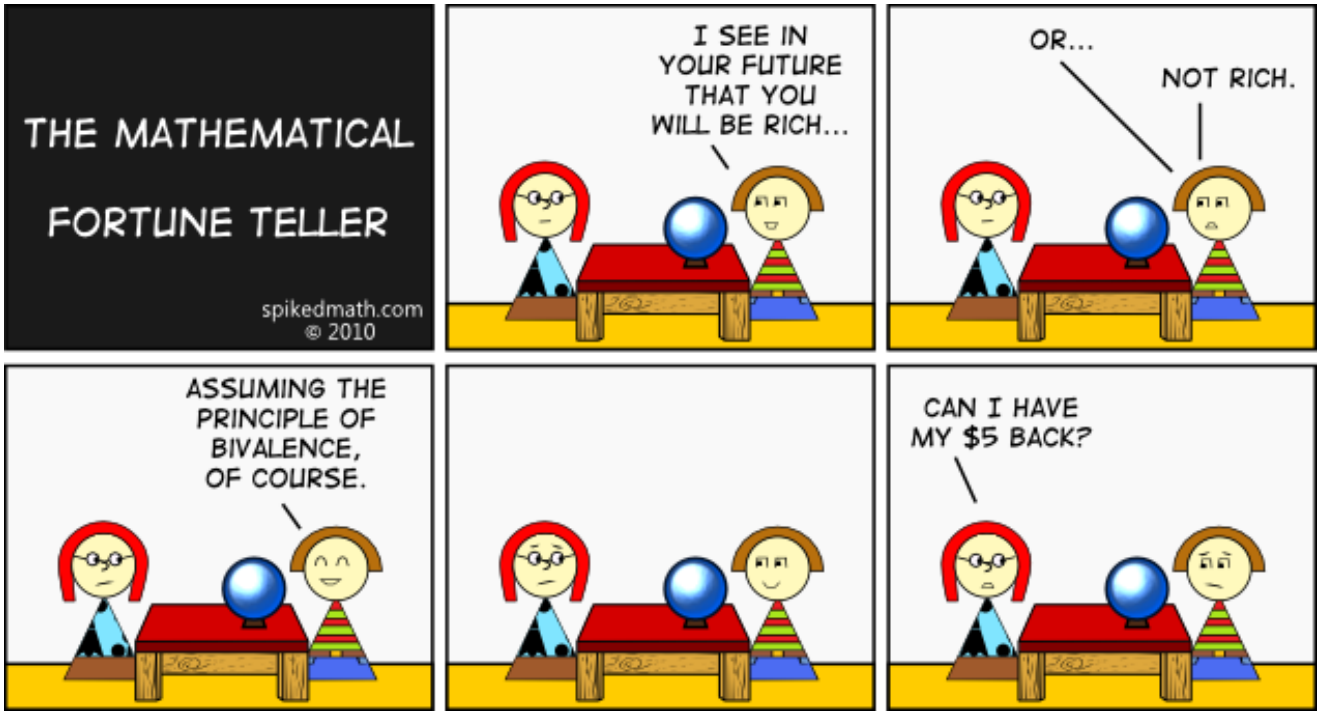


Konstruktive Mathematik, die Doppelnegationsübersetzung und Continuations

Ingo Blechschmidt
Universität Augsburg

Haskell in Leipzig
4. Dezember 2015

Gliederung

Ein Märchen über klassische Logik

Erzähler. Vor langer, langer Zeit begab sich im fernen, fernen Curryland folgende Geschichte. Eines Tages holte die Königin des Landes und aller Haskellistas und Lambdroiden ihren Haus- und Hof-Philosophen zu sich.

Königin. Philosoph! Ich habe folgenden Auftrag an dich: Beschaffe mir den Stein der Weisen, oder alternativ finde heraus, wie man mithilfe des Steins unbegrenzt Gold herstellen kann!

Philosoph. Aber meine Königin! Ich habe nichts Brauchbares studiert! Wie soll ich diese Aufgabe erfüllen?

Königin. Das ist mir egal! Wir sehen uns morgen wieder. Erfüllst du deine Aufgabe nicht, sollst du gehängt werden. Oder wir hacken deinen Kopf ab und verwenden ihn als Cricket-Ball.

Erzähler. Nach einer schlaflosen Nacht voller Sorgen wurde der Philosoph erneut zur Königin berufen.

Königin. Nun! Was hast du mir zu berichten?

Philosoph. Ich habe es tatsächlich geschafft, herauszufinden, wie man den Stein verwenden könnte, um unbegrenzt Gold herzustellen. Aber nur ich kann dieses Verfahren durchführen, Eure Hoheit.

Königin. Nun gut, dann sei es so!

Erzähler. Und so vergingen die Jahre, in denen sich der Philosoph in Sicherheit wähnte und die Angst vor Cricket-Schlägern langsam verlor. Die Königin suchte nun selbst nach dem Stein, aber solange sie ihn nicht fand, hatte der Philosoph nichts zu befürchten. Doch eines Tages passierte das Unfassbare: Die Königin hatte den Stein gefunden! Und lies prompt den Philosophen zu sich rufen.

Königin. Philosoph, sieh! Ich habe den Stein der Weisen gefunden, hier! Nun erfülle du deinen Teil der Abmachung! *[übergibt den Stein]*

Philosoph. Danke. Ihr hattet von mir verlangt, Euch den Stein der Weisen zu beschaffen oder herauszufinden, wie man mit ihm unbegrenzt Gold herstellen kann. Hier habt Ihr den Stein der Weisen. *[übergibt den Stein zurück]*

Edward Yang erzählt in seinem Blog eine leichte Variante dieses Märchens. Es wurde von Philip Wadler in seinem „CbV is dual to CbN“-Aufsatz popularisiert und geht vielleicht auf Peter Selinger zurück.

