Informatik und Logik:

Syntax und Semantik formaler Sprachen
Beweise und Programme

Gerhard Jäger

Theoretische Informatik und Logik

Herbstsemester 2009

Logik

Grundsätzliches

- Penguin Encyclopedia: "The formal systematic study of the principles of valid inference and correct reasoning."
- Wikipedia: "The unifying themes in mathematical logic include the study of the expressive power of formal systems and the deductive power of formal proof systems."

Einteilung

Logik

mathematische Logik | philosophische Logik

Logik

Kurzer Blick auf die Vorgeschichte

- Aristoteles (384–322): Entwicklung einer Bedeutungstheorie, Deduktion und Induktion, Theorie der Argumentation, Syllogismen
- Wilhelm von Ockham (1285–1347): Wesentliche Beiträge zur mittelalterlichen Sprachlogik (Hauptwerk: summa logicae), Ockhams Rasiermesser
- Gottfried Wilhelm Leibniz (1646–1716): Gleichungslogik, Formulierung der Syllogistik in diesem Rahmen
- George Boole (1815–1864): Erster algebraischer Logikkalkül, Boolesche Algebren, Grundlagen der klassischen Aussagenlogik

Logik

- Gottlob Frege (1848–1925): Entwicklung einer formalen Sprache, formale Beweise, Grundlagen der klassischen Prädikatenlogik
- Bertrand Russell (1872–1970) und Alfred North Whitehead (1861–1947): Principia Mathematica (erschienen 1910), Grundlagen der Mathematik in einem vollständig formalen typentheoretischen Rahmen
- David Hilbert (1862–1943): Entwicklung der Beweistheorie als Reaktion auf die Grundlagenkrise der Mathematik, Formeln und Beweise als Bitstrings
- Alonzo Church (1903–1995): Entwicklung des Lambda-Kalküls als formalem Rahmen für Berechnungen, Entdeckung unentscheidbarer Probleme

Logik

- Kurt Gödel (1906–1978): Vollständigkeits- und Unentscheidbarkeitssätze, Beweisbarkeitslogik
- Gerhard Gentzen (1909–1945): Mitbegründer der modernen mathematischen Beweistheorie, Grundlagen für "Programs as Proofs"
- Alain Turing (1912–1954): Turing-Maschinen als universelle Berechnungsmodelle, Unentscheidbarkeit des Halteproblems

Hauptgebiete der mathematischen Logik

- Mengenlehre
- Modelltheorie
- Berechnungstheorie
- Beweistheorie
- Logik in der Informatik

Logik in der Informatik

Allgemeiner Slogan

Logik ist für die Informatik das, was die Analysis für die Physik ist.

Einige Anwendungsbereiche

- Klassische Aussagenlogik für den Design von Booleschen Schaltkreisen; Hardware-Spezifikationen
- Klassische Logik erster Stufe als (sehr) allgemeiner Rahmen zur formalen Spezifikation von System- und Programmeigenschaften
- Varianten davon als Spezifikationssprachen (z.B. die bekannte Spezifikationssprache **Z**)
- Logical Frameworks und Typensysteme im Zusammenhang mit der Entwicklung von Programmiersprachen

Logik in der Informatik

- Programmlogiken (Hoare Logic, Hennesey-Miler Logic) und dynamische Logiken zur Untersuchung der Korrektheit von Programmen
- Multimodale und epistemische Logiken zur formalen Repräsentation und Verarbeitung von Wissen in Mehragentensystemen
- Rekursionstheoretische Konzepte für die Entwicklung abstrakter Berechnungsmodelle (sequentiell und parallel); Komplexitätstheorie
- Automatisches Beweisen und Logik-Programmierung
- Proofs as Computations

Allgemeine Gesichtspunkte

Definition

- Eine formale Sprache ist eine Menge von Wörtern, die aus einem gegebenen Alphabet gebildet werden. Auf diesen Wörtern können Operationen definiert werden, die unter Umständen zu neuen Wörtern führen.
- Die Syntax einer formalen Sprache legt fest, auf welche Weise die Wörter gebildet werden und welche Operationen mit den Wörtern durchgeführt werden dürfen. Sie regelt die rein formalen Beziehungen zwischen Zeichen und Wörtern. Von der inhaltlichen Bedeutung der Zeichen und Wörter wird auf Ebene der Syntax abgesehen.
- Die Semantik einer formalen Sprache interpretiert die Zeichen und Wörter und gibt ihnen in diesem Sinn eine Bedeutung. Diese Bedeutung wird dabei durch Regeln explizit festgelegt.

