Proving with Computer Assistance

Lecture 11

Programming with dependent inductive types

Herman Geuvers

For the slides, thanks to: Freek Wiedijk

Coq examples using dependent types

natlist

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

append & reverse

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

Fixpoint reverse (k : natlist) : natlist :=
  match k with
    nil => nil
  | cons h t => append (reverse t) (cons h nil)
  end.
```

dependent lists

we will define a type for lists of a given length

```
a : (natlist_dep 6)
```

this corresponds in normal programming languages to something like

```
int a[6]
```

the type of a is called a **dependent** type
it **depends** on the natural number 6

```
natlist_dep : nat -> Set
natlist_dep 6 : Set
```

natlist_dep

```
Inductive natlist_dep : nat -> Set :=
    nil : natlist_dep 0
  | cons : forall n : nat,
      nat -> natlist_dep n -> natlist_dep (S n).
```

$$3, 1, 4, 1, 5, 9$$

$$\downarrow$$

```
cons 5 3 (cons 4 1 (cons 3 4 (cons 2 1 (cons 1 5 (cons 0 9 nil)))))

                                                    : (natlist_dep 6)
```

zeroes for dependent lists

```
Fixpoint zeroes (n : nat) : natlist_dep n :=
  match n with
    O => nil
  | S n' => cons n' O (zeroes n')
  end.
```

the type of dependent zeroes

~~zeroes : nat -> natlist_dep ?~~

```
zeroes : forall n : nat, natlist_dep n
```

dependent product

generalizes the notion of function type

append for dependent lists

```
Fixpoint append (n m : nat)
  (k : natlist_dep n) (l : natlist_dep m) {struct k} :
    natlist_dep (plus n m) :=
  match k with
    nil => l
  | cons n' h t => cons (plus n' m) h (append n' m t l)
  end.
```

programming with dependent types: `reverse` for dependent lists

```
Fixpoint reverse
  (n : nat) (k : natlist_dep n) {struct k} :
    natlist_dep n :=
  match k with
    nil => nil
  | cons n' h t =>
      eq_rec (plus n' 1) (fun n => natlist_dep n)
        (append n' 1 (reverse n' t) (cons 0 h nil))
        (S n') (plus_one n')
  end.
```

has type `natlist_dep (plus n' 1)`

but should have type `natlist_dep (S n')`

dependently typed functional programming languages

- **cayenne**

  'dependent haskell'

  Lennart Augustsson

- **dependent ML**

  Hongwei Xi

- **epigram**

  Conor McBride