Wenn sich das Märchen „continuation-y“ anfühlt, hat man schon eine gewisse Vorahnung, wohin die Reise geht.

Nichtkonstruktive Beweise

Eine Zahl heißt genau dann **rational**, wenn sie sich als Bruch zweier ganzer Zahlen schreiben lässt.

- $\frac{21}{13}$ und 37 sind rational.
- $\sqrt{2}$ und π sind irrational.

Nichtkonstruktive Beweise

Eine Zahl heißt genau dann **rational**, wenn sie sich als Bruch zweier ganzer Zahlen schreiben lässt.

- $\frac{21}{13}$ und 37 sind rational.
- $\sqrt{2}$ und π sind irrational.

Satz. Es gibt **irrationale** Zahlen x und y sodass x^y rational ist.

Nichtkonstruktive Beweise

Eine Zahl heißt genau dann **rational**, wenn sie sich als Bruch zweier ganzer Zahlen schreiben lässt.

- $\frac{21}{13}$ und 37 sind rational.
- $\sqrt{2}$ und π sind irrational.

Satz. Es gibt **irrationale** Zahlen x und y sodass x^y rational ist.

Beweis. Entweder ist $\sqrt{2}^{\sqrt{2}}$ rational oder nicht.

1. Im ersten Fall sind wir fertig.
2. Im zweiten Fall können wir $x := \sqrt{2}^{\sqrt{2}}$ und $y := \sqrt{2}$ nehmen. Dann ist $x^y = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2$ rational.

Der Beweis ist schön und kurz. Allerdings kennen wir nach dem Beweis immer noch nicht ein konkretes Beispiel für ein Paar irrationaler Zahlen, die potenziert eine rationale Zahl ergeben! Der Beweis war *nichtkonstruktiv*.

Wenn wir aus einem Beweis explizite Zeugen (witnesses) extrahieren möchten, muss der Beweis konstruktiv sein, zum Beispiel wie folgender:

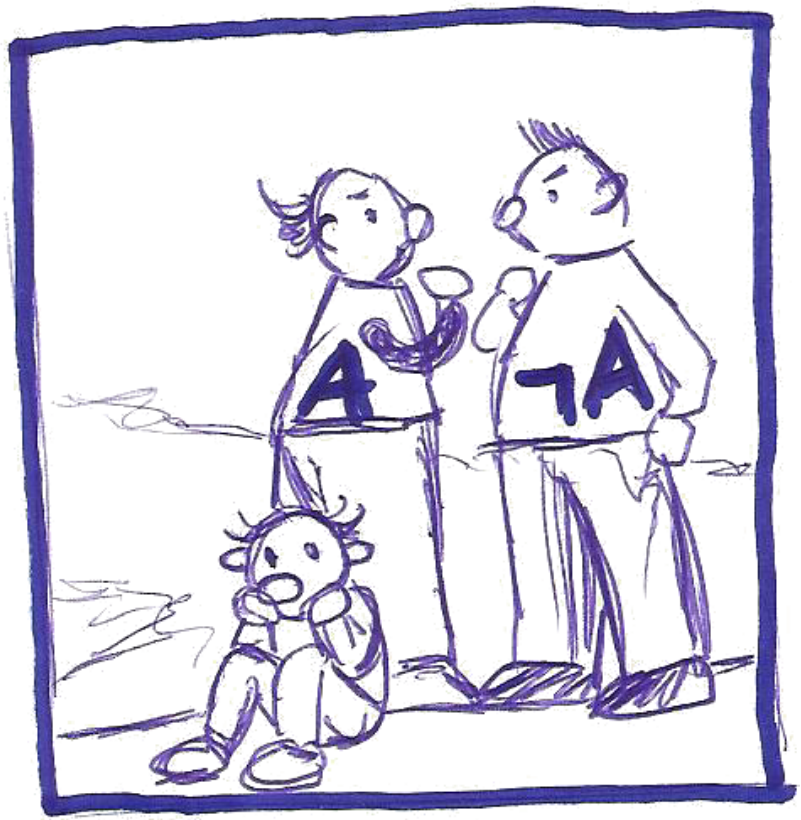
> *Setze* $x := \sqrt{2}$ *und* $y := \log_{\sqrt{2}} 3$. *Dann ist* $x^y = 3$ *rational. Der Beweis, dass y irrational ist, führt sich sogar leichter als der übliche Beweis, dass* $\sqrt{2}$ *irrational ist.*

Es stellt sich heraus, dass von all den Axiomen klassischer Logik genau eines für Nichtkonstruktivität verantwortlich ist: das auf der nächsten Folie beschriebene Axiom vom ausgeschlossenen Dritten. In dem Beweis auf der vorherigen Folie ging es gleich in der ersten Zeile ein.

Das Axiom vom ausgeschlossenen Dritten

„Für jede Aussage A dürfen wir $A \vee \neg A$ voraussetzen.“

Klassische Logik =
konstruktive Logik + das Axiom vom ausgeschlossenen Dritten.



Das Axiom vom ausgeschlossenen Dritten

„Für jede Aussage A dürfen wir $A \vee \neg A$ voraussetzen.“

Klassische Logik =
konstruktive Logik + das Axiom vom ausgeschlossenen Dritten.

