# Einführung in die konstruktive Logik

Andreas Abel

Oktober 2007

# Kapitel 1

# Aussagenlogik

## 1.1 Motivation

**Disjunktionseigenschaft.** Wenn Aussage $A \vee B$ gilt, wüsste man gerne, ob nun $A$ oder $B$ wahr ist. Im Allgemeinen kann einem in der klassischen Logik dieser Wunsch jedoch nicht erfüllt werden. Das Gesetz vom ausgeschlossenen Dritten (lat. *tertium non datur* (TND), engl. *law of the excluded middle*) $A \vee \neg A$ gilt jedoch für beliebige Aussagen $A$, ohne dass man einen Beweis für $A$ oder sein Gegenteil hätte. Spekulation: Das entspricht philosophisch der Postulation eines logisch allwissenden Gottes.

Man kann sich jedoch auch auf den Standpunkt des *Konstruktivisten* stellen: Wahr ist nur, was bewiesen ist. Dann erfordert die Wahrheit von $A \vee \neg A$ entweder einen Beweis von $A$ oder von $\neg A$, also kann TND nicht allgemein gelten. Während die klassische Logik bereits im Altertum formalisiert wurde (Aristoteles), ist die konstruktive (oder *intuitionistische*) Logik eine Entwicklung des 20. Jahrhunderts, angestoßen durch die große Grundlagenkrise der Mathematik um 1900.

## 1.2 Brouwer-Heyting-Kolmogorov-Interpretation

Stellen wir uns nun auf den intuitionistischen Standpunkt. Was gilt nun als Beweis einer Aussage $A$? Kolmogorov [Kol32] sieht eine Aussage als Aufgabenstellung an und einen Beweis als eine Vorschrift, wie die Lösung zu erlangen ist.

- Ein Beweis von $A \wedge B$ ist ein Paar $(p, q)$, wobei $p$ ein Beweis von $A$ und $q$ ein Beweis von $B$ ist.

- Ein Beweis von $A \vee B$ ist ein Paar $(b, p)$, wobei $b$ ein Bit ist und $p$ ein Beweis von $A$, falls $b = 0$, und ein Beweis von $B$, falls $b = 1$.

- Ein Beweis von $A \Rightarrow B$ ist ein Programm $f$, dass einen beliebigen Beweis von $A$ in einen Beweis von $B$ transformiert.

- Ein Beweis von $\neg A$ ist ein Programm $f$, dass einen beliebigen Beweis von $A$ in einen Beweis von $\perp$ transformiert.

- Der Beweis von $\top$ ist trivial.

Diese Definition kann auch als Vorschrift zur Vereinfachung von Aufgabenstellungen gelesen werden, z. B., “um die Aufgabe $A \wedge B$ zu lösen, löse sowohl $A$ als auch $B$”, “um die Aufgabe $A \vee B$ zu lösen, entscheide dich, entweder $A$ oder $B$ zu lösen”. Die Aufgabe $\top$ kann als schon gelöst betrachtet werden. Die Aufgabe $\perp$ kann nicht weiter vereinfacht werden, d.h., es gibt keinen kanonischen Beweis von $\perp$.

Mehr zur BHK-Interpretation unter “Beweisterme”.

Referenz: Andrei Kolmogorov, *Zur Deutung der intuitionistischen Logik* (1932) [Kol32].

## 1.3 Natürliches Schließen

Engl. natural deduction (ND).

**Aussagen:**

$$A, B, C ::= P \mid \top \mid \perp \mid A \wedge B \mid A \vee B \mid A \Rightarrow B.$$

Die Negation $\neg A$ ist definiert als $A \Rightarrow \perp$.