Allgemeine Gesichtspunkte

Beispiele von formalen Sprachen

- Die Sprache **CPL** der klassischen Aussagenlogik (Classical Propositional Logic)
- Die Sprache **FOL** der klassischen Logik erster Stufe (Classical First Order Logic)
- Die Sprache der Peano-Arithmetik **PA** und die Sprache der Zermelo-Fraenkel-Mengenlehre **ZFC**
- **Die üblichen Programmiersprachen**
- Sprachen im Zusammenhang mit Lindenmayer-Systemen, der Chomsky-Hierarchy und formalen Grammatiken

Allgemeine Gesichtspunkte

Je nach Kontext und Anwendungsbereich betrachtet man verschiedene Arten von Semantiken

Beispiele von Semantiken

- Wahrheitsfunktionale Semantik: Die Bedeutung eines Satzes wird auf die Frage nach seiner Wahrheit zurückgeführt.
- Algebraic Semantics: Algebraische Spezifikation von Datentypen und Programmiersprachen; Beschreibung der Programmschritte durch algebraische Operationen auf den entsprechenden Bereichen.
- Denotational Semantics (Scott & Strachey): Einem Programm π wird eine Funktion $[\![\pi]\!]$ als Bedeutung zugeordnet, die Inputs in Outputs abbildet.

Allgemeine Gesichtspunkte

Beispiele von Semantiken (Fortsetzung)

- Operational Semantics (Plotkin & Scott): Eine mathematische Beschreibung des "Verhaltens" von Programmen und Systemen. Ein Programm π wird verstanden als eine Folge von einzelnen Berechnungsschritten.
- Axiomatic Semantics (Hoare): Die Bedeutung eines Befehls wird verstanden als der Effekt, den dieser Befehl auf Aussagen über den Programmzustand ausübt; $\{A\}\pi\{B\}$: Gilt A vor Ausführung von π, so gilt B nach Ausführung von π.

Vorbemerkungen

George Boole (1815–1864)

- Übersetzung von Logik (logischen Fragen) in die Sprache der Algebra
- Korrespondenz: logische Symbole $\approx$ algebraische Operationen
- Algebraisierung der klassischen Aussagenlogik und damit Einbindung der Logik in die Mathematik
- Beginn der modernen Logik
- Relativ einfache und durchsichtige Syntax und Semantik mit dennoch beachtlicher Ausdrucksstärke
- Grundlage vieler Logikkalküle und Systeme für Reasoning, Informationsverarbeitung und Wissensrepräsentation

Syntax von **CPL**

Ausgangsalphabet von **CPL**

- Atomare Propositionen: $U, V, W, U_0, V_0, W_0, U_1, V_1, W_1, \ldots$
- Logische Konnektive: $\neg, \vee, \wedge, \rightarrow$
- Als weitere Hilfszeichen Klammern und Komma

Atomare Propositionen sind Grundaussagen, die (im gegenwärtigen Kontext) nicht weiter untergliedert werden. Aus ihnen werden mit Hilfe der logischen Konnektive komplexe Aussagen aufgebaut.

Syntax von **CPL**

Definition (Formeln von **CPL**)

Die Formeln von **CPL** werden induktiv wie folgt definiert:

1. Jede atomare Proposition ist eine Formel von **CPL**.
2. Ist A eine Formel von **CPL**, so ist auch $\neg A$ eine Formel von **CPL**.
3. Sind A und B Formeln von **CPL**, so sind auch $(A \vee B)$, $(A \wedge B)$ sowie $(A \rightarrow B)$ Formeln von **CPL**.

Konvention

Als Mitteilungszeichen (Metavariablen) verwenden wir (u.U. mit Indizes):

A, B, C, D, E für Formeln von **CPL**.