Klassische Interpretation

$\bot$	Widerspruch.
$A \wedge B$	A und B sind wahr.
$A \vee B$	A ist wahr oder B ist wahr.
$A \Rightarrow B$	Sollte A wahr sein, so auch B.
$\forall x{:}X.\, A(x)$	Für alle $x : X$ gilt $A(x)$.
$\exists x{:}X.\, A(x)$	Es gibt ein $x : X$ mit $A(x)$.

Klassische vs. konstruktive Logik

Symbol	klassisch	konstruktiv
$\bot$	Widerspruch.	Widerspruch.
$A \wedge B$	A und B sind wahr.	Wir haben Beleg für A und für B.
$A \vee B$	A ist wahr oder B ist wahr.	Wir haben Beleg für A oder für B.
$A \Rightarrow B$	Aus A folgt B.	Wir können Beleg für A in Beleg für B transformieren.
$\forall x{:}X.\, A(x)$	Für alle $x : X$ gilt $A(x)$.	Zu jedem $x : X$ können wir Beleg für $A(x)$ konstruieren.
$\exists x{:}X.\, A(x)$	Es gibt ein $x : X$ mit $A(x)$.	Wir haben ein $x : X$ zusammen mit Beleg für $A(x)$.

Klassische vs. konstruktive Logik

Symbol	klassisch	konstruktiv
$\bot$	Widerspruch.	Widerspruch.
$A \wedge B$	A und B sind wahr.	Wir haben Beleg für A und für B.
$A \vee B$	A ist wahr oder B ist wahr.	Wir haben Beleg für A oder für B.
$A \Rightarrow B$	Aus A folgt B.	Wir können Beleg für A in Beleg für B transformieren.
$\forall x{:}X.\, A(x)$	Für alle $x : X$ gilt $A(x)$.	Zu jedem $x : X$ können wir Beleg für $A(x)$ konstruieren.
$\exists x{:}X.\, A(x)$	Es gibt ein $x : X$ mit $A(x)$.	Wir haben ein $x : X$ zusammen mit Beleg für $A(x)$.
$\neg A$	A ist falsch.	Es gibt keinen Beleg für A.
$\neg\neg A$	A ist nicht nicht wahr.	Es gibt keinen Beleg für $\neg A$.

In der konstruktiven Mathematik verwendet man dieselben logischen Symbole wie in der klassischen Mathematik, versieht sie aber mit einer leicht anderen Bedeutung. Die klassische Interpretation des Axioms vom ausgeschlossenen Dritten ist einfach die, dass jede Aussage wahr oder falsch ist. Da es hier nur um mathematische (also in einem starken Sinn objektive) Aussagen geht, ist das eine banale Trivialität: Entweder gibt es unendlich viele Primzahlzwillinge oder eben nicht. [Es gibt auch die Philosophie, das Axiom vom ausgeschlossenen Dritten auch auf Meta-Ebene abzulehnen. Darum geht es hier aber nicht.]

Die konstruktive Interpretation dagegen lautet: Für jede Aussage A haben wir entweder Beleg für A oder für $\neg A$. Das ist eine absurde Behauptung. (Man erinnere sich an Gödels Unvollständigkeitssatz: Es gibt Aussagen – *Gödelsätze* – die die Eigenschaft haben, dass weder sie noch ihre Negation beweisbar sind.)

Mit „wir“ in der Tabelle ist nicht wortwörtlich irgendeine Gruppe von Leuten gemeint; es sollte in einem generischen mathematischen Sinn gelesen werden. Stichwörter sind die *Brouwer–Heyting–Kolmogorov-Interpretation* und *Realisierbarkeitstheorie* (siehe zum Beispiel Notizen von Andrej Bauer).

Die Notation „$\neg A$“ für die Negation ist *syntaktischer Zucker* für „$A \Rightarrow \bot$“.

Aus einer Aussage A folgt konstruktiv wie klassisch ihre doppelte Negation $\neg\neg A$, aber konstruktiv gilt im Allgemeinen nicht die Umkehrung.

Vor ein paar Jahren tauchte ein Video von Kate Moss auf, das sie beim Konsumieren von Drogen zeigte. Aus dem Video war klar, dass es sich dabei entweder um Drogen von einem gewissen Typ *A* oder von einem gewissen Typ *B* handelte; aber es gab keinen direkten Beleg für einen der beiden Typen. Kate Moss wurde nicht verfolgt; in diesem Sinn verwendete das Justizsystem also konstruktive Logik. Siehe ein Blog-Post von Dan Piponi über das Thema.

Konstruktive Logik kann feinere Unterschiede abbilden als klassische Logik. Wenn wir zum Beispiel wissen, dass sich unser Haustürschlüssel irgendwo in der Wohnung befinden muss (da wir ihn letzte Nacht verwendet haben, um die Tür aufzusperren), wir ihn momentan aber nicht finden, so können wir konstruktiv nicht die Aussage vertreten, dass es eine Stelle gäbe, an der der Schlüssel liege. Denn dazu müssten wir in der Lage sein, einen expliziten Zeugen dieser Existenzbehauptung (also den Aufenthaltsort des Schlüssels) anzugeben. Wir können konstruktiv nur die durch doppelte Negation abgeschwächte Aussage vertreten.

(Die Beispiele hinken etwas, da sie sich auf den Kenntnisstand von gewissen Personen beziehen.)

Nebenbei bemerkt: Konstruktive MathematikerInnen behaupten *nicht*, dass das Axiom vom ausgeschlossenen Dritten falsch sei (d. h. dass seine Negation gelten würde). Sie setzen es nur nicht in seiner vollen Allgemeinheit voraus.

