Formalizing inference systems in Coq by means of type systems for Curry

Niels Bunkenburg

Programming Languages and Compiler Construction
Department of Computer Science
Christian-Albrechts-University of Kiel

29.09.2016

Motivation

Outline

Outline

Coq - Data Types and Functions

- Inductive definitions

```
Inductive list {X : Type} : Type :=
  | nil  : list X
  | cons : X -> list X -> list X.
```

Coq - Data Types and Functions

- Inductive definitions

```
Inductive list {X : Type} : Type :=
  | nil  : list X
  | cons : X -> list X -> list X.
```

- (Recursive) functions

```
Fixpoint app {X: Type} (l1 l2: list X) : (list X) :=
  match l1 with
  | nil => l2
  | cons h t => cons h (app t l2)
  end.
```

Coq - Propositions

- Equations

```
1 + 1 = 2.
forall (X : Type) (l : list X), l ++ [] = l.
```

- Inductively Defined Propositions

```
Inductive inInd : nat -> list nat -> Prop :=
  | head: forall n l, inInd n (n :: l)
  | tail: forall n l e, inInd n l -> inInd n (e :: l).
```

Curry

- Syntax similar to Haskell
- Nondeterminism

```
(?)    :: a -> a -> a
x ? _ = x
_ ? y = y
```

- Free variables

```
1 + 1 == x where x free
```

Outline

Typing

Context

Inference rules

$$\frac{}{\texttt{In n (n :: l)}}\ \texttt{In\_H} \qquad\qquad \frac{\texttt{In n l}}{\texttt{In n (e :: l)}}\ \texttt{In\_T}$$

Outline

Syntax - Backus–Naur Form

$$
\begin{aligned}
P &::= D; P \mid D \\
D &::= f :: \kappa\tau; f\,\overline{x_n} = e \\
\kappa &::= \forall^{\epsilon}\alpha.\kappa \mid \forall^{*}\alpha.\kappa \mid \epsilon \\
\tau &::= \alpha \mid \mathrm{Bool} \mid \mathrm{Nat} \mid [\tau] \mid (\tau, \tau') \mid \tau \rightarrow \tau' \\
e &::= x \mid f_{\overline{\tau_m}} \mid e_1\ e_2 \mid \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 \mid n \mid e_1 + e_2 \mid e_1 \stackrel{\circ}{=} e_2 \\
&\mid (e_1, e_2) \mid \mathrm{case}\ e\ \mathrm{of}\ \langle (x, y) \rightarrow e_1 \rangle \\
&\mid \mathrm{True} \mid \mathrm{False} \mid \mathrm{case}\ e\ \mathrm{of}\ \langle \mathrm{True} \rightarrow e_1;\ \mathrm{False} \rightarrow e_2 \rangle \\
&\mid \mathrm{Nil}_{\tau} \mid \mathrm{Cons}(e_1, e_2) \mid \mathrm{case}\ e\ \mathrm{of}\ \langle \mathrm{Nil} \rightarrow e_1;\ \mathrm{Cons}(x, y) \rightarrow e_2 \rangle \\
&\mid \mathrm{failure}_{\tau} \mid \mathrm{anything}_{\tau}
\end{aligned}
$$

fst :: $\forall^*\alpha.\forall^*\beta.(\alpha,\beta) \rightarrow \alpha$

fst p = case p of $\langle(u, v) \rightarrow u\rangle$

one :: Nat

one = $\text{fst}_{\mathit{Nat},\mathit{Bool}}$ (1, True)

Syntax - Coq

```
Inductive quantifier : Type :=
  | for_all : id -> tag -> quantifier.

Inductive ty : Type :=
  | TVar  : id -> ty
  | TBool : ty
  | TNat  : ty
  | TList : ty -> ty
  | TPair : ty -> ty -> ty
  | TFun  : ty -> ty -> ty.

Definition program := list func_decl.
Inductive func_decl : Type :=
  | FDecl : id -> list quantifier ->
        ty -> list id -> tm -> func_decl.
```

Context

Outline

Typing rules

Examples

Outline

Syntax

Outline

Summary

Future Work