Dependent Types and Theorem Proving: Introduction to Dependent Types

Wojciech Kołowski

March 2021

Non-dependent pairs

- Recall how ordinary pairs work in F#.
- If `a : Type` is a type and `b : Type` is a type, then there is a type `a * b : Type` of pairs.
- To create a pair, we write `(x, y)` where `x` is of type `a` and `y` is of type `b`.
- To use a pair `p : a * b`, we use projections – we have `fst p : a` and `snd p : b`.
- We can also pattern match on pairs.

Dependent pairs

- Now, watch the analogy unfold. . .
- If `a : Type` is a type and `b : a -> Type` **is a family of types**, then there is **a type** `(x : a) & b x : Type` **of dependent pairs**.
- To create a dependent pair, we write `(| x, y |)` where `x` is of type `a` **and** `y` **is of type** `b x`.
- To use a pair `p : (x : a) & b x`, we use projections – we have `fst p : a` and `snd p : b (fst p)` (**note that the type of the second projection depends on the value of the first projection**).
- We can also pattern match on dependent pairs.

More dependent pairs

- **We can iterate the dependent pair type**, while dropping unneeded parentheses – analogously to what we did for dependent functions.
- `(x : a) & b x`
- `(x : a) & (y : b x) & c x y`
- `(x : a) & (y : b x) & (z : c x y) & d x y z`
- **But using iterated dependent pairs is very inconvenient!**
- To access component of a dependent quadruple `p` we would have to write `fst p`, `fst (snd p)`, `fst (snd (snd p))` and `snd (snd (snd p))`.

Dependent record types

- There's a better way than iterating dependent pair types: dependent record types.
- A record is basically a labeled tuple.
- **A dependent record is basically a labeled dependent tuple**.
- This means that the TYPES of later fields in a dependent record can depend on the VALUES of earlier fields.

Code snippet no 4 - dependent records in F*

- Let's see how dependent records work in F*.
- See the code snippet `Lecture1/DependentRecords.fst`

The running summary 4

- Dependent types are types that can depend on values.
- In dependently typed languages:
- There is a universe – a type whose elements are themselves types.
- There is a type of dependent functions which are just like ordinary functions, but their output TYPE can depend on the VALUE of their input.
- **Dependent record types are just like ordinary records, but the TYPES of later fields can depend on the VALUE of earlier fields**.

Pi type and multiplication

- The dependent function type **is also known as the Pi type**.
- **This name comes from a notation**: `(x : a) -> b x` is sometimes written as $\prod_{x:\ a} b(x)$.
- **This notation comes from an analogy with multiplication**. In math $\prod_{k=0}^{n} a_k$ means $a_0 \cdot a_1 \cdot \ldots \cdot a_n$.
- **We can think about dependent function types in this way too**. For example, the type `(x : bool) -> p x` is equivalent to `p true * p false`.
- The result of multiplication is called a product, hence the dependent function type **is also known as the dependent product type**.
- As it turns out, the dependent function type **generalizes both the ordinary function type and the product type**, but in different ways.

Sigma type and addition

- The dependent pair type **is also known as the Sigma type**.
- **This name comes from a notation**: `(x : a) & b x` is sometimes written as $\sum_{x:\ a} b(x)$.
- **This notation comes from an analogy with addition**. In math $\sum_{k=0}^{n} a_k$ means $a_0 + a_1 + ... + a_n$.
- **We can think about dependent pair types in this way too**. For example, the type `(x : bool) & p x` is equivalent to `p true + p false` (where + just means a simple tagged union).
- The result of addition is called a sum, hence the dependent pair type **is also known as the dependent sum type**.
- As it turns out, the dependent pair type **generalizes both the product type and the sum type**, but in different ways.