**Der Kalkül des Natürlichen Schließens**

$$\dfrac{\begin{array}{c}\overline{A}\\ \vdots\\ B\end{array}}{A \Rightarrow B}\,\Rightarrow\mathsf{I} \qquad \dfrac{A \Rightarrow B \qquad A}{B}\,\Rightarrow\mathsf{E}$$

$$\dfrac{A \qquad B}{A \wedge B}\,\wedge\mathsf{I} \qquad \dfrac{A \wedge B}{A}\,\wedge\mathsf{E}_1 \qquad \dfrac{A \wedge B}{B}\,\wedge\mathsf{E}_2$$

$$\dfrac{A}{A \vee B}\,\vee\mathsf{I}_1 \qquad \dfrac{B}{A \vee B}\,\vee\mathsf{I}_2$$

$$\dfrac{A \vee B \qquad \begin{array}{c}\overline{A}\\ \vdots\\ C\end{array} \qquad \begin{array}{c}\overline{B}\\ \vdots\\ C\end{array}}{C}\,\vee\mathsf{E}$$

$$\dfrac{}{\top}\,\top\mathsf{I} \qquad \dfrac{\bot}{C}\,\bot\mathsf{E}$$

Dabei entsprechen die Einführungsregeln (*introduction rules*) mit Namenssuffix $\mathsf{I}$ den BHK-Konstruktionsprinzipien für Beweise. Die Eliminationsregeln (*elimination rules*) mit Suffix $\mathsf{E}$ sind nötig, um hypothetische Beweise zu zerlegen, um z.B. aus einem angenommenen Beweis von $A \wedge B$ einen Beweis von $A$ zu machen. Einen Beweis mit einer $\mathsf{I}$-Regel zu konstruieren und unmittelbar danach mit einer $\mathsf{E}$-Regel wieder zu zerlegen, stellt einen Umweg dar, und kann abgekürzt werden (siehe Beweisreduktionen).

## 1.4 Natürliches Schließen in Tutch

Tutch (*tut*orial proof *ch*ecker) ist ein Beweisprüfer für natürliches Schließen zur Übungszwecken. Download und Dokumentation siehe:

`http://www.tcs.ifi.lmu.de/~abel/tutch`

Tutch ist in SML implementiert und benötigt einen aktuellen Standard ML of New Jersey Compiler zur Installation. Entwicklung eines Beweises in Tutch: Tutch kann auch unvollständige Beweise prüfen, meldet dann aber einen Fehler der auf die Lücke zeigt. Dieses Feature machen wir uns zunutze und entwickeln einen Beweis inkrementell:

```
proof andI : A => B => A & B =
begin
```

| Formel | in Tutch |
|---|---|
| $\top$ | `T` |
| $\bot$ | `F` |
| $A \wedge B$ | `A & B` |
| $A \vee B$ | `A \| B` |
| $A \Rightarrow B$ | `A => B` |
| $\neg A$ | `~ A` |
| $A \Leftrightarrow B$ | `A <=> B` |

Tabelle 1.1: Propositionale Formeln in Tutch.

```
  A => B => A & B
end;
```

Dies ist die minimale Eingabe, die Tutch verarbeiten kann. `proof`, `begin`, `end` sind Schlüsselwörter. Wir entwickeln einen Beweis der Aussage `A => B => A & B` und geben ihm den Namen `andI`. Zwischen `begin` und `end` steht der Beweis als Liste von Aussagen, die letzte Aussage muss mit der zu beweisenden übereinstimmen.

Nach Aufruf von `tutch andI.tut` erhalten wir folgendes Ergebnis:

```
TUTCH 0.52 beta, $Date: 2002/10/24 19:25:49 $
[Opening file andI.tut]
Proving andI: A => B => A & B ...
andI.tut:4.3-4.18 Error:
Unjustified line .  |-  A => B => A & B
Assuming this line, checking remainder...
Proof incomplete
[Closing file andI.tut]
```

Die Datei ist also syntaktisch korrekt, aber der Beweis ist unvollständig. Um die Implikation `A => (B => A & B)` zu beweisen, nehmen wir `A` an und beweisen unter dieser Annahme `B => A & B`. In Tutch gibt es dafür die eckigen Klammern: Die erste Aussage nach der öffnenden Klammer ist die Annahme, die letzte Aussage vor der schliessenden Klammer die Konklusion.

```
proof andI: A => B => A & B =
begin
[ A;

  B => A & B ];
A => B => A & B
end;
```

