Formalisierung von Inferenzsystemen in Coq am Beispiel von Typsystemen für Curry

Niels Bunkenburg

29.09.2016

Arbeitsgruppe für Programmiersprachen und Übersetzerkonstruktion
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Motivation

Curry

Coq

CuMin

FlatCurry

Introduction

Curry

- Syntax ähnlich zu Haskell
- Nichtdeterminismus

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

- Freie Variablen

```
> 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
```

Coq - Aussagen

- Gleichungen

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

- Induktiv definierte Aussagen

```
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).
```

Was ist Typisierung?

- **Typ**: Menge von Werten, die Eigenschaften und Bedeutung der Elemente bestimmt, beispielsweise `Int`, `Maybe` oder `[]`.
- **Ausdruck**: Kombination von Literalen, Variablen, Operatoren und Funktionen, z.B. `1 + 1` or `map double`.
- **Kontext**: Enthält Informationen über Variablen und das Programm.
- **Typisierung**: In einem Kontext Γ wird einem Ausdruck e ein Typ τ zugewiesen, notiert als $\Gamma \vdash e :: \tau$.

Beispiele:

- $\Gamma \vdash 2 :: \texttt{Int}$
- $\Gamma \vdash \texttt{let x = 2 in x + 2} :: \texttt{Int}$

Inferenzregeln

$\frac{p_1 \ldots p_n}{c}$ wo p_i Prämissen und c Konklusion der Regel.

- Notation für Implikation $p_1 \to \cdots \to p_n \to c$
- Typing: **If** $p_1 \ldots p_n$ **then** $\Gamma e \vdash \tau$

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

CuMin

Syntax – Backus-Naur Form

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

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

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

Syntax – Coq

```
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

Typing rules

Examples

FlatCurry

Syntax

Conclusion

Summary

Future Work