Dependent Types and Theorem Proving: Introduction to Dependent Types

Wojciech Kołowski

March 2021

General info

- The lectures will be held weekly on Fridays.
- Don't worry if you miss a lecture – the slides are pretty massive and the talks are going to be recorded.
- Each lecture ends with some exercises which will help you familiarize yourself with F* and better understand the ideas covered in the talk.
- But you don't need to do them if you don't want to.

Plan of lectures

- Lecture 1: Programming with dependent types.
- Lecture 2: Proving theorems with dependent types.
- Lecture 3: Differences between programming and proving.
- Lecture 4: Examples of bigger programs and longer proofs.
- Lecture 5: A deeper dive into F*.

Learning outcomes

- You won't be scared of all those obscure, scary and mysterious names and notations.
- You will get basic familiarity with the ideas behind dependent types.
- You will begin to see logic and mathematics in a very different light, much closer to your day job (at least if you are a programmer working in F#).
- If you do the exercises, you will gain a basic proficiency in F*.

Introducing F* 1/2

- F* (pronounced "eff star") is a general-purpose purely functional programming language.
- It comes from the ML family of languages. Its syntax most closely resembles that of F#.
- It is aimed at program verification, i.e. first you write a program and then you prove theorems which say that there are no bugs in the program. This will be covered in lecture 2.
- It has traditional dependent types similar to those found in Coq, Agda, Lean or Idris. They are covered in lecture 1, i.e. today.
- It also has refinement types, a different flavour of dependent types which is better suited for automatic theorem proving (traditional dependent types are more suited for "manual" proving). They are covered in lecture 1, i.e. today.
- It has an effect system. This is a thing similar to what can be done with monads in Haskell. We will see it in lecture 3.

Introducing F* 1/2

- F* also has some more features which are unusual for functional languages, like mutable references and a weakest precondition calculus. These will be covered in lecture 5 or not at all.
- F* is NOT a .NET language.
- F* is neither compiled nor interpreted – it is mostly a typechecker.
- To run an F* program, it has to be extracted to some other language, like F# or OCaml, and then compiled.

F* ecosystem

- You can run F* inside your browser (and have a nice tutorial guide you): `http://www.fstar-lang.org/tutorial/`
- Homepage: `http://www.fstar-lang.org/`
- GitHub: `https://github.com/FStarLang/FStar`
- Download: `http://www.fstar-lang.org/#download`
- Papers (not approachable for ordinary mortals): `http://www.fstar-lang.org/#papers`
- Talks/presentations (more approahcable): `http://www.fstar-lang.org/#talks` (some of these are quite approachable if you're interested)

Prerequisites

-