Nun meldet Tutch:

```
Unjustified line A  |-  B => A & B
```

Weiter nehmen wir `B` an und beweisen `A & B`:

```
proof andI: A => B => A & B =
begin
[ A;
  [ B;
    A & B ];
  B => A & B ];
A => B => A & B
end;
```

Der Beweis ist fertig, da `A & B` aus den sichtbaren, darüberliegenden Aussagen durch Anwendung einer einzigen Regel, Und-Einführung, folgt. Mit `tutch -v andI.tut` erhalten wir ein Protokoll der Beweisprüfung:

```
Proving andI: A => B => A & B ...
  1  [ A;
  2    [ B;
  3      A & B ];             by AndI 1 2
  4    B => A & B ];          by ImpI 3
  5  A => B => A & B          by ImpI 4
QED
```

Tutch hat jede Aussage (außer den Annahmen) mit der Regel annotiert, die die Aussage beweist, mit den Nummern der Aussagen, die an diesem Schritt direkt beteiligt sind.

## 1.5 Natürliches Schließen mit expliziten Annahmen

Besonders wenn man etwas über den Kalkül des natürlichen Schließens zeigen möchte, ist es gut, die aktuell sichtbaren Annahmen zu jeder Zeit explizit zu nennen. Damit ergeben sich folgende Regeln:

**Der Kalkül des Natürlichen Schließens mit expliziten Annahmen** $\Gamma \vdash A$. $\Gamma$ sei eine Menge von Aussagen, die aktuellen Annahmen (Hypothesen).

$$\frac{A \in \Gamma}{\Gamma \vdash A}\,\mathsf{hyp} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\Rightarrow\!\mathsf{I} \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\Rightarrow\!\mathsf{E}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\wedge\mathsf{I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\wedge\mathsf{E}_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\wedge\mathsf{E}_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\vee\mathsf{I}_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\vee\mathsf{I}_2$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\vee\mathsf{E}$$

$$\frac{}{\Gamma \vdash \top}\top\mathsf{I} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash C}\bot\mathsf{E}$$

Es ist eine Regel $\mathsf{hyp}$ hinzu gekommen, die besagt: Jede Annahme gilt als bewiesen. Die Regeln $\Rightarrow\!\mathsf{I}$ und $\vee\mathsf{E}$ fügen neue Annahmen hinzu (liest man die Regeln von unten nach oben), alle anderen Regeln lassen $\Gamma$ unberührt.

## 1.6 Beweisterme

Nach der BHK-Interpretation ist ein Beweis ein Programm zur Lösung einer Aufgabe. Im Folgenden formalisieren wir diese Programme.

**Beweisterme (Church-Stil):**

| | |
|---|---|
| $r, s, t ::= x \mid \lambda x^A.t \mid r\,s$ | einfach getypter $\lambda$-Kalkül (für $\Rightarrow$) |
| $\mid (r, s) \mid \mathsf{fst}\,r \mid \mathsf{snd}\,r$ | Paare und Projektionen (für $\wedge$) |
| $\mid \mathsf{inl}_B\,t \mid \mathsf{inl}_A\,t$ | Injektionen und... |
| $\mid \mathsf{case}\,r\ \mathsf{of}\ \mathsf{inl}\,x^A \Rightarrow s \mid \mathsf{inr}\,y^B \Rightarrow t$ | ...Fallunterscheidung (für $\vee$) |
| $\mid ()$ | leeres Tupel (für $\top$) |
| $\mid \mathsf{abort}_C\,r$ | Ausnahme (*exception*) (für $\bot$) |

**Typisierung:** $\Gamma$ sei eine Menge von Variable-Aussage-Paaren, geschrieben $x : A$. Wir definieren induktiv ein Urteil $\Gamma \vdash t : C$ das besagt: *Unter den Annahmen*