Syntax von **CPL**

Beispiel

Stehen etwa die atomare Proposition U und V für die Aussagen "es regnet" und "die Strasse ist nass", so bedeuten:

$$(U \to V) \quad :: \quad \text{wenn es regnet, ist die Strasse nass}$$

$$(U \wedge V) \quad :: \quad \text{es regnet, und die Strasse ist nass}$$

$$(\neg V \to \neg U) \quad :: \quad \text{wenn die Strasse nicht nass ist, regnet es nicht}$$

Definition (Länge)

Unter der Länge $\ell(A)$ einer Formel A von **CPL** verstehen wir die Anzahl der Vorkommnisse der atomaren Propositionen und logischen Konnektive in A.

Semantik von **CPL**

Die übliche Semantik von **CPL** ist eine wahrheitsfunktionale Semantik (siehe oben). Letztlich sollen jeder Formel von **CPL** einer der beiden Wahrheitswerte **w** (wahr) oder **f** (falsch) zugeordnet werden. Da jedoch die Wahrheit der Formeln von **CPL** davon abhängt, wie wir die atomaren Propositionen interpretieren, müssen zuerst deren Wahrheitswerte festlegen.

Definition (Propositionenbelegung)

Eine *Propositionenbelegung* ist eine Abbildung $\mathbb{B}$, die jeder atomaren Proposition U einen Wahrheitswert $\mathbb{B}(U) \in \{\mathbf{w}, \mathbf{f}\}$ zuordnet.

Jede Propositionenbelegung $\mathbb{B}$ lässt sich dann sehr einfach zu einer Funktion $\widehat{\mathbb{B}}$ fortsetzen, die jede Formel A von **CPL** auf einen Wahrheitswert $\widehat{\mathbb{B}}(A) \in \{\mathbf{w}, \mathbf{f}\}$ abbildet.

Semantik von **CPL**

Definition (Wahrheitswert)

Gegeben sei eine Propositionenbelegung $\mathbb{B}$. Dann wird der Wahrheitswert $\widehat{\mathbb{B}}(A) \in \{\mathbf{w}, \mathbf{f}\}$ einer Formel A von **CPL** induktiv wie folgt definiert:

1. Für jede atomare Proposition U setzen wir $\widehat{\mathbb{B}}(U) := \mathbb{B}(U)$.
2. Ist A die Formel $\neg B$, dann sei

$$\widehat{\mathbb{B}}(A) := \begin{cases} \mathbf{f}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{w}, \\ \mathbf{w}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{f}. \end{cases}$$

Semantik von **CPL**

Definition (Wahrheitswert, Fortsetzung)

3. Ist A die Formel $(B \vee C)$, dann sei

$$\widehat{\mathbb{B}}(A) := \begin{cases} \mathbf{f}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{f} \text{ und } \widehat{\mathbb{B}}(C) = \mathbf{f}, \\ \mathbf{w}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{w} \text{ oder } \widehat{\mathbb{B}}(C) = \mathbf{w}. \end{cases}$$

4. Ist A die Formel $(B \wedge C)$, dann sei

$$\widehat{\mathbb{B}}(A) := \begin{cases} \mathbf{f}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{f} \text{ oder } \widehat{\mathbb{B}}(C) = \mathbf{f}, \\ \mathbf{w}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{w} \text{ und } \widehat{\mathbb{B}}(C) = \mathbf{w}. \end{cases}$$

Semantik von **CPL**

Definition (Wahrheitswert, Fortsetzung)

5. Ist A die Formel $(B \to C)$, dann sei

$$\widehat{\mathbb{B}}(A) := \begin{cases} \mathbf{f}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{w} \text{ und } \widehat{\mathbb{B}}(C) = \mathbf{f}, \\ \mathbf{w}, & \text{falls } \widehat{\mathbb{B}}(B) = \mathbf{f} \text{ oder } \widehat{\mathbb{B}}(C) = \mathbf{w}. \end{cases}$$

Definition (Gültigkeit, Erfüllbarkeit)

1. Eine Formel A von **CPL** heisst *gültig*, falls $\widehat{\mathbb{B}}(A) = \mathbf{w}$ für alle Propositionenbelegungen $\mathbb{B}$ gilt.
2. Eine Formel A von **CPL** heisst *erfüllbar*, falls es eine Propositionenbelegung $\mathbb{B}$ mit $\widehat{\mathbb{B}}(A) = \mathbf{w}$ gibt.