Tatsächlich lassen sich manche Instanzen des Axioms auch konstruktiv zeigen: Zum Beispiel folgt mit Induktion, dass jede natürliche Zahl gleich Null oder ungleich Null ist. (Die analoge Behauptung für reelle Zahlen lässt sich konstruktiv nicht zeigen. Das hat eine Parallele in der Programmierung: Bekanntlich ist es unschicklich, Fließkommazahlen auf Gleichheit zu testen, während das bei Ganzzahlen kein Problem ist.)

An dieser Stelle sollte auch das mancherorts immer noch geisternde Gerücht zerstreut werden, dass in konstruktiver Mathematik das Wort *Widerspruch* grundsätzlich verboten wäre. Das stimmt so nicht; wenn man die Situation genauer verstehen möchte, muss man den Unterschied zwischen *Beweisen von negierten Aussagem* und echten *Widerspruchsbeweisen* verstehen. Das erklärt Andrej Bauer auf seinem Blog ganz wunderbar.

Zwei letzte Bemerkungen. PhilosophInnen untersuchen nicht nur was *wahr* ist, sondern auch was stimmen *sollte*, was *möglich* ist, was *notwendig* ist, was jemand *weiß* oder was jemand *glaubt*. Das formalisiert man mit *modalen Operatoren*.

Als MathematikerIn kann man daher manchmal neidisch werden. In konstruktiver Mathematik aber gibt es durchaus eine Vielzahl von modalen Operatoren! Die Doppelnegation ist das wichtigste Beispiel. (In klassischer Logik ist die Doppelnegation auch ein modaler Operator, dort aber ein sehr uninteressanter.)

Es gibt mehrere gute und von philosophischen Vorlieben unabhängige Gründe, konstruktive Mathematik zu untersuchen. Einer steht auf der übernächsten Folie: die Curry–Howard-Korrespondenz funktioniert nur mit konstruktiven Beweisen. Ein anderer ist, dass nur konstruktive Logik allen *Topoi* – mathematischen Alternativuniversen – gemein ist.

Wer sich von einer mathematischen Sicht näher für das Thema interessiert, kann in ein Skript zu einem Pizzaseminar schauen.

Die Doppelnegationsübersetzung

$$
\begin{aligned}
(x = y)^{\square} &:\equiv \neg\neg(x = y) \\
(A \wedge B)^{\square} &:\equiv \neg\neg(A^{\square} \wedge B^{\square}) \\
(A \vee B)^{\square} &:\equiv \neg\neg(A^{\square} \vee B^{\square}) \\
(A \Rightarrow B)^{\square} &:\equiv \neg\neg(A^{\square} \Rightarrow B^{\square}) \\
(\forall x{:}X.\, A(x))^{\square} &:\equiv \neg\neg(\forall x{:}X.\, A^{\square}(x)) \\
(\exists x{:}X.\, A(x))^{\square} &:\equiv \neg\neg(\exists x{:}X.\, A^{\square}(x))
\end{aligned}
$$

Beispiel: Die Doppelnegationsübersetzung von

Es gibt eine Stelle, an der der Schlüssel liegt.

ist

*Es gibt **nicht nicht** eine Stelle, an der der Schlüssel liegt.*

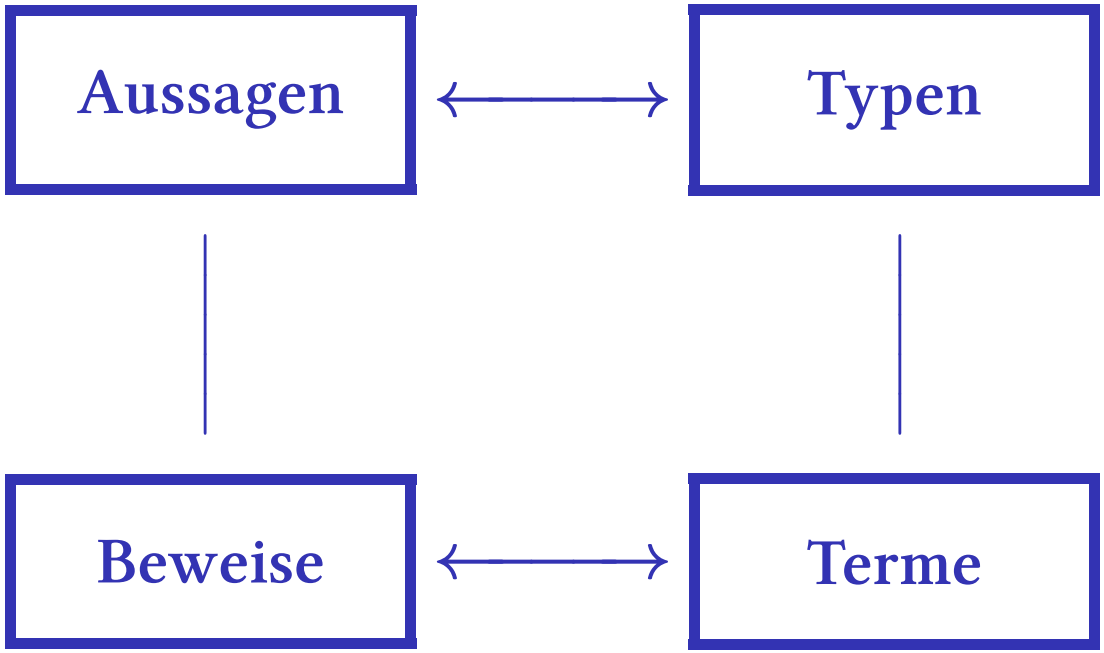
Satz. Es gilt A genau dann klassisch, wenn $A^{\square}$ konstruktiv gilt.