$\Gamma$ *ist der Term* $t$ *ein gültiger Beweis der Aussage* $C$*.*

$$\frac{(x{:}A) \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{hyp}$$

$$\frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \lambda x^A .\, t : A \Rightarrow B}\,{\Rightarrow}\mathsf{I} \qquad \frac{\Gamma \vdash r : A \Rightarrow B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B}\,{\Rightarrow}\mathsf{E}$$

$$\frac{\Gamma \vdash r : A \qquad \Gamma \vdash s : B}{\Gamma \vdash (r,\, s) : A \wedge B}\,{\wedge}\mathsf{I} \qquad \frac{\Gamma \vdash r : A \wedge B}{\Gamma \vdash \mathsf{fst}\, r : A}\,{\wedge}\mathsf{E}_1 \qquad \frac{\Gamma \vdash r : A \wedge B}{\Gamma \vdash \mathsf{snd}\, r : B}\,{\wedge}\mathsf{E}_2$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{inl}_B\, t : A \vee B}\,{\vee}\mathsf{I}_1 \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \mathsf{inr}_A\, t : A \vee B}\,{\vee}\mathsf{I}_2$$

$$\frac{\Gamma \vdash r : A \vee B \qquad \Gamma, x{:}A \vdash s : C \qquad \Gamma, y{:}B \vdash t : C}{\Gamma \vdash \mathsf{case}\, r\ \mathsf{of}\ \mathsf{inl}\, x^A \Rightarrow s \ \mid\ \mathsf{inr}\, y^B \Rightarrow t : C}\,{\vee}\mathsf{E}$$

$$\frac{}{\Gamma \vdash () : \top}\,\top\mathsf{I} \qquad \frac{\Gamma \vdash r : \bot}{\Gamma \vdash \mathsf{abort}_C\, r : C}\,\bot\mathsf{E}$$

Die Propositions-Annotationen in den Beweistermen zu $\Rightarrow$I und $\bot$E sind so gewählt, dass man die bewiesene Proposition $C$ zu einem Beweisterm $t$ durch strukturelle Rekursion über $t$ mühelos reproduzieren kann.

**Übung 1** Gegeben ein Typisierungskontext $\Gamma$ und ein Beweisterm $t$ im Church-Stil berechnet die Funktion $\mathsf{typeOf}(\Gamma \vdash t)$ die von $t$ bewiesene Aussage $A$ durch Rekursion über $t$. Falls $t$ nicht wohlgetypt ist in $\Gamma$, wird $\emptyset$ zurückgegeben. Definieren Sie typeOf, in Anlehnung an obige Regeln.

In den Einführungsregeln (I, engl. *introduction*) erkennen wir die Definition eines kanonischen Beweises nach Brouwer-Heyting-Kolmogorov wieder. Es gibt keinen kanonischen Beweis der Absurdität, also keine Regel $\bot$I. Die Beseitigungsregeln (E, engl. *elimination*) erklären, wie man vorhandene Beweise verwenden kann, um die Beweisaufgabe zu lösen. Auf den trivialen Beweis von $\top$ hat man keine Arbeit verwendet, kann ihn also auch nicht sinnvoll benutzen, deshalb keine Regel $\top$E.

## 1.7 Beweisterme in Tutch

In Tutch werden Beweisterme sehr ähnlich geschrieben, es gibt die folgenden Unterschiede:

1. Keine Propositions-Annotationen.

2. $\lambda x.t$ wird geschrieben als `fn x => t` (wie in SML).

