Programming with Evidence

The introduction to an introduction to Agda

Uma Zalakain
Formal Methods group, University of Glasgow

Basque Center for Applied Mathematics
November 23, 2021

About me

- second year PhD student at the Formal Methods group at University of Glasgow
- machine verification of typed process calculi:
  using proof assistants to model typed concurrency languages and to verify their meta-theory
- programming languages theory, concurrency theory,
  type theory, distributed systems

Yesterday: Zermelo-Fraenkel Set Theory

- a foundation for mathematics
- untyped, $x \in A$ is a *proposition*
- elements can belong to different sets
- a set is fully characterised by its elements
- primitives: set operations ($\cup$, $\cap$)
- predicate logic

Today: Martin-Löf Type Theory

- also a foundation for mathematics
- typed, $x : A$ is a *judgment*
- an element has a unique type
- a type is *not* characterised by its elements
- primitives: datatypes and functions
- propositions as types (Π and Σ model predicate logic)
- constructive: *a programming language!*

Propositions as Types

- propositions are (proof-relevant) types
- proofs are programs
- evidence is data
- constructivism:
  existence requires the construction of a witness

Propositions as Types

proposition	type
$\bot$	$\texttt{Zero}$
$\top$	$\texttt{One}$
$A \wedge B$	$\texttt{A} \times \texttt{B}$
$A \vee B$	$\texttt{A} \uplus \texttt{B}$
$A \implies B$	$\texttt{A} \rightarrow \texttt{B}$
$\neg A$	$A \rightarrow \texttt{Zero}$
$\forall x. P\,x$	$\Pi\,(\texttt{x} : \texttt{A})\,(\texttt{P}\,\texttt{x})$
$\exists x. P\,x$	$\Sigma\,(\texttt{x} : \texttt{A})\,(\texttt{P}\,\texttt{x})$

Interactive Proof Assistants

- system checks proofs for correctness
- system helps the user to construct those proofs interactively
- interactive proving becomes interactive programming
- requires less trust, easier to refactor with confidence
- educational value — instant feedback for the student
- easy to reuse: shared library of definitions and proofs
- proofs compute!
- a lot of fun!

Interactive Proof Assistants

- Coq: based on the Calculus of Inductive Constructions, heavy use of tactics
- Lean: based on the Calculus of Inductive Constructions, small kernel, support for quotient types
- Idris2: based on Quantitative Type Theory, supports linearity annotations, focuses on compilation
- Agda: very close to Martin-Löf Type Theory, handles proof terms directly

Agda

- developed mainly at Chalmers, Sweden
- clean syntax, unicode support
- based on dependent pattern matching
- mostly used in:
    - Programming Language Theory
    - Category Theory
    - Homotopy Theory

Dependent Types

- types contain value-level expressions
- allow correct-by-construction problem modelling
- pre and post conditions can be tightened
  using types as specifications
- empty types rule out impossible cases

About the tutorials

- Monday to Thursday
- 9h total
- interactive — an emacs buffer
- available online: `https://umazalakain.github.io/agda-bcam/`
- recorded for posterity (including mistakes)

About the tutorials

- simple and composite types
- unicode and mixfix operators
- interactive programming
- record types
- Curry-Howard correspondence
- dependent function types
- indexed data types
- parametrised modules
- with abstraction
- automated evidence-providing solvers

Bibliography

- *Introduction to Agda*, **Andreas Abel**, 8th Summer School on Formal Techniques (SSFT'18) Menlo College, California, US
- *Computer Aided Formal Reasoning*, **Thorsten Altenkirch**, 2010
- *A Practical Agda Tutorial*, **Péter Diviánszky and Ambrus Kaposi**, 2013