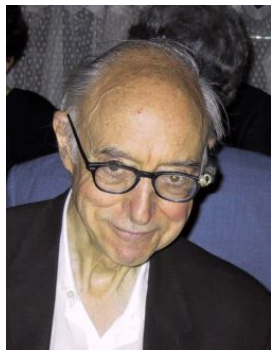
Die grauen $\neg\neg$'s können weggelassen werden. Die blauen $\neg\neg$'s dagegen sind entscheidend; man könnte sagen, dass der einzige Unterschied zwischen klassischer und konstruktiver Logik in der Bedeutung der Disjunktion und des Existenzquantors liegen.

Siehe ein Skript zu einem Pizzaseminar für Details zur Doppelnegationsübersetzung.

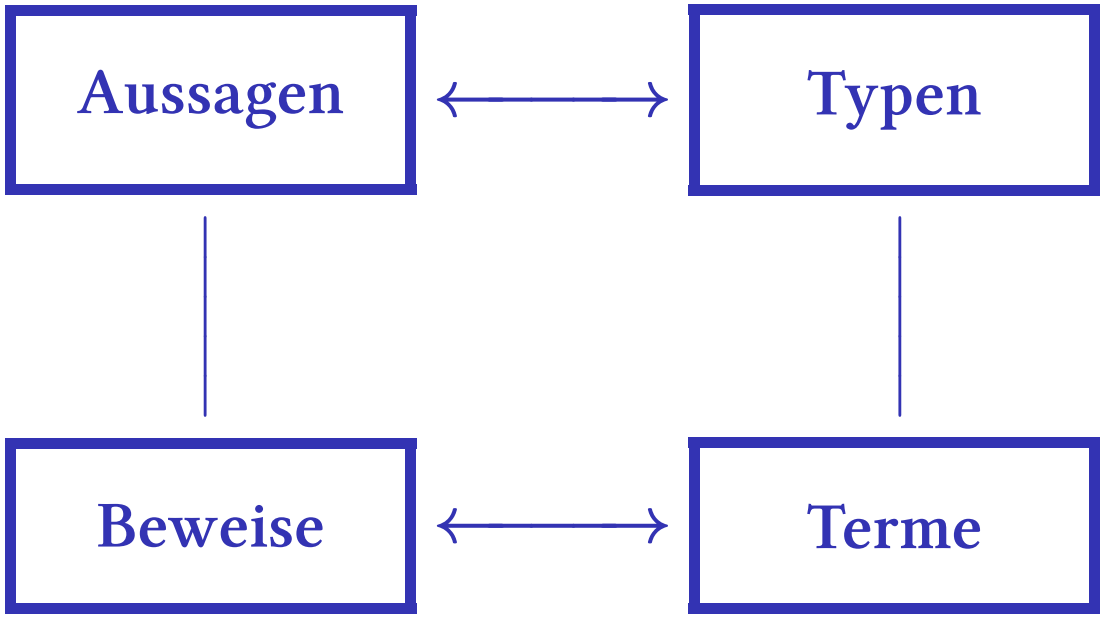
Die grundlegende Beweisidee werden wir im Laufe der Folien noch verstehen.

Die Curry–Howard-Korrespondenz





Die Curry–Howard-Korrespondenz



Logik	Programmierung
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`
$\neg A$	??

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`
$\neg A$, d. h. $A \Rightarrow \bot$	??

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`
$\neg A$, d. h. $A \Rightarrow \bot$	`A → r`

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`
$\neg A$, d. h. $A \Rightarrow \bot$	`A → r`
$\neg\neg A$, d. h. $(A \Rightarrow \bot) \Rightarrow \bot$	??

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$A \Rightarrow A$	`A → A`
$(A \wedge B) \Rightarrow A$	`(A,B) → A`
$A \Rightarrow (A \vee B)$	`A → Either A B`
$A \Rightarrow B$	`A → B`
Es gibt eine natürliche Zahl.	`Nat`
$\neg A$, d. h. $A \Rightarrow \bot$	`A → r`
$\neg\neg A$, d. h. $(A \Rightarrow \bot) \Rightarrow \bot$	`(A → r) → r`

Aus jedem konstruktiven Beweis kann man ein Programm extrahieren. Jedes Programm beweist eine Behauptung.

Die Curry–Howard-Korrespondenz schlägt eine erstaunliche und tief reichende Brücke zwischen konstruktiver Mathematik und Programmierung. Zu jeder logischen Aussage gehört ein Typ, anschaulich vorgestellt als der Typ ihrer Belege. Zu jedem konstruktiven Beweis der Aussage gehört ein Wert des entsprechenden Typs.

Zu einer falschen Aussage gehört dann einfach der leere Typ, der keinerlei Werte hat. Etwa ist die Aussage „aus A folgt B“ im Allgemeinen falsch, und passend dazu ist der Typ `A` $\rightarrow$ `B` im Allgemeinen unbewohnt (wenn man von `unsafeCoerce` und Nichtterminierung absieht).

Das andere Extrem ist die Aussage „Es gibt eine natürliche Zahl.“, für die jede natürliche Zahl jeweils ein Beleg ist. Der Typ der Belege dieser Aussage ist also der Typ der natürlichen Zahlen.

Die Curry–Howard-Korrespondenz ist fundamental auf konstruktive Beweise beschränkt.