3. Fallunterscheidung wird mit `end` abgeschlossen, also

```
case r of inl x => s | inr y => t end
```

Ein Tutch-Beweis kann mit Beweistermen annotiert werden, die von Tutch dann auf Korrektheit geprüft werden. Z.B.

```
annotated proof andI : A => B => A & B =
begin
[ x : A;
  [ y : B;
    (x,y) : A & B ];
  fn y => (x,y) : B => A & B ];
fn x => fn y => (x,y) : A => B => A & B
end;
```

Das Schlüsselwort `annotated` zeigt an, dass nun jede Aussage einen Beweisterm erfordert. Annahmen werden mit Variablen annotiert, hergeleitete Aussagen mit einem zusammengesetzten Term. Dabei ist jeder Term ein voller Beweis für die dahinterstehende Aussage, dadurch ergibt sich eine gewaltige Redundanz: Jeder Teilbeweis muss vollständig wiederholt werden.

## 1.8 Coq

Das Beweissystem Coq basiert auf einem Kalkül ähnlich dem natürlichen Schließen.

```
Section Example1.

Variables (A B C: Prop).

Theorem currying : and A B -> C <-> A -> B -> C.
split.
 intros f a b.
 apply f.
 split. assumption. assumption.
intros f ab.
inversion ab.
apply f. assumption. assumption.
Qed.

Theorem orElim : or A B -> C <-> and (A -> C) (B -> C).
split.
 intros f.
 split.
  intros a. apply f. left. exact a.
 intros b. apply f. right. exact b.
intros gh d.
inversion_clear gh as [ g h ].
inversion_clear d as [ a | b ].
 apply g. assumption.
apply h. assumption.
Qed.

End Example1.
```

Eine Coq-Datei kann mehrere `Section`s enthalten, die jeweils lokale Annahmen (z.B: `Variables`) deklarieren dürfen. Das ist dem `theory`-Mechanismus von PVS ähnlich. Die Bibliothek `Coq.Init.Logic`, die automatisch geladen wird, enthält die Definition der logischen Konnektive. `Prop` ist der Typ der Aussagen (war `boolean` in PVS).

Tabelle 1.3 zeigt die den Beweisregeln des natürlichen Schließens entsprechende Coq-Taktiken. Die Entsprechung ist nicht eins-zu-eins, die `inversion`-Taktik entspricht eher einer Links-Regel des Sequenzenkalküls denn einer Eliminationsregel des natürlichen Schließens. Varianten:

- `intros x y z` führt gleich 3 Hypothesen ein.
- `apply f` führt auch mehrere $\Rightarrow$E-Schritte auf einmal aus, z.B., falls $f : A \Rightarrow B \Rightarrow C$ und das aktuelle Ziel $C$ ist, dann werden zwei neue Ziele $A$

| Formel | in Coq | Alternative |
|---|---|---|
| $\top$ | `True` | |
| $\bot$ | `False` | |
| $A \wedge B$ | `A /\ B` | `and A B` |
| $A \vee B$ | `A \/ B` | `or A B` |
| $A \Rightarrow B$ | `A -> B` | |
| $\neg A$ | `~ A` | `not A` |
| $A \Leftrightarrow B$ | `A <-> B` | `iff A B` |

Tabelle 1.2: Propositionale Formeln in Coq.

| Regel | Coq-Taktik | Alternative |
|---|---|---|
| hyp | `exact x` | `assumption` |
| $\Rightarrow$I | `intros x` | |
| $\Rightarrow$E | `apply r` | |
| $\top$I | `split` | |
| $\wedge$I | `split` | |
| $\wedge$E | `inversion x` | `elim x` |
| $\vee$I$_1$ | `left` | |
| $\vee$I$_2$ | `right` | |
| $\vee$E | `inversion x` | `elim x` |
| $\bot$E | `inversion x` | `elim x` |
| weak | `clear x` | |

Tabelle 1.3: Natürliches Schließen in Coq.

und $B$ erzeugt.

- `split` kann auch angewendet werden, falls das Ziel von der Form $A_1 \to \cdots \to A_n \to C$ ist und $C$ eine Konjuktion. Dann werden vor der Aufsplittung die Aussagen $A_i$ als Hypotheses eingeführt. Dasselbe gilt analog für `left` und `right`.