Semantik von **CPL**

Definition (Wahrheitstafeln)

A	$\neg A$
f	**w**
w	**f**

A	B	$A \vee B$	$A \wedge B$	$A \rightarrow B$
f	**f**	**f**	**f**	**w**
f	**w**	**w**	**f**	**w**
w	**f**	**w**	**f**	**f**
w	**w**	**w**	**w**	**w**

Semantik von **CPL**

Bemerkung 1

Gegeben sei eine Formel A von **CPL** sowie eine Propositionenbelegung $\mathbb{B}$.

1. Dann lässt sich $\widehat{\mathbb{B}}(A)$ mit einem Zeitaufwand berechnen, der polynomial in der Länge $\ell(A)$ von A ist.
2. Für die Berechnung von $\widehat{\mathbb{B}}(A)$ spielen nur die Werte der atomaren Propositionen, die in A auftreten, eine Rolle.

Beispiel

Gegeben seien die **CPL**-Formel $A := ((\neg U_2 \vee U_3) \wedge U_1)$ sowie die Propositionenbelegung $\mathbb{B}$, die U_1 auf **w** und alle anderen atomaren Propositionen auf **f** abbildet. Dann gilt:

$$\widehat{\mathbb{B}}(A) = \mathbf{w}.$$

Semantik von **CPL**

Bemerkung 2

Von eine Formel A von **CPL** lässt sich auf folgende Weise feststellen, ob sie erfüllbar ist:

1. Bestimme die (endlich vielen) atomaren Propositionen, die in A auftreten. Nehmen wir an, dass es sich dabei um die atomaren Propositionen $U_1, \ldots, U_n$ handelt.
2. Nun ordnen wir den $U_1, \ldots, U_n$ systematisch alle möglichen Kombinationen von Wahrheitswerten zu und bestimmen – mit polynomialem Aufwand – den jeweiligen Wahrheitswert von A und zwar so lange bis
   (a) A den Wert **w** erhält oder
   (b) alle möglichen Kombinationen ausprobiert worden sind.

   Im Fall (a) ist die Formel A erfüllbar, im Fall (b) ist sie nicht erfüllbar.

Semantik von **CPL**

Bemerkung 3

Ist A eine Formel von **CPL**, die unerfüllbar ist und in der n verschiedene atomare Propositionen auftreten, so müssen beim eben beschriebenen Verfahren 2^n Kombinationen von Wahrheitswertzuordnungen getestet werden. Dieses Verfahren ist also exponentiell in der Anzahl der vorkommenden atomaren Propositionen.

Es ist eines der bedeutendsten ungelösten Probleme der Informatik und Mathematik – das sogenannte P-NP-Problem –, ob die Erfüllbarkeit aussagenlogischer Formeln mit polynomialen Zeitaufwand getestet werden kann.

Dies liegt daran, dass das Erfüllbarkeitsproblem aussagenlogischer Formeln polynomial äquivalent zu einer ganzen Reihe wichtiger kombinatorische Probleme ist (z.B. Traveling Salesperson).

Vorbemerkung

Die Sprache WHILE ist ein sehr kleines Fragment einer "richtigen" Programmiersprache, das sich auf einen für Berechnungen zentralen Kern beschränkt. Diese Sprache ist aber dennoch aus mehreren Gründen von grossem Interesse:

- Einige zentrale Aspekte von Programmiersprachen lassen sich anhand von WHILE paradigmatisch diskutieren.
- WHILE is "computationally complete", d.h. alle partiell-rekursiven (partiellen berechenbaren) zahlentheoretischen Funktionen lassen sich durch WHILE-Programme berechnen.

Syntax von WHILE

Ausgangsalphabet von WHILE

- Variablen: X_0, X_1, X_2, ...
- Grundzeichen: 0, S, P, $\leftarrow$, $\neq$, ;, (,), while, do, od

Davon ausgehend werden Wertzuweisungen und Programme durch folgende Produktionsregeln definiert.

