Obliczenia i wnioskowanie w systemie Coq

Małgorzata Biernacka

Wykład 6
09.04.2013

Predykat le

Relacja <= : nat -> nat -> Prop

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

Check le_ind.
> le_ind:
     forall (n : nat) (P : nat -> Prop),
     P n ->
     (forall m : nat, n <= m -> P m -> P (S m)) ->
     forall n0 : nat, n <= n0 -> P n0
```

Indukcyjne typy zależne

- typy, które mogą zależeć od termów-obiektów (parametryczne lub indeksowane)
- pozwalają lepiej wyspecyfikować własności obiektów
- alternatywnie: silna specyfikacja (typy zależne) albo słaba specyfikacja + predykaty na termach w typie

  przykłady:
  - typ tablic o rozmiarze n
  - typ drzew binarnych o wysokości h
  - typ reprezentacji lambda termów z typami prostymi à la Church

Indukcyjne typy parametryczne

- przykład: listy liczb naturalnych, których wszystkie elementy są większe od pewnej liczby n
- typ list zależnych od n, których konstruktor cons można zapisać regułą

$$\frac{\texttt{k>n} \quad \vdash \texttt{l : list n}}{\vdash \texttt{cons k l : list n}}$$

```
Inductive llist (n:nat) : Set :=
| lnil : llist n
| lcons : forall k:nat, k > n -> llist n -> llist n.

Check llist_ind.
> llist_ind
 : forall (n : nat) (P : llist n -> Prop),
   P (lnil n) ->
   (forall (k : nat) (l : k > n) (l0 : llist n),
    P l0 -> P (lcons n k l l0)) -> forall l : llist n, P l
```

Indukcyjne typy indeksowane

- przykład: typ drzew binarnych o wysokości h, których oba poddrzewa mają tę samą wysokość
- typ drzew zależnych od zmiennego n, których konstruktor `node` można zapisać regułą

$$\frac{\vdash \texttt{l1 : btree m} \quad \vdash \texttt{l2 : btree m}}{\vdash \texttt{node x l1 l2 : btree (S m)}}$$

```
Inductive btree : nat -> Set :=
| leaf : nat -> btree 0
| node : forall m, nat -> btree m -> btree m -> btree (S m).
```

Indukcyjne typy indeksowane, c.d.

```
Check btree_ind.
> btree_ind
 : forall P : forall n : nat, btree n -> Prop,

   (forall n : nat, P 0 (leaf n)) ->

   (forall (m n : nat) (b : btree m),
   P m b -> forall b0 : btree m, P m b0 ->
     P (S m) (node m n b b0)) ->

   forall (n : nat) (b : btree n), P n b
```

Konstrukcja warunkowa

- konstrukcja `if t then e1 else e2` jest skrótowym zapisem konstrukcji

```
match t with
| c1 ...  => e1
| c2 ...  => e2
end
```

- typ termu `t` musi mieć dokładnie 2 konstruktory (`c1, c2`)
- wyrażenia `e1, e2` nie zależą od argumentów konstruktorów
- szczególny przypadek: gdy `b:bool`, to `if b then e1 else e2` jest skrótem dla

```
match b with
| true => e1
| false => e2
end
```

Własności rozstrzygalne

```
Inductive sumbool (A B : Prop) : Set :=
|  left : A -> {A} + {B}
| right : B -> {A} + {B}.

Check sumbool_ind.
> sumbool_ind
 : forall (A B : Prop) (P : {A} + {B} -> Prop),
   (forall a : A, P (left B a)) ->
   (forall b : B, P (right A b)) -> forall s : {A} + {B}, P s
```

- w przeciwieństwie do `A \/ B`, typ `{A} + {B}` można eliminować w obliczeniach (mamy `sumbool_rec`, `sumbool_rect`)
- typ używany do rozstrzygania równości w typach obliczeniowych

Własności rozstrzygalne, c.d.

- własność rozstrzygalnej równości w typie `T`
  `forall a b:T, {a = b} + {a <> b}`
- `eq_T_dec : forall a b:T, {a = b} + {a <> b}` nie tylko sprawdza równość, ale dostarcza odpowiedniego dowodu
- dla przyzwoitych typów indukcyjnych da się dowieść taktyką `decide equality`
- można używać w konstrukcji `if ... then ... else` (o ile nie potrzebujemy dowodu)
- dla typu `nat`: `if eq_nat_dec a b then ... else ...`