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 $a\colon A$ 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 $b\colon B$ 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

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

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

Funktion

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

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

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

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

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 $A \times B \to A$ 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 $A \times B \to A$ 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 $A \times B \to A$ 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.
```

Aussagen bekommen eigentlich den speziellen Typ `Prop`. Infolgedessen verändern sich Syntax und genutzte Funktionen, wobei die Mechanismen aber äquivalent bleiben.

Gallina

```
Definition proof3 (A B: Prop): A /\ B -> A
  := fun h => proj1 h.
```

Aussagen bekommen eigentlich den speziellen Typ `Prop`. Infolgedessen verändern sich Syntax und genutzte Funktionen, wobei die Mechanismen aber äquivalent bleiben.

Gallina

```
Definition proof3 (A B: Prop): A /\ B -> A
  := fun h => proj1 h.
```

Auch hier ist eine Zerlegung per Musterabgleich durchführbar:

Gallina

```
Definition proof4 (A B: Prop): A /\ B -> A
  := fun h => match h with conj a b => a end.
```

Aussagen bekommen eigentlich den speziellen Typ `Prop`. Infolgedessen verändern sich Syntax und genutzte Funktionen, wobei die Mechanismen aber äquivalent bleiben.

Gallina

```
Definition proof3 (A B: Prop): A /\ B -> A
  := fun h => proj1 h.
```

Auch hier ist eine Zerlegung per Musterabgleich durchführbar:

Gallina

```
Definition proof4 (A B: Prop): A /\ B -> A
  := fun h => match h with conj a b => a end.
```

Äquivalent zum Musterabgleich ist das Induktionsprinzip der Konjunktion:

Gallina

```
Definition proof5 (A B: Prop): A /\ B -> A
  := fun h => and_ind (fun a b => a) h.
```

Der Beweis einer Konjunktion wird hier in seine beiden Teile zerlegt, um einen oder beide nutzen zu können. Die Zerlegung wird durch eine Rückruffunktion dargestellt, die `and_ind` als erstes Argument bekommt.

Taktiken

Als Alternative zur direkten Formulierung von Termen bietet Coq außerdem noch Taktiken an, darunter versteht man Prozeduren zur automatischen Erstellung von Teilbeweisen.

Als Alternative zur direkten Formulierung von Termen bietet Coq außerdem noch Taktiken an, darunter versteht man Prozeduren zur automatischen Erstellung von Teilbeweisen.

Die Erstellung des Beweisbaums verläuft hierbei rückwärts. Man arbeitet sich also von der Wurzel aus bis zu den Blättern durch. Ein noch unbestätigter Knoten stellt in diesem Zusammenhang ein zu erreichendes *Ziel* dar, dessen Kindknoten die *Unterziele*.

Es stehen Basistaktiken zur Verfügung, die im Prinzip die Schlussregeln des natürlichen Schließens widerspiegeln.

Taktik zum Axiom

Regel	**Taktik**
$\dfrac{}{\Gamma, h : A, \Gamma' \vdash h : A}$	`exact h`

Taktiken zu den Einführungsregeln

Regel	**Taktik**
$\dfrac{\Gamma, h : A \vdash ? : B}{\Gamma \vdash ? : A \to B}$	`intro h`
$\dfrac{\Gamma \vdash ? : A \qquad \Gamma \vdash ? : B}{\Gamma \vdash ? : A \land B}$	`split`
$\dfrac{\Gamma \vdash ? : A}{\Gamma \vdash ? : A \lor B}$	`left`
$\dfrac{\Gamma \vdash ? : B}{\Gamma \vdash ? : A \lor B}$	`right`

Taktiken zu den Beseitigungsregeln

Regel	**Taktik**
$$\frac{\Gamma \vdash h : A \to B \qquad \Gamma \vdash ? : A}{\Gamma \vdash ? : B}$$	`apply h`
$$\frac{\Gamma, a : A, b : B, \Gamma' \vdash ? : C}{\Gamma, h : A \land B, \Gamma' \vdash ? : C}$$	`destruct h as (a, b)`

Um $A \wedge B \to A$ zu zeigen, können wir also wie folgt vorgehen:

$$\dfrac{\dfrac{\overline{a\colon A, b\colon B \vdash a\colon A}\ \texttt{exact a}}{h\colon A \wedge B \vdash a\colon A}\ \texttt{destruct h as (a, b)}}{\vdash (h \mapsto a)\colon A \wedge B \to A}\ \texttt{intro h}$$

Um $A \wedge B \rightarrow A$ zu zeigen, können wir also wie folgt vorgehen:

$$\dfrac{\dfrac{\dfrac{}{a\colon A, b\colon B \vdash a\colon A}\;\texttt{exact a}}{h\colon A \wedge B \vdash a\colon A}\;\texttt{destruct h as (a, b)}}{\vdash (h \mapsto a)\colon A \wedge B \rightarrow A}\;\texttt{intro h}$$

Die Implementierung:

Gallina

```
Theorem projl (A B: Prop): A /\ B -> A.
Proof.
  intro h.
  destruct h as (a, b).
  exact a.
Qed.
```

Um $(A \to B) \wedge A \to B$ zu zeigen, können wir wie folgt vorgehen:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \to B \vdash f : A \to B} \quad \dfrac{}{a : A \vdash a : A}\ \texttt{exact a}}{f : A \to B, a : A \vdash f(a) : B}\ \texttt{apply f}}{h : (A \to B) \wedge A \vdash f(a) : B}\ \texttt{destruct h as (f, a)}}{\vdash (h \mapsto f(a)) : (A \to B) \wedge A \to B}\ \texttt{intro h}$$

Um $(A \to B) \land A \to B$ zu zeigen, können wir wie folgt vorgehen:

$$\dfrac{\dfrac{\dfrac{\overline{f\colon A \to B \vdash f\colon A \to B} \quad \overline{a\colon A \vdash a\colon A}\ \texttt{exact a}}{f\colon A \to B, a\colon A \vdash f(a)\colon B}\ \texttt{apply f}}{h\colon (A \to B) \land A \vdash f(a)\colon B}\ \texttt{destruct h as (f, a)}}{\vdash (h \mapsto f(a))\colon (A \to B) \land A \to B}\ \texttt{intro h}$$

Die Implementierung:

Gallina

```
Theorem mp (A B: Prop): (A -> B) /\ A -> B.
Proof.
  intro h.
  destruct h as (f, a).
  apply f.
  exact a.
Qed.
```

Um $A \wedge B \to B \wedge A$ zu zeigen, können wir wie folgt vorgehen:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{b : B \vdash b : B}\ \texttt{exact b} \qquad \cfrac{}{a : A \vdash a : A}\ \texttt{exact a}}{a : A, b : B \vdash \mathrm{conj}(b)(a) : B \wedge A}\ \texttt{split}}{h : A \wedge B \vdash \mathrm{conj}(b)(a) : B \wedge A}\ \texttt{destruct h as (a, b)}}{\vdash (h \mapsto \mathrm{conj}(b)(a)) : A \wedge B \to B \wedge A}\ \texttt{intro h}$$

Um $A \land B \to B \land A$ zu zeigen, können wir wie folgt vorgehen:

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{b : B \vdash b : B}\;\texttt{exact b} \qquad \dfrac{}{a : A \vdash a : A}\;\texttt{exact a}}{a : A, b : B \vdash \mathrm{conj}(b)(a) : B \land A}\;\texttt{split}}{h : A \land B \vdash \mathrm{conj}(b)(a) : B \land A}\;\texttt{destruct h as (a, b)}}{\vdash (h \mapsto \mathrm{conj}(b)(a)) : A \land B \to B \land A}\;\texttt{intro h}
$$

Die Implementierung:

Gallina

```
Theorem conjunction_commutes (A B: Prop): A /\ B -> B /\ A.
Proof.
  intro h.
  destruct h as (a, b).
  split.
  - exact b.
  - exact a.
Qed.
```

Um $A \lor B \to B \lor A$ zu zeigen, können wir wie folgt vorgehen:

$$\dfrac{\dfrac{\dfrac{}{h : A \lor B \vdash h : A \lor B} \quad \dfrac{\dfrac{}{a : A \vdash a : A}}{a : A \vdash r(a) : B \lor A} \quad \dfrac{\dfrac{}{b : B \vdash b : B}\ \texttt{exact b}}{b : B \vdash l(b) : B \lor A}\ \texttt{left}}{h : A \lor B \vdash s : B \lor A}\ \texttt{destruct h as [a | b]}}{\vdash (h \mapsto s) : A \lor B \to B \lor A}\ \texttt{intro h}$$

mit $l :=$ or_introl, $r :=$ or_intror und

$$s := \textbf{match}\ h \begin{cases} l(a) \mapsto r(a), \\ r(b) \mapsto l(b). \end{cases}$$

Um $A \lor B \to B \lor A$ zu zeigen, können wir wie folgt vorgehen:

$$\dfrac{\dfrac{}{h\colon A \lor B \vdash h\colon A \lor B} \quad \dfrac{\dfrac{}{a\colon A \vdash a\colon A}}{a\colon A \vdash r(a)\colon B \lor A} \quad \dfrac{\dfrac{}{b\colon B \vdash b\colon B}\ \texttt{exact b}}{b\colon B \vdash l(b)\colon B \lor A}\ \texttt{left}}{\dfrac{h\colon A \lor B \vdash s\colon B \lor A}{\vdash (h \mapsto s)\colon A \lor B \to B \lor A}\ \texttt{intro h}}\ \texttt{destruct h as [a | b]}$$

mit $l :=$ or_introl, $r :=$ or_intror und

$$s := \textbf{match}\ h \begin{cases} l(a) \mapsto r(a), \\ r(b) \mapsto l(b). \end{cases}$$

Die Implementierung:

Gallina