- `inversion_clear h` zerlegt wie `inversion h` die Hypothese $h$ in ihre Bestandteile, zusätzlich wird noch $h$ entfernt. Beide Taktiken kann noch eine Liste von Bezeichnern `[ x1 ... xn ]` für die neuen Hypothesen mitgeliefert werden. Entstehen durch die Zerlegung mehrere Unterziele (wie z.B. bei $\vee\mathsf{E}$), so müssen Bezeichner für jedes Unterziel mitgeliefert werden. Die Bezeichnerliste wird dann durch `|` entsprechend partitioniert.

- `elim x` kann auch auf Hypothesen der Form $A_1 \to \cdots \to A_n \to C$ und $C$ Konjunktion, Disjunktion oder Falsum. Dann werden die $A_i$ als neue Ziele hinzugefügt.

- `cut (A)` fügt die Annahme $A$ hinzu, die dann separat noch bewiesen werden muss.

**Beweisterme.** Wurde `Theorem bla : A` in Coq bewiesen, so kann der Beweisterm mit `Print bla` (oder auch `Check bla`) eingesehen werden. In Coq können Beweisterme auch direkt eingegeben werden. Hier haben wir einige Aussagen mit Termen bewiesen, u.a. die des letzten Beispiels:

```
Section Example2.

Variables (A B C: Prop).

Definition curry  : (and A B -> C) -> A -> B -> C
  := fun f a b => f (conj a b).

Definition curry'
  := fun (f : and A B -> C) (a : A) (b : B) => f (conj a b).

Definition uncurry : (A -> B -> C) -> and A B -> C
  := fun f p => f (proj1 p) (proj2 p).

Definition join :  and (A -> C) (B -> C) -> or A B -> C
  := fun gh d =>
     match d with
     | or_introl a => proj1 gh a
     | or_intror b => proj2 gh b
     end.

Definition fsplit : (or A B -> C) -> and (A -> C) (B -> C)
```

```
  := fun f => conj (fun a => f (or_introl B a))
                   (fun b => f (or_intror A b)).

Definition abort : False -> C
  := fun h => match h with end.

Definition delay : C -> (True -> C)
  := fun c i => c.
Definition force : (True -> C) -> C
  := fun f => f I.

End Example2.
```

Die `Definition` eines Beweistermes kann entweder die zu beweisende Proposition mit angeben (Bsp. `curry`) oder vom System inferieren lassen (Bsp. `uncurry`). Dann müssen aber die Aussagen zu den (wenigstens einigen) Hypothesenvariablen angegeben werden. (Die Annotation der Variable `a` und `b` ist hier überflüssig.) Besondere Vorsicht ist bei `or_introl` und `or_intror` geboten: Sie

| Beweisterm | in Coq |
|---|---|
| $\lambda x^A.\, t$ | `fun (x : A) => t` |
| $r\, s$ | `r s` |
| $()$ | `I` |
| $(r, s)$ | `conj r s` |
| $\mathsf{fst}\, r$ | `proj1 r` |
| $\mathsf{snd}\, r$ | `proj2 r` |
| $\mathsf{inl}\, r : A \vee B$ | `or_introl B r` |
| $\mathsf{inr}\, r : A \vee B$ | `or_intror A r` |
| $\mathsf{case}\, r\ \mathsf{of}\ \mathsf{inl}\, x^A \Rightarrow s \mid \mathsf{inr}\, y^B \Rightarrow t$ | |
| `match r with \| or_introl x => s \| or_intror y => t end` | |
| $\mathsf{abort}_C\, r$ | `match r return C with end` |

Tabelle 1.4: Beweisterme in Coq.

benötigen als erstes Argument die Aussage der Disjunktion, die *nicht* vom Beweisterm (zweites Argument) beweisen wird. In einem `match`-Ausdruck muss dieses zusätzliche Argument aber wegelassen werden. Die $\perp$E-Term abort wird als Fallunterscheidung über 0 Alternativen repräsentiert. Die durch ein `match` zu beweisende Aussage `C` kann immer mit `return C` angegeben werden, muss aber nicht.