Dependent Types and Theorem Proving: Introduction to Dependent Types

Wojciech Kołowski

March 2021

Inductive types are discriminated unions

- Recall how ordinary inductive types work in F# (where they are called discriminated unions).
- To define an inductive type `I : Type`, we list its constructors.
- The constructors are ordinary functions which take some arguments and return an element of the type.
- To create an element of `I`, we use one of the constructors and provide it with the arguments it requires.
- To use an element of `I`, we pattern match on it and for each case we provide an expression which will be computed if that case matches.

Inductive families

- Now watch the analogy unfold...
- To define an **inductive family** `I : (#a : Type) -> a -> Type`, we list its constructors.
- **The constructors are dependent functions which take some arguments and return an element of the type** `I x`, where `x : a` was constructed from the arguments.
- To create an element of `I x`, we use one of the constructors and provide it with the arguments it requires.
- To use an element of `I x`, we pattern match on it and for each case we provide an expression which will be computed if that case matches.

Code snippet no 5 - inductive families in F*

- Let's see how inductive families work in F*.
- See the code snippet `Lecture1/InductiveFamilies.fst`

The running summary 5

- 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.
- Inductive families are just like ordinary inductive types, but the TYPES in the family can depend on the VALUE of the index.

Inductive types and polynomials 1/2

- An inductive type is **EITHER** constructor 1 applied to arguments `x1` **and** `x2` ... **and** `xN` **OR** constructor 2 applied to arguments ... **OR** constructor M applied to arguments ...
- In math, OR means **addition**, whereas AND means **multiplication**.
- So, an inductive type boils down to a **Sum of Products**.
- These products are made of two kinds of arguments: recursive arguments (whose type is the inductive type that is being defined) and non-recursive ones.
- If you think about it long enough, **inductive types correspond to polynomials**!

Inductive types and polynomials 2/2

- This could be hard to swallow, so let's see examples.
- Lists satisfy the equation $\texttt{List}(A) = 1 + A \times \texttt{List}(A)$.
- Here 1 corresponds to the `nil` constructor, whereas the A and $\texttt{List}(A)$ on the right correspond to the arguments of the `cons` constructor.
- This corresponds to the polynomial $F(X) = 1 + A \times X$.
- $\texttt{List}(A)$ is the least fixed point of this polynomial, i.e. the smallest type X that satisfies $F(X) = X$.
- Here "fixed point" corresponds to the fact that we create lists using constructors (`nil` and `cons`), whereas "least" corresponds to the fact that all lists are made of finitely many constructors.