```
Theorem disjunction_commutes (A B: Prop): A \/ B -> B \/ A.
Proof.
  intro h.
  destruct h as [a | b].
  - right. exact a.
  - left. exact b.
Qed.
```

Nun wird man irgendwann höhre kognitive Sprünge ausführen und mühseligen Kleinkram auslassen wollen. Hierfür stehen höhre Taktiken zur Verfügung, die einfache Beweise gänzlich automatisch konstruieren. Ein Beispiel hierfür ist `tauto`.

Nun wird man irgendwann höhre kognitive Sprünge ausführen und mühseligen Kleinkram auslassen wollen. Hierfür stehen höhre Taktiken zur Verfügung, die einfache Beweise gänzlich automatisch konstruieren. Ein Beispiel hierfür ist `tauto`.

Gallina

```
Theorem projl (A B: Prop): A /\ B -> A.
Proof.
  tauto.
Qed.
```

Nun wird man irgendwann höhre kognitive Sprünge ausführen und mühseligen Kleinkram auslassen wollen. Hierfür stehen höhre Taktiken zur Verfügung, die einfache Beweise gänzlich automatisch konstruieren. Ein Beispiel hierfür ist `tauto`.

Gallina

```
Theorem projl (A B: Prop): A /\ B -> A.
Proof.
  tauto.
Qed.
```

Gut, das wäre im ersten Semester nicht gestattet. Nun kann man trotzdem Schummeln, indem man sich den erzeugten Beweis einfach anschaut:

Gallina

```
Print projl.

(* Ausgabe: *)
projl =
fun (A B: Prop) (H: A /\ B) => and_ind (fun (H0: A) (_: B) => H0) H
     : forall A B: Prop, A /\ B -> A
```

Für $(A \wedge \forall x.\, P(x)) \rightarrow \forall x.\, A \wedge P(x)$ findet sich:

Für $(A \land \forall x.\, P(x)) \to \forall x.\, A \land P(x)$ findet sich:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a : A \vdash a : A}\ \texttt{exact a} \qquad \dfrac{}{f : \forall x.\, P(x), x : \mathrm{Type} \vdash f(x)}\ \texttt{apply f}}{a : A, f : \forall x.\, P(x), x : \mathrm{Type} \vdash y : A \land P(x)}\ \texttt{split}}{a : A, f : \forall x.\, P(x) \vdash (x \mapsto y) : \forall x.\, A \land P(x)}\ \texttt{intro x}}{h : A \land \forall x.\, P(x) \vdash (x \mapsto y) : \forall x.\, A \land P(x)}\ \texttt{destruct h as (a, f)}}{\vdash (h \mapsto x \mapsto y) : (A \land \forall x.\, P(x)) \to \forall x.\, A \land P(x)}\ \texttt{intro h}$$

Hierbei gilt $y := \mathrm{conj}(a)(f(x))$. Die Implementierung:

Für $(A \land \forall x.\, P(x)) \to \forall x.\, A \land P(x)$ findet sich:

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{a : A \vdash a : A}\;\texttt{exact a}
        \qquad
        \dfrac{}{f : \forall x.\, P(x), x : \mathrm{Type} \vdash f(x)}\;\texttt{apply f}
      }{a : A, f : \forall x.\, P(x), x : \mathrm{Type} \vdash y : A \land P(x)}\;\texttt{split}
    }{a : A, f : \forall x.\, P(x) \vdash (x \mapsto y) : \forall x.\, A \land P(x)}\;\texttt{intro x}
  }{h : A \land \forall x.\, P(x) \vdash (x \mapsto y) : \forall x.\, A \land P(x)}\;\texttt{destruct h as (a, f)}
}{\vdash (h \mapsto x \mapsto y) : (A \land \forall x.\, P(x)) \to \forall x.\, A \land P(x)}\;\texttt{intro h}
$$

Hierbei gilt $y := \mathrm{conj}(a)(f(x))$. Die Implementierung:

Gallina

```
Theorem const_factor (A B: Prop) (P: Type -> Prop):
  A /\ (forall x, P x) -> forall x, A /\ P x.
Proof.
  intro h.
  destruct h as (a, f).
  intro x.
  split.
  - exact a.
  - apply f.
Qed.
```

Literatur

- Christine Paulin-Mohring: *Introduction to the Coq proof-assistant for practical software verification*. Laboratoire de recherche en informatique, Université Paris-Saclay, 2011. Link (Open Access).
- Benjamin C. Pierce u. a.: *Software Foundations. Volume 1: Logical Foundations.* University of Pennsylvania, 2022. Link (Open Access).

Ende.

November 2022