Syntax von WHILE

Definition (Wertzuweisungen, Programme)

$$
\begin{aligned}
\langle \text{Wertzuweisung} \rangle \quad ::= \quad & \langle \text{Variable} \rangle \leftarrow 0 \mid \\
& \langle \text{Variable} \rangle \leftarrow \langle \text{Variable} \rangle \mid \\
& \langle \text{Variable} \rangle \leftarrow \mathrm{S}(\langle \text{Variable} \rangle) \mid \\
& \langle \text{Variable} \rangle \leftarrow \mathrm{P}(\langle \text{Variable} \rangle) \\
\\
\langle \text{Programm} \rangle \quad ::= \quad & \langle \text{Wertzuweisung} \rangle \mid \\
& (\langle \text{Programm} \rangle; \ldots; \langle \text{Programm} \rangle) \mid \\
& \text{while } \langle \text{Variable} \rangle \neq 0 \text{ do } \langle \text{Programm} \rangle \text{ od}
\end{aligned}
$$

Syntax von WHILE

Ist σ ein WHILE-Programm, so verstehen wir unter $Var(\sigma)$ die Menge aller Variablen, die in σ auftreten. Es ist offensichtlich, dass $Var(\sigma)$ immer eine endliche Menge ist.

Definition (Länge)

Allen WHILE-Programmen σ ordnen wir auf folgende Weise induktiv eine Länge $\ell(\sigma)$ zu:

1. Ist σ eine Wertzuweisung, so sei $\ell(\sigma) := 1$.
2. Ist σ von der Form $(\sigma_1; \ldots; \sigma_m)$, so sei $\ell(\sigma) := \ell(\sigma_1) + \ldots + \ell(\sigma_m)$.
3. Ist σ von der Form while $X_i \neq 0$ do π od, so sei $\ell(\sigma) := \ell(\pi) + 1$.

Syntax von WHILE

Beispiel

1. Betrachte das folgende While-Programm σ:

$$\text{while } X_0 \neq 0 \text{ do } X_0 \leftarrow S(X_0) \text{ od}$$

   Wird X_0 mit 0 belegt, so liefert σ den Wert 0; in allen anderen Fällen terminiert σ nicht.

2. Addition $[X_2 = X_0 + X_1]$:

$$(X_2 \leftarrow X_0;$$
$$\text{while } X_1 \neq 0 \text{ do } (X_2 \leftarrow S(X_2);\ X_1 \leftarrow P(X_1)) \text{ od})$$

Semantik von WHILE

Exemplarisch stellen wir im Folgenden eine sogenannte denotationelle Semantik von WHILE vor, die besonders einfach und durchsichtig ist.

Durch Induktion nach der Länge von WHILE-Programmen ordnen wir jedem WHILE-Programm σ mit $Var(\sigma) \subseteq \{X_0, \ldots, X_{k-1}\}$ und jeder natürlichen Zahl $n \geq k$ eine partielle Funktion

$$[\![\sigma, n]\!] : \mathbb{N}^n \to \mathbb{N}^n$$

zu, die man im folgenden Sinn als Bedeutung von σ auffassen kann:

Werden den Variablen $X_0, \ldots, X_{n-1}$ vor Ausführung von σ die Werte $(a_0, \ldots, a_{n-1})$ zugeordnet, so haben sie nach Ausführung von σ die Werte $[\![\sigma, n]\!](a_0, \ldots, a_{n-1})$, falls die partielle Funktion $[\![\sigma, n]\!]$ bei diesem Input einen Wert liefert.

Semantik von WHILE

Definition (Semantik von WHILE)

Für WHILE-Programme $\sigma_0, \ldots, \sigma_{m-1}$ und σ mit

$$Var(\sigma_0) \cup \ldots \cup Var(\sigma_{m-1}) \cup Var(\sigma) \subseteq \{X_0, \ldots, X_{k-1}\},$$

für alle natürlichen Zahlen $n \geq k$ und alle $\vec{a} = a_0, \ldots, a_{k-1}$ setzen wir:

1. $[\![X_i \leftarrow 0, n]\!](\vec{a}) := (a_0, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_{k-1})$,
2. $[\![X_i \leftarrow X_j, n]\!](\vec{a}) := (a_0, \ldots, a_{i-1}, a_j, a_{i+1}, \ldots, a_{k-1})$,
3. $[\![X_i \leftarrow S(X_j), n]\!](\vec{a}) := (a_0, \ldots, a_{i-1}, a_j + 1, a_{i+1}, \ldots, a_{k-1})$,
4. $[\![X_i \leftarrow P(X_j), n]\!](\vec{a}) := (a_0, \ldots, a_{i-1}, a_j \dot{-} 1, a_{i+1}, \ldots, a_{k-1})$,
5. $[\![(\sigma_0; \ldots; \sigma_{m-1})]\!](\vec{a}) :\simeