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

Curry

- Syntax ähnlich zu Haskell
- Nichtdeterminismus

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

- Freie Variablen

```
> True && x == False where x free
  {x = False} True
  {x = True} False
```

Coq - Aussagen

- Aussagen in Coq: `Prop`

```
1 + 1 = 2
False -> False
forall (X : Type) (l : list X), [] ++ l = l
fun (x : nat) => x <> 2
```

- Induktiv definierte Aussagen

```
Inductive inInd {X : Type} : X -> list X -> Prop :=
  | head : forall x l, inInd x (x :: l)
  | tail : forall x y l, inInd x l -> inInd x (y :: l).

Example e : inInd 2 [1;2;4].
Proof.
  apply tail. (* inInd 2 [2; 4] *)
  apply head.
Qed.
```

Was ist Typisierung?

- **Typ**: Menge von Werten, die Eigenschaften und Bedeutung der Elemente bestimmt.
- **Ausdruck**: Kombination von Literalen, Variablen, Operatoren und Funktionen.
- **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:

- $\Gamma \vdash 2 :: \texttt{Int}$
- $\Gamma, x \mapsto \texttt{Int} \vdash \texttt{x + 2} :: \texttt{Int}$

Inferenzsysteme

- Inferenzsystem: Menge von Inferenzregeln
- Inferenzregel: $\dfrac{p_1 \dots p_n}{c}$ wo p_i Prämissen und c Konklusion
- Notation für Implikation $p_1 \to \dots \to p_n \to c$
- Darstellung von Inferenzregeln in Coq:

$$\frac{}{\texttt{In x (x :: l)}}\ \texttt{head} \qquad\qquad \frac{\texttt{In x l}}{\texttt{In x (y :: l)}}\ \texttt{tail}$$

```
Inductive inInd {X : Type} : X -> list X -> Prop :=
  | head : forall x l, inInd x (x :: l)
  | tail : forall x y l, inInd x l -> inInd x (y :: l).
```

Formalisierung von Typsystemen für Curry

Vorgehensweise:

1. Syntax und Kontext in Coq darstellen.
2. Typisierungsregeln in induktive Aussagen umwandeln.
3. Code umwandeln und Eigenschaften beweisen.

Sprachen:

- CuMin (Curry Minor): Vereinfachte Teilsprache von Curry.
- FlatCurry: Zwischensprache, die in Curry Compilern benutzt wird.

CuMin – Syntax in BNF

Auszug aus der Syntax von CuMin in Backus-Naur-Form:

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

Beispiel CuMin

Formale Definition:

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

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

Coq Definition:

```
Definition fst := FDecl (Id 0)
  [for_all (Id 1) tag_star; for_all (Id 2) tag_star]
  (TFun (TPair (TVar (Id 1)) (TVar (Id 2))) (TVar (Id 1)))
  [Id 3]
  (tcasep (tvar (Id 3)) (Id 4) (Id 5) (tvar (Id 4))).

Definition one := tapp (tfun (Id 0) [TNat; TBool])
  (tpair (tsucc tzero) ttrue).
```

CuMin – Datentypen und Kontext

Inferenzregeln für Datentypen:

$$\Gamma, \alpha^* \vdash \alpha \in \mathsf{Data} \qquad \Gamma \vdash \mathsf{Bool} \in \mathsf{Data} \qquad \Gamma \vdash \mathsf{Nat} \in \mathsf{Data}$$

$$\frac{\Gamma \vdash \tau \in \mathsf{Data}}{\Gamma \vdash [\tau] \in \mathsf{Data}} \qquad \frac{\Gamma \vdash \tau \in \mathsf{Data} \quad \Gamma \vdash \tau' \in \mathsf{Data}}{\Gamma \vdash (\tau, \tau') \in \mathsf{Data}}$$

Informationen, die im Kontext enthalten sind:

```
Definition program := list func_decl.

Inductive context : Type :=
  | con: (partial_map id tag) -> (partial_map id ty) ->
        context.
```

$$\frac{\Gamma \vdash e_1 :: \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 :: \tau_1}{\Gamma \vdash e_1\ e_2 :: \tau_2}$$

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

$$\frac{(f :: \forall^{v_1}\alpha_1.\cdots\forall^{v_m}\alpha_m.\tau; f\,\overline{x_n} = e) \in P}{\Gamma \vdash f_{\overline{\tau_m}} :: \tau[\overline{\tau_m/\alpha_m}]}\ \bigstar$$

$\bigstar$ Falls für alle i mit $v_i = *$ gilt $\Gamma \vdash \tau_i \in \text{Data}$.

CuMin – Typsystem in Coq

```
Inductive has_type : context -> tm -> ty -> Prop :=
  | T_App: forall Gamma e1 e2 T1 T2,
             Gamma |- e1 ::: (TFun T1 T2) ->
             Gamma |- e2 ::: T1 ->
             Gamma |- (tapp e1 e2) ::: T2

  | T_Fun: forall Gamma id tys T,
             let result := lookup_func Prog id in
             let fdecl  := fromOption default_fd result in
             specialize_func fdecl tys = Some T ->
             Forall (is_data_type Gamma)
                    (fd_to_star_tys fdecl tys) ->
             Gamma |- (tfun id tys) ::: T

where "Gamma '|-' t ':::' T" := (has_type Gamma t T).
```

Live Vorführung

Beweis der Aussage $\Gamma \vdash \texttt{one} :: \texttt{Nat}$

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

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

FlatCurry Überblick

- Selbstdefinierte Datentypen
- Weniger Ausdrücke und Typen als CuMin, dafür allgemeinere Form
  $\rightarrow$ Komplexeres Typsystem

FlatCurry Überblick

- Selbstdefinierte Datentypen
- Weniger Ausdrücke und Typen als CuMin, dafür allgemeinere Form
  $\rightarrow$ Komplexeres Typsystem

$$\frac{\Gamma, \overline{x_i \mapsto \tau_{1_i}[\ \overline{v_j \mapsto t_j}\ ]} \vdash e_1 :: \tau' \qquad \ldots \qquad \Gamma, \overline{x_i \mapsto \tau_{k_i}[\ \overline{v_j \mapsto t_j}\ ]} \vdash e_k :: \tau' \qquad \Gamma \vdash e :: \tau[\ \overline{v_j \mapsto t_j}\ ]}{\Gamma, Cs \vdash \text{(f)case e of } \{C_1\ x_1 \ldots x_n \rightarrow e_1; \ldots; C_k\ x_1 \ldots x_m \rightarrow e_k\} :: \tau'} \text{ with } k > 0$$

$$\text{with } Cs = \{C_1 \mapsto (\tau_{1_1} \rightarrow \cdots \rightarrow \tau_{1_n} \rightarrow \tau, \overline{v_i}), \ldots, C_k \mapsto (\tau_{k_1} \rightarrow \cdots \rightarrow \tau_{k_m} \rightarrow \tau, \overline{v_i})\}$$

Zusammenfassung

- Formalisierung der Syntax als induktive Datentypen
- Erstellung des Kontexts
- Darstellung der Inferenzregeln des Typsystems in Coq
- Zwei Ansätze: CuMin und FlatCurry
- FlatCurry: Kontextparser + Curry Übersetzer

Erweiterungsmöglichkeiten: Typinferenz und Formalisierung der Semantik.