Inductive types in Coq

Wessel van Staal

November 23, 2012

Inductive types

```
Inductive nattree : Set :=
  leaf : nat      -> nattree
| node : nattree -> nattree -> nattree.
```

- Adds constant or function with final type *Prop*, *Set* or *Type* to the context.
- An inductive type is closed under its *constructors*, functions that produce the type.
- Enables computational concepts (case distinction, recursion).
- Enables proof by induction principle.

Parametric arguments

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

- *Parametric arguments* are defined for the whole inductive definition.
- Stability constraint: parameters must be reused in the exact order of definition in the final term of the constructor.
- Each inductive type definition with parametric arguments can be converted to an inductive type without parametric arguments.
  - Each parameter is pushed down to each constructor.

Parametric arguments (2)

```
Inductive Term (A:Set) : Type :=
| App : forall B:Set, Term (A->B) -> Term A -> Term B
.

Inductive Term : Set -> Type :=
| App : forall A:Set, forall B:Set, Term (A->B) -> Term A -> Term B
.
```

Constructors

Each constructor of inductive type T is of the following form:

$$t_1 \to t_2 \to \cdots \to t_j \to T\ a_1\ a_2\ \cdots\ a_n$$

If T is a function, then $n > 0$. Each t_i constitutes an argument of the constructor and must be well-typed, with $j \geq 0$.

- The term $T\ a_1\ a_2\ \cdots\ a_n$ must be well-formed and well-typed; it must respect the stability constraint.
- The type T cannot appear among arguments $a_1\ a_2\ \cdots\ a_n$.

Positivity constraints

$$t_1 \to t_2 \to \cdots \to t_j \to T\ a_1\ a_2\ \cdots\ a_n$$

Each t_i with $1 \leq i \leq j$ must respect the following constraints:

- If t_i is a function, then the inductive type T may only occur in the final type of the function (i.e. T must not appear left of the arrow).
- For each occurrence of $T\ a'_1\ a'_2\ \cdots\ a'_m$ in t_i, T must not appear in a'_j where $1 \leq j \leq m$.

The constructor can have a dependent type of form $\forall t \in D, U$. In that case, D and U must respect the positivity constraints.

Well-formed or not?

```
Inductive T : Type := t : (T->T) -> T.

Inductive I : Type := i : forall T:Type, (T -> I) -> I.

Inductive Term : Type -> Type :=
| Abs : forall A:Type, forall B:Type, (A -> Term B) -> Term (A->B).

Inductive T2 : Type->Type := p : T2 (T2 nat).
```

Violating the positivity constraint

```
Inductive T : Set := Fn : (T->T) -> T.

Definition Iterate : T->T :=
 fun (t:T) => match t with Fn f => f t.
```

Violating the positivity constraint (2)

```
Inductive T : Type := Fn : (T -> T) -> T .

Definition app : T->T->T :=
fun x:T => fun y:T => match x with Fn f => f y.

Definition t : T->T := fun x:T => app x x.

Definition omega : T := app (Fn t) (Fn t).
```

Simulating Ω $(\lambda x.xx)(\lambda x.xx)$:

- app : $\lambda xy.\ x\ y$.
- t : $\lambda x.\ app\ x\ x$.
- omega : $app\ t\ t$.

Universe constraint

- Sort of the inductive type T and the sort of each constructor type is the same up to convertibility.
- For T in sort s, for all constructor arguments in sort s', $s' : s$.
  - Prop : Set
  - Set : Type_i, $\forall i$
  - Type_i : Type_j, if $i \leq j$

Universe constraint

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

Inductive T1 : Set :=
c1 : Set -> T.

Inductive T2 : Set :=
c2 : forall x:Set, T2.

Inductive T3 : Type :=
c3 : forall x:Type, T3.
```

Induction principle cookbook

For inductive type T,

1. Header
   - Universal quantification over the parameters of T
   - Universal quantification over predicates ranging over elements of T
2. Principal premises
   - Predicate needs to hold for all uses of each constructor
   - Induction hypothesis for each argument of with final type T
3. Epilogue
   - The predicate holds for all elements of T

Header

- Universal quantification over the parameters of T
- Universal quantification over predicates ranging over elements of T
  - Predicates receive $k + 1$ arguments, where k is the number of non-parametric arguments of T.
- Construct headers for the following types:

```
Inductive T1 (A:Set) (B:Set) : Set := t1 : T1 A B.

Inductive T2 : Set -> Set -> Set   := t2 : T2 nat nat.
```

Principal premises

For each constructor of T,

- Universal quantification for each argument
  - Add induction hypothesis for arguments with type T
  - Also add induction hypothesis if the argument is a function with final type T

Construct principal premises for the following example:

```
Inductive Term : Type -> Type :=
| Val : forall A:Type, A -> Term A
| Abs : forall (A:Type) (B:Type), (A -> Term B) -> Term (A->B)
| App : forall (A:Type) (B:Type), Term (A->B) -> Term A -> Term B.
```