5th Asian-Pacific Summer School on Formal Methods

August 5-10, 2013, Tsinghua University, Beijing, China

Inductive predicates

Jean-François Monin

2013-08-08

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

Predicates

Given a type T, how to select elements of T satisfying some property p ?

Let us consider a predicate

$$p : T \to \textbf{Prop}$$

Let u be an inhabitant of T.

We say that u is selected when

- we can prove $p\ u$
- there is a proof tree in $p\ u$
- $p\ u$ is inhabited

Otherwise u is not selected

Predicates, relations

Given a type T, how to select elements of T satisfying some property ?

General type to be considered: $T \to \textbf{Prop}$

Sometimes, the answer can be computed
Then we can also use: $T \to \textbf{bool}$
Note that $T \to \textbf{color}$ works as well

More generally: inductive relations

Given types T_1, T_2, ... T_n, how to simultaneously select elements of T_1, T_2, ... T_n, satisfying some property ?

General type to be considered: $T_1 \to T_2 \to \dots T_n \to \textbf{Prop}$

To some extent, we can also use:
$T_1 \to T_2 \to \dots T_n \to \textbf{bool}$

What does it mean, not to be selected ?

We say that u is selected when

- there is no proof tree in $p\ u$
- $p\ u$ is not inhabited

So, what we need here is a special type (a proposition) which is **empty**

How to make it?

Take any inductive type with **zero** constructor

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

This makes sense because of **strong normalization**

Strong normalization

Let T be an inductive type

Let u an inhabitant of T

Then

- u can be reduced by computation to a unique normal form u_0
- (in the empty environment),
  the normal form u_0 starts with a constructor of T

Therefore, there is no inhabitant at all in `False`

Using a data type (bool, color,...)

Take a decision function generally written $\texttt{p\_b} : T \to \texttt{bool}$
such that

forall $x : T$, $\texttt{p\_b}\ x = \texttt{true}$ is equivalent to $p\ x$

No magic in bool

Take a decision function $\texttt{p\_c} : T \to \texttt{color}$
such that

forall $x : T$, $\texttt{p\_b}\ x = \texttt{white}$ is equivalent to $p\ x$

Inductive predicates on days

How to select elements of `day` ?

Even numbers

```
Inductive even : nat -> Prop :=
  | E0 : even 0
  | E2:  forall n:nat, even n -> even (S (S n)).
```

We expect the following induction principle:

$$\frac{P\ 0 \qquad \forall n, even\ n \to P\ n \to P\ (S\ (S\ n))}{\forall n, even\ n \to P\ n}$$

Lists of consecutive even numbers

```
Inductive natlist: Set :=
  | E : natlist
  | C :  nat -> natlist -> natlist.
```

$$\frac{P\ E \qquad \forall n \forall l, P\ l \to P\ (C\ n\ l)}{\forall l, P\ l}$$

```
Inductive evl : nat -> Set :=
  | E0 : evl 0
  | E2:  forall n:nat, evl n -> evl (S (S n)).
```

$$\frac{P\ E0 \qquad \forall n \forall l, P\ l \to P\ (E2\ n\ l)}{\forall l, P\ l}$$

$$\frac{P\ 0\ E0 \qquad \forall n \forall l, P\ n\ l \to P\ (S\ (S\ n))\ (E2\ n\ l)}{\forall n l, P\ n\ l}$$

Lists of consecutive even numbers (cont'd)

```
Inductive evl : nat -> Set :=
  | E0 : evl 0
  | E2:  forall n:nat, evl n -> evl (S (S n)).
```

$$\frac{P\ 0\ E0 \qquad \forall n \forall l, P\ n\ l \to P\ (S\ (S\ n))\ (E2\ n\ l)}{\forall n l, P\ n\ l}$$

Take for P a predicate which does not depend on its second argument: $P\ n\ l \stackrel{\text{def}}{=} Q\ n$

$$\frac{Q\ 0 \qquad \forall n\, \forall (l : evl\ n), Q\ n \to Q\ (S\ (S\ n))}{\forall n (l : evl\ n), Q\ n}$$

$$\frac{Q\ 0 \qquad \forall n, evl\ n \to Q\ n \to Q\ (S\ (S\ n))}{\forall n, evl\ n \to Q\ n}$$

Now, evl reads just *even*

Functional interpretation

```
Inductive list : Set :=
  | E : list
  | C :  nat -> list -> list.
```

$$\frac{P\ E \qquad \forall n \forall l, P\ l \to P(C\ n\ l)}{\forall l, P\ l}$$

Lists of consecutive even numbers
typed according to the value of the expected next head

```
Inductive evl : nat -> Set :=
  | E0 : evl 0
  | E2:  forall n:nat, evl n -> evl (S (S n)).
```

$$\frac{P\ E0 \qquad \forall n \forall l, P\ l \to P(E2\ n\ l)}{\forall l, P\ l}$$

$$\frac{P\ 0\ E0 \qquad \forall n \forall l, P\ n\ l \to P\ (S\ (S\ n))\ (E2\ n\ l)}{\forall n l, P\ n\ l}$$

Booleans and inductively defined predicates

```
Fixpoint evenb (n:nat) : bool :=
  match n with
  | O         => true
  | S O       => false
  | S (S n') => evenb n'
  end.
```

```
Inductive even : nat -> Prop :=
  | E0 : even O
  | E2 : ∀ n, even n -> even (S (S n)).
```

```
Theorem even_evenb : ∀ n, even n -> evenb n = true.
```

By induction on the structure of the proof of even n

```
Theorem evenb_even : ∀ n, evenb n = true -> even n.
```

By induction on *n*

Booleans and inductively defined predicates

```
Theorem even_evenb :
  ∀ n, even n -> evenb n = true.
```

By induction on the structure of the proof of even n
Don't have to bother about odd numbers

```
Theorem evenb_even :
   ∀ n, evenb n = true -> even n.
```

By induction on *n*: need for strengthening and discrimination.

Inversion
Issue: getting the possible ways of constructing a hypothesis
Easier for `evenb` than for `even`

This issue cannot be avoided for non-deterministic relations

Inductive relation: less or equal on nat

From standard library

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

Inductive definitions / function to bool

- Inductive definitions are more flexible, easier to define
- With inductive definitions, we only care with the positive cases
- Inductive hypotheses may require heavy steps called inversions
- Functional definitions allow reasoning steps by computation, which is powerful and convenient
- In particular, "inversion is for free" with functional definitions
- Important: a functional definition can be used for tests in a program

`if e then A else B` is a shorthand for

```
match e with
| true => A
| false => B
end
```