Für eine formale Behandlung muss man sowohl die linke Seite („Logik“) als auch die rechte („Programmierung“) konkretisieren. Eine Möglichkeit ist *propositionale intuitionistische Logik* als linke und das *einfach-getypte Lambda-Kalkül* als rechte Seite.

Unter der Curry–Howard-Korrespondenz gehört zur Negation $\neg A$ der Typ `A → r`. Das stellt man sich vor: Wenn man konstruktiv $\neg A$ behauptet, so behauptet man, aus einem hypothetischen Beleg für A Beleg einer beliebigen anderen Aussage machen zu können. Schließlich glaubt man ja, dass es gar keinen Beleg für A gibt, sodass man also überzeugt ist, nie in die Verlegenheit zu geraten, diese Transformation durchführen zu müssen.

Mit der Curry–Howard-Korrespondenz ist auch klarer, was der semantische Gehalt einer doppelt negierten Aussage ist. Unter der Korrespondenz gehört zur Aussage $\neg\neg A$ der Typ `(A → r) → r`, also einfach `Cont r A`. Ein Wert dieses Typs kann zum Beispiel von der Form `return x` sein, wobei `x` ein Wert vom Typ `A` ist. Ein Wert dieses Typs kann aber auch eine andere Form haben – er kann mit der gegebenen Continuation spielen, zum Beispiel sie mehrmals ausführen.

Etwas unpräzise ausgedrückt kann man sich merken: Eine doppelt negierte Aussage $\neg\neg A$ konstruktiv zu belegen bedeutet, die Aussage A *innerhalb der Continuation-Monade*, also etwa unter Verwendung von Zeitsprüngen in die Vergangenheit, zu belegen.

$\neg\neg$-Übersetzung $\widehat{=}$ CPS-Transformation

Logik	Programmierung
Aussage A	Typ `A` der Belege für A
Beweis von A	Programm vom Typ `A`
$\neg\neg A$, d. h. $(A \Rightarrow \bot) \Rightarrow \bot$	`(A → r) → r`, d. h. `Cont r A`
Doppelnegationsübersetzung	CPS-Transformation

```
type Cont r a = ((a → r) → r)

-- Axiom vom ausgeschlossenen Dritten ohne Übersetzung:
-- lem :: Either a (a → r)

lem :: Cont r (Either a (a → Cont r b))
lem = \k → k $ Right $ \x → (\k' → k (Left x))
```

Mit den Übersetzungsregeln der Curry–Howard-Korrespondenz entspricht zur Aussage $A \vee \neg A$ der Typ `Either A (A → r)`. Dieser ist im Allgemeinen nicht bewohnt, passend dazu, dass sich konstruktiv das Axiom vom ausgeschlossenen Dritten nicht zeigen lässt.

Die Doppelnegationsübersetzung des Axioms vom ausgeschlossenen Dritten lässt sich aber sehr wohl konstruktiv zeigen. Das zeigt der auf der vorherigen Folie angegebene Term `lem`.

Es gibt übrigens mehrere Möglichkeiten, aus einem klassischen Beweis einer Aussage einen konstruktiven Beweis der übersetzten Aussage zu machen. Diese Möglichkeiten entsprechen den unterschiedlichen Varianten der CPS-Transformation (call by name, call by value, …).

Wie funktioniert das Programm `lem`? Zunächst sichern wir uns die aktuelle Continuation `k`. Dann behaupten wir sofort: „Die Aussage ist falsch!“ Beziehungsweise: „Es gibt keine Werte vom Typ `a`!“ Wir geben also `Right f` zurück, mit `f` vom Typ `a → Cont r b`. Diese Funktion `f` ist der Zeuge für unsere Behauptung.

Nun kann es zufälligerweise sein, dass unsere Behauptung richtig ist. Dann wird `f` nie aufgerufen. Es kann aber auch sein, dass der Aufrufer einen Wert `x` vom Typ `a` auftreibt und damit unseren Bluff herausfordert. Natürlich ist es uns nicht möglich, mit einem Wert vom Typ `b` aufzuwarten (über diesen wissen wir ja nichts weiter). Wir befreien uns aus unserer Not, indem wir die momentan laufende Berechnung abbrechen und die anfangs gesicherte Continuation nehmen. Dieser übergeben wir gerade den uns präsentierten Wert `x`.

Das ist auch genau die Strategie, die der Philosoph aus dem Märchen vom Anfang verfolgen sollte. Bei wörtlicher Auslegung der Geschichte nimmt er ja die Königin etwas auf den Arm, was sie vermutlich nicht gutheißen wird. Er sollte stattdessen, sobald er von der Königin den Stein der Weisen ausgehändigt bekommen hat, zurück in die Vergangenheit springen, zu jenem verhängnisvollen Tag, an dem die Königin den Philosophen das erste Mal zu sich rief.

Minima unendlicher Listen

Satz. In jeder unendlichen Liste `xs` natürlicher Zahlen gibt es ein kleinstes Element.

Aber nicht konstruktiv! Es gibt kein Programm vom Typ:

```
minimum :: [Nat] → Ix
-- Ix: Typ der Indizes (also Nat)
```

Minima unendlicher Listen

Satz. In jeder unendlichen Liste `xs` natürlicher Zahlen gibt es ein kleinstes Element.

Aber nicht konstruktiv! Es gibt kein Programm vom Typ:

```
minimum :: [Nat] → Ix
-- Ix: Typ der Indizes (also Nat)
```

Beweis. Durch noethersche Induktion über die Größe eines gegebenen Elements `xs!!i`. Entweder gibt es einen Index `j` mit `xs!!j < xs!!i` oder nicht.

