Dependent types II: Proofs == Programs

Guest lecture (Mark Barbone)

Last time...

- Types are terms, too!
- Dependent function type (“pi type”)
  - f has type $\Pi(x{:}A) \to B$ if whenever $a : A$, then $f(a) : B[a/x]$
  - Handles both functions $A \to B$ and polymorphic types $\forall x. B$
- Dependent pair type (“sigma type”)
  - (a,b) has type $\Sigma(x{:}A) \times B$ if $a : A$, and $b : B[a/x]$
  - Syntactic sugar: $A \times B$ if x is not used

A couple basic new types

We'll use these (non-dependent) types, too.

- The unit type **1** has the one element () : **1**
- The type A + B has elements inl(a) and inr(b), for a : A and b : B.
  - Pattern matching: case e of inl(a) ⇒ … | inr(b) ⇒ …
  - Special case: bool = **1** + **1**
- The empty type **0** has no elements
  - Pattern matching: case e of {} (outputs any type you like!)

Interesting things you can do with boring types

Suppose we have a BST data structure, and a function

valid : BST → bool

Q: what are the inhabitants of

Σ(tree : BST) × (if valid(tree) then **1** else **0**) ?

A: pairs (tree, ()), but only if valid(tree)

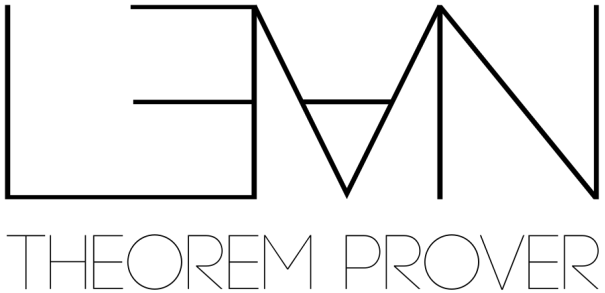
→ (in bijection with) the set of valid trees!

Propositions

The type if valid(tree) then **1** else **0** is a proposition:

- We don't really care about its elements — just that it has one
- e : if valid(tree) then **1** else **0** is evidence that the tree is valid

Q: How much logic can we do with propositions as types?



Implementing logic with types

Proposition	**Type**
true	**1**
false	**0**
P and Q	P × Q
P or Q	P + Q
P implies Q	P → Q

Problems

For each tautology, (a) write down a corresponding type, and (b) prove it by making a term of that type

- P implies P
- P implies (Q implies P)
- P implies true
- false implies P
- (P and Q) implies P
- (P and Q) implies (Q and P)

Is there anything you can’t prove like this?

It turns out you can’t do proof by contradiction

To support proof by contradiction, we need to add an extra built-in operation.

Implementing logic with types (cont'd)

Proposition	**Type**
for all x:A, P(x)	$\Pi(x{:}A) \to P(x)$
there exists x:A, s.t. P(x)	$\Sigma(x{:}A) \times P(x)$

Leibniz equality

A coercion function $f : \forall\ B.\ B(x) \to B(y)$ gives evidence that $x == y$.

Let $x == y$ abbreviate the type $\forall\ B.\ B(x) \to B(y)$.

Prove (by writing well-typed lambda terms) the following:

1. Reflexivity: $x == x$
2. Transitivity: if $x == y$ and $y == z$, then $x == z$.
3. Symmetry: if $x == y$, then $y == x$. (harder!)

More quantifiers

1. If for all x, P(x) implies Q, then (there exists x s.t. P(x)) implies Q
2. If (there exists x s.t. P(x)) implies Q, then for all x, P(x) implies Q
3. If there exists x s.t. for all y, P(x,y), then for all y, there exists x s.t. P(x,y)
4. ("Axiom of choice") If for all x : A, there exists y : B s.t. P(x,y),
   then there exists a function f : A → B s.t. for all x : A, P(x, f(x))