Der Kalkül des natürlichen Schließens

Teil 4: Programme als Beweise

Die Curry-Howard-Korrespondenz

Die Schreibweise $a : A$ drückt das Urteil aus, dass a ein Term vom Typ A sein muss.

Die Schreibweise $a : A$ drückt das Urteil aus, dass a ein Term vom Typ A sein muss. Analog zur Logik ist ein Kontext eine Liste

$$\Gamma = [a_1 : A_1, \ldots, a_n : A_n].$$

Ist nun die Sequenz

$$\Gamma \vdash (b : B)$$

eine ableitbare, liegt das Urteil $b : B$ vor, sofern die Terme in Γ vorausgesetzt werden dürfen. Das ist so zu verstehen, dass man mit dem Termen aus Γ den Term b zusammenbasteln kann. Ein Term darf hierbei mehrmals benutzt werden.

Die Schreibweise $a : A$ drückt das Urteil aus, dass a ein Term vom Typ A sein muss. Analog zur Logik ist ein Kontext eine Liste

$$\Gamma = [a_1 : A_1, \ldots, a_n : A_n].$$

Ist nun die Sequenz

$$\Gamma \vdash (b : B)$$

eine ableitbare, liegt das Urteil $b : B$ vor, sofern die Terme in Γ vorausgesetzt werden dürfen. Das ist so zu verstehen, dass man mit dem Termen aus Γ den Term b zusammenbasteln kann. Ein Term darf hierbei mehrmals benutzt werden.

Zur Konstruktion von Termen finden sich nun Schlussregeln, die die logischen widerspiegeln.

Zur Konstruktion von Termen finden sich nun Schlussregeln, die die logischen widerspiegeln.

Axiom (Einführung von Grundsequenzen)

Aussagen	**Typurteile**
$\dfrac{}{A \vdash A}$	$\dfrac{}{(a : A) \vdash (a : A)}$

Schlussregeln

Konjunktion	**Paar**
$\dfrac{\Gamma \vdash A \qquad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \wedge B}$	$\dfrac{\Gamma \vdash a : A \qquad \Gamma' \vdash b : B}{\Gamma, \Gamma' \vdash (a, b) : A \times B}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_l(t) : A}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_r(t) : B}$

Schlussregeln

Konjunktion	**Paar**
$\dfrac{\Gamma \vdash A \qquad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \wedge B}$	$\dfrac{\Gamma \vdash a : A \qquad \Gamma' \vdash b : B}{\Gamma, \Gamma' \vdash (a, b) : A \times B}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_l(t) : A}$
$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$	$\dfrac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_r(t) : B}$

In Worten: Die Einführung der Konjunktion entspricht der Konstruktion des Paares. Die Beseitigung der Konjunktion entspricht der Projektion auf eine der Komponenten.

Schlussregeln

Implikation

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash B}$$

Funktion

$$\frac{\Gamma, a : A \vdash b : B}{\Gamma \vdash (a \mapsto b) : A \to B}$$

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma' \vdash a : A}{\Gamma, \Gamma' \vdash f(a) : B}$$

Schlussregeln

Implikation	**Funktion**
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma, a : A \vdash b : B}{\Gamma \vdash (a \mapsto b) : A \to B}$
$\dfrac{\Gamma \vdash A \to B \qquad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash B}$	$\dfrac{\Gamma \vdash f : A \to B \qquad \Gamma' \vdash a : A}{\Gamma, \Gamma' \vdash f(a) : B}$

In Worten: Die Einführung der Implikation entspricht der Einführung einer anonymen Funktion (Abstraktion). Die Beseitigung der Implikation entspricht der Applikation der Funktion.

Schlussregeln

Implikation	**Funktion**
$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma, a : A \vdash b : B}{\Gamma \vdash (a \mapsto b) : A \to B}$
$\dfrac{\Gamma \vdash A \to B \qquad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash B}$	$\dfrac{\Gamma \vdash f : A \to B \qquad \Gamma' \vdash a : A}{\Gamma, \Gamma' \vdash f(a) : B}$

In Worten: Die Einführung der Implikation entspricht der Einführung einer anonymen Funktion (Abstraktion). Die Beseitigung der Implikation entspricht der Applikation der Funktion.

Alonzo Church schrieb $\lambda x.\, t$ anstelle von $x \mapsto t$. Die Applikation $f(x)$ wird oft zu fx verkürzt. Churchs Notation hat sich im Laufe der Zeit irgendwie in die Informatik eingebürgert. Die Regeln, die wir hier aufstellen, bilden den einfach typisierten λ-Kalkül mit Erweiterungen um Produkte von zwei Faktoren und Summen von zwei Summanden.

Aussagen

$$\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash A}}{\vdash A \wedge B \to A}$$

Programmterme

$$\dfrac{\dfrac{\dfrac{}{t : A \times B \vdash t : A \times B}}{t : A \times B \vdash \pi_l(t) : A}}{\vdash (t \mapsto \pi_l(t)) : A \times B \to A}$$

Aussagen	**Programmterme**
$\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{\dfrac{A \wedge B \vdash A}{\vdash A \wedge B \to A}}$	$\dfrac{\dfrac{\dfrac{}{t : A \times B \vdash t : A \times B}}{t : A \times B \vdash \pi_l(t) : A}}{\vdash (t \mapsto \pi_l(t)) : A \times B \to A}$

Bemerkung. Anstelle von $t \mapsto \pi_l(t)$ kann man auch $(a, b) \mapsto a$ schreiben. Streng genommen muss hierbei allerdings ein unabweisbarer Musterabgleich durchgeführt werden. Ist t das Argument, wird t mit (a, b) abgeglichen, weshalb $a = \pi_l(t)$ sein muss.

Implementierung der Konstruktion

Der konstruierte Programmterm liefert den Beweis, dass der Typ $A \times B \rightarrow A$ ein bewohnter ist. Man kann den Term nun in Gallina formulieren und durch den Beweisassistent Coq prüfen lassen, ob die Konstruktion fehlerfrei durchgeführt wurde:

Implementierung der Konstruktion

Der konstruierte Programmterm liefert den Beweis, dass der Typ $A \times B \rightarrow A$ ein bewohnter ist. Man kann den Term nun in Gallina formulieren und durch den Beweisassistent Coq prüfen lassen, ob die Konstruktion fehlerfrei durchgeführt wurde:

Gallina

```
Definition proof1(A B: Type): A*B -> A
  := fun t => fst t.
```

Implementierung der Konstruktion

Der konstruierte Programmterm liefert den Beweis, dass der Typ $A \times B \rightarrow A$ ein bewohnter ist. Man kann den Term nun in Gallina formulieren und durch den Beweisassistent Coq prüfen lassen, ob die Konstruktion fehlerfrei durchgeführt wurde:

Gallina

```
Definition proof1(A B: Type): A*B -> A
  := fun t => fst t.
```

Es ginge auch so:

Gallina

```
Definition proof2(A B: Type): A*B -> A
  := fun t => match t with (a, b) => a end.
```

Ende.

November 2022