1. Im ersten Fall gibt es ein Minimum nach Ind'voraussetzung.
2. Im zweiten Fall ist `xs!!i` minimal.

Minima unendlicher Listen

```
type Cont r a = ((a → r) → r)

lem :: Cont r (Either a (a → Cont r b))
lem = \k → k $ Right $ \x → (\k' → k (Left x))

minimum :: [Nat] → Cont r (Ix, Ix → Cont r ())
minimum xs = go 0 where
    go i = do
        oracle ← lem
        case oracle of
            Left  j → go j
            Right f → return (i, \j →
                if xs!!j ≥ xs!!i then return () else f j)
```

Minima unendlicher Listen

```
type Cont r a = ((a → r) → r)

lem :: Cont r (Either a (a → Cont r b))
lem = \k → k $ Right $ \x → (\k' → k (Left x))

minimum :: [Nat] → Cont r (Ix, Ix → Cont r ())
minimum xs = go 0 where
   go i = do
      oracle ← lem
      case oracle of
         Left  j → go j
         Right f → return (i, \j →
            if xs!!j ≥ xs!!i then return () else f j)

example = do
   (i, g) ← minimum [...]
   g 5 ≫ g 7 ≫ g 3
   ...
```

Die angegebene Funktion `minimum` berechnet (innerhalb der Continuation-Monade) das Minimum einer unendlichen Liste von natürlichen Zahlen. Ihr Rückgabewert ist ein Tupel `(i, g)` bestehend aus dem Index `i` des minimalen Elements und einem Zeugen `g` seiner Minimalität: Gegeben ein weiterer Index `j`, so soll `g j` einen Beweis der Behauptung `xs!!j >= xs!!i` zurückgeben.

Um das treu umsetzen zu können, bräuchten wir abhängige Typen. Da es die in Haskell nicht gibt, muss der Unit-Typ `()` zusammen mit einem sozialen Versprechen als Ersatz herhalten.

Wie geht die Funktion `minimum` in der Praxis vor? Um eigene Fehler später korrigieren zu können, sichert sie zunächst die aktuelle Continuation. Anschließend proklamiert sie das erste Listenelement (Index `0`) als Minimum. Das kann tatsächlich korrekt sein. Falls allerdings irgendwann `g` mit einem Index `j`, für den `xs!!j < xs!!i` ist, aufgerufen wird, so nimmt sie die eingangs gesicherte Continuation und sorgt so dafür, dass es den Anschein hat, als hätte sie die gesamte Zeit über behauptet, das Element `xs!!j` sei das Minimum. Auf dieselbe Art und Weise geht es weiter.

Von innerhalb der Continuation-Monade aus kann dieses Schummeln nicht beobachtet werden.

Man muss allerdings beim Verlassen der Monade Sorgfalt walten lassen: Je nach konkretem Einsatzszenario kann es schlimm sein oder auch nicht, dass die Funktion nicht das tatsächlich minimale Element bestimmt, sondern nur das kleinste unter allen vom Rest des Programms inspizierten Elementen.

Spannend ist noch, dass die Funktion `minimum` *nicht* das Ergebnis einer kreativen Überlegung ist. Stattdessen ergibt sich die Funktion rein mechanisch, indem man zunächst den nur in klassischer Logik gültigen Beweis des besprochenen Satzes mit der Doppelnegationsübersetzung in einen konstruktiven Beweis überführt und anschließend diesen mit der Curry–Howard-Korrespondenz zu einem Programm transformiert.

Die Struktur des klassischen Beweises kann man in dem Programmcode deutlich erkennen. Der Aufruf von `lem` steht sowohl im Code als auch im Beweis ganz am Anfang. Der `Left`-Fall im Code entspricht Fall 1 im Beweis. Der Beweis verwendete in diesem Fall die Induktionsvoraussetzung; das äußert sich im Code durch den rekursiven Aufruf `go j`.

Der `Right`-Fall im Code entspricht Fall 2 im Beweis. Dieser war auf der Folie verkürzt wiedergegeben; ausführlich lautet er so (ja, das ist etwas umständlich formuliert): „Um zu beweisen, dass `xs!!i` minimal ist, sei ein beliebiger Index `j` gegeben. Wir müssen zeigen, dass `xs!!j` $\geq$ `xs!!i`. Falls dem so sein sollte (diese Fallunterscheidung können wir konstruktiv treffen), sind wir fertig. Andernfalls gilt also `xs!!j < xs!!i`. Das ist ein Widerspruch zur Fallvoraussetzung. Nach dem Explosionsprinzip folgt aus diesem Widerspruch unsere Behauptung.“ Die Ähnlichkeit zum Code ist offenkundig; der Anwendung des Explosionsprinzips nach dem Herleiten des Widerspruchs entspricht dem Aufruf `f j`.

Hier folgt noch ein praktisches Beispiel aus algorithmischer Zahlentheorie für die Nützlichkeit dieses „algorithmischen Axioms vom ausgeschlossenen Dritten“. Es soll demonstrieren, dass das schummelnde Verhalten in manchen Fällen völlig ausreicht.

Sei x eine algebraische Zahl, also eine komplexe Zahl, die Nullstelle eines normierten Polynoms $f(X) \in \mathbb{Q}[X]$ ist. Dann gibt es die von x erzeugte Körpererweiterung $\mathbb{Q}(x)$ von $\mathbb{Q}$; dieser Körper enthält die rationalen Zahlen, die Zahl x und alle Zahlen, die induktiv aus diesen durch Addition, Subtraktion, Multiplikation und Division gewonnen werden können.

Wir möchten einen Algorithmus dafür angeben, ein von Null verschiedenes Element in $\mathbb{Q}(x)$ zu invertieren und das Inverse als polynomiellen Ausdruck in x anzugeben. Theoretisch funktioniert das wie folgt:

Zunächst zerlegen wir $f(X)$ in seine irreduzible Faktoren. Einer der Faktoren, sagen wir $g(X)$, hat immer noch x als Nullstelle. Dieser Faktor heißt auch *Minimalpolynom* von x, und die allgemeine Theorie besagt, dass $\mathbb{Q}(x)$ isomorph zum Ring $\mathbb{Q}[X]/(g(X))$ ist. Das heißt: In $\mathbb{Q}(x)$ zu arbeiten ist gleichbedeutend damit, im Polynomring $\mathbb{Q}[X]$ modulo $g(X)$ zu arbeiten. Von diesem weiß man, dass der erweiterte euklidische Algorithmus ein effizientes Verfahren zur Bestimmung von modularen Inversen ist.

Allerdings hat dieses Verfahren ein gravierendes Problem: Die Zerlegung von $f(X)$ in seine irreduziblen Faktoren ist sehr aufwendig. Kann man die Zerlegung irgendwie umgehen?

Ja! Sei $[h(X)] \in \mathbb{Q}[X]/(f(X))$. Wir möchten das Inverse von $h(x)$ in $\mathbb{Q}(x)$ bestimmen. Da $f(X)$ nicht notwendigerweise irreduzibel ist, ist der Ring $\mathbb{Q}[X]/(f(X))$ nicht notwendigerweise ein Körper (jeder Faktor von $f(X)$ ist in ihm ein Nullteiler). Wir können aber trotzdem den erweiterten euklidischen Algorithmus verwenden, um den (monischen) größten gemeinsamen Teiler $d(X)$ von $f(X)$ und $h(X)$ und eine *Bézoutdarstellung*

$$d(X) = a(X)f(X) + b(X)h(X).$$

zu berechnen. Dann können drei Fälle eintreten:

1. $d(X) = 1$. Die Gleichung zeigt dann, dass $b(X)$ modulo $f(X)$ zu $h(X)$ invers ist. Insbesondere ist also $b(x)$ in $\mathbb{Q}(x)$ invers zu $h(x)$.

2. $d(X) = f(X)$. Dann zeigt die Gleichung, dass $h(X)$ ein Vielfaches von $f(X)$ ist und somit $h(x)$ Null ist. In diesem Fall müssen (und können) wir das Inverse nicht berechnen.

3. Ansonsten ist $d(X)$ ein nichttrivialer Faktor von $f(X)$. Mindestens eines der beiden Polynome $d(X)$ und $f(X)/d(X)$ hat immer noch x als Nullstelle; wir nennen dieses $\widetilde{f}(X)$ und starten die Berechnung mit $\widetilde{f}(X)$ statt $f(X)$ neu.

Auf diese Art und Weise können wir in $\mathbb{Q}[X]/(g(X))$ arbeiten ohne explizit $g(X)$ bestimmen zu müssen. Die Continuation-Monade zu verlassen ist kein Problem, da Inverse modulo $f(X)$, modulo $\widetilde{f}(X)$ oder gar modulo $\widetilde{\widetilde{f}}(X)$ auch Inverse modulo des eigentlich relevanten $g(X)$ sind.

Fazit

- Konstruktive Beweise haben algorithmischen Inhalt. Dieser lässt sich mit der Curry–Howard-Korrespondenz maschinell extrahieren.
- Die Doppelnegationsübersetzung macht aus jedem klassischen Beweis einen konstruktiven. Auf diese Weise steckt auch in klassischen Beweisen noch algorithmischer Inhalt. Dieser spielt sich in der Continuation-Monade ab.
- Manchmal lösen die so erhaltenen Algorithmen konkrete Probleme.

Dieser Foliensatz stellt nur eine Popularisierung der Ideen anderer dar. Die Originalliteratur ist die Doktorarbeit von Chetan Murthy: *Extracting Constructive Content from Classical Proofs* (1990).

Andere Popularisierungen findet man in Artikeln von

- Thierry Coquand,
- Oleg Kiselyov,
- Luke Palmer,
- Dan Piponi (sigfpe),
- Philip Wadler und
- Edward Z. Yang.

Empfehlenswert ist in diesem Zusammenhang auch ein Buchkapitel von Andrej Bauer.

Gründe einen funktionalen Stammtisch!

In Augsburg haben wir ein monatliches Treffen mit Vorträgen und gemeinsamen Restaurantbesuch für alle, die an funktionalen Sprachen interessiert sind, ins Leben gerufen. Überleg doch, so etwas auch in deiner Stadt zu gründen!

Gründe einen Matheschülerzirkel!

Organisiere ein Angebot, bei dem an Mathematik interessierte Schülerinnen und Schüler aus deiner Stadt regelmäßig an die Uni kommen können, um in kleinen Gruppen spannende Mathematik abseits des Schulunterrichts zu sehen. Mach das ganze per Post für weiter entfernt wohnende Kinder und Jugendliche. Und veranstalte immer in den Sommerferien ein einwöchiges Mathecamp.

In Augsburg machen wir das seit etwa drei Jahren mit durchschnittlich etwa 200 Teilnehmenden. Vielleicht würde auch in deiner Stadt ein solches Angebot gut aufgenommen werden! Wir können dich mit umfangreichen Materialien und Erfahrungsberichten versorgen.