S 0$

use the tactic

`discriminate H.`

in general: for hypotheses $s = t$ where s and t start with different constructors

tactic injection

if a hypothesis has the form

$$H : S\ n = S\ m$$

use the tactic

`injection H.`

in general: for hypothesis $s = t$ where s and t start with the same constructors

even: definition

```
Inductive even : nat -> Prop :=  
| even0   : even 0  
| evenSS  : forall n:nat ,  
             even n -> even (S (S n)) .
```

even: induction principle

```
even_ind :  
  forall P : nat -> Prop,  
  P 0 ->  
  (forall n : nat, even n -> P n -> P (S (S n))) ->  
  forall n : nat, even n -> P n
```

even: tactic inversion

the tactic inversion (very) roughly does apply even ind

tactics inversion

`inversion H.`

`simple inversion H.`

`inversion_clear H.`

le: family of inductive predicates

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

le_ind

```
: forall (n : nat) (P : nat -> Prop),  
P n ->  
(forall m : nat, le n m -> P m -> P (S m)) ->  
forall n0 : nat, le n n0 -> P n0
```

truth: inductive definition

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

falsity: inductive definition

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

```
·
```

falsity: induction principle

```
False_ind :  
  forall P : Prop, False -> P
```

gives the elimination rule via the tactics

- `elim h`
- `elimtype False`
- `apply False_ind`

conjunction: inductive definition

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

gives the introduction rule via the tactics

- apply conj
- split

conjunction: induction principle

```
and_ind: forall A B P : Prop,  
  (A -> B -> P) -> A /\ B -> P
```

gives the elimination rule via the tactics

- `elim h`
- `apply and_ind`

disjunction: inductive definition

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

gives the introduction rule via the tactics

- left.
- right.

disjunction: induction principle

gives the elimination rule via the tactic

```
or_ind: forall A B P : Prop,  
  (A -> P) -> (B -> P) -> A \/ B -> P
```

- `elim h.`

tactic elim

elim H can be used for every hypothesis H in some inductive type

Prop versus bool

```
      I : True  : Prop
true  : bool  : Set
```

True : inductive type

bool : inductive type

true : not a type

it is all lambda-calculus

application

$$\frac{\Gamma \vdash H : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash (H a) : B}$$

dependent application

$$\frac{\Gamma \vdash H : \forall x : A, B \quad \Gamma \vdash a : A}{\Gamma \vdash (H a) : B[x := a]}$$