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.
```

- (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
{x = (-_x2)} False
{x = 0} False
{x = 1} False
{x = 2} True
{x = (2 * _x3 + 1)} False
{x = (4 * _x4)} False
{x = (4 * _x4 + 2)} False
```

Outline

Typing

- **Type**: Set of values that determines properties and meaning of its elements. For example `Int`, `Maybe` or `[]`.
- **Expression**: Combination of literals, variables, operators and functions. E.g. `1 + 1` or `map double [1,2,3]`.
- **Context**: Contains information about variables and the program.
- **Typing**: Assigning a type to an expression in a context.

Inference rules

$\dfrac{p_1 \ldots p_n}{c}$ where p_i are premises and c is the conclusion of the rule.

- Typing: **If** $p_1 \ldots p_n$ **then** $\Gamma e \vdash \tau$

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

Outline

Syntax – Backus-Naur Form

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

$$\text{fst} :: \forall^*\alpha.\forall^*\beta.(\alpha, \beta) \to \alpha \qquad \text{one} :: \text{Nat}$$

$$\text{fst}\ p = \text{case}\ p\ \text{of}\ \langle(u, v) \to u\rangle \qquad \text{one} = \text{fst}_{Nat,Bool}\ (1, \text{True})$$

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

$$\Gamma,\ x \mapsto \tau \vdash x :: \tau \qquad \Gamma \vdash \text{True} :: \text{Bool} \qquad \Gamma \vdash \text{False} :: \text{Bool} \qquad \Gamma \vdash n :: \text{Nat} \qquad \Gamma \vdash \text{Nil}_\tau :: [\tau]$$

$$\frac{\Gamma \vdash e_1 :: \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 :: \tau_1}{\Gamma \vdash e_1\, e_2 :: \tau_2} \qquad \frac{\Gamma \vdash e_1 :: \tau_1 \quad \Gamma, x \mapsto \tau_1 \vdash e_2 :: \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 :: \tau} \qquad \frac{(f :: \forall^{v_1}\alpha_1. \cdots \forall^{v_m}\alpha_m.\tau; \overline{f x_n}}{\Gamma \vdash f_{\overline{\tau_m}} :: \tau[\overline{\tau_m/\alpha_m}]}$$

$$\frac{\Gamma \vdash e_1 :: \text{Nat} \quad \Gamma \vdash e_2 :: \text{Nat}}{\Gamma \vdash e_1 + e_2 :: \text{Nat}} \qquad \frac{\Gamma \vdash e_1 :: \text{Nat} \quad \Gamma \vdash e_2 :: \text{Nat}}{\Gamma \vdash e_1 \stackrel{\circ}{=} e_2 :: \text{Bool}} \qquad \frac{\Gamma \vdash e_1 :: \tau_1 \quad \Gamma \vdash e_2 :: \tau_2}{\Gamma \vdash (e_1, e_2) :: (\tau_1, \tau_2)} \qquad \frac{\Gamma \vdash e_1 :: \tau}{\Gamma \vdash \text{Co}}$$

$$\frac{\Gamma \vdash e :: [\tau'] \quad \Gamma \vdash e_1 :: \tau \quad \Gamma, h \mapsto \tau', t \mapsto [\tau'] \vdash e_2 :: \tau}{\Gamma \vdash \text{case } e \text{ of } \langle \text{Nil} \to e_1; \text{Cons}(h, t) \to e_2 \rangle :: \tau}$$

$$\frac{\Gamma \vdash e :: (\tau_1, \tau_2) \quad \Gamma, l \mapsto \tau_1, r \mapsto \tau_2 \vdash e_1 :: \tau}{\Gamma \vdash \text{case } e \text{ of } \langle (l, r) \to e_1 \rangle :: \tau}$$

$$\frac{\Gamma \vdash e :: \text{Bool} \quad \Gamma \vdash e_1 :: \tau \quad \Gamma \vdash e_2 :: \tau}{\Gamma \vdash \text{case } e \text{ of } \langle \text{True} \to e_1; \text{False} \to e_2 \rangle :: \tau} \qquad \Gamma \vdash \text{failure}_\tau :: \tau \qquad \frac{\Gamma \vdash \tau \in \text{Data}}{\Gamma \vdash \text{anything}_\tau :: \tau}$$

★ if for all i with $v_i = *$ we have $\Gamma \vdash \tau_i \in \text{Data}$

Figure: Typing rules for CuMin

Examples

Outline

Syntax

Outline

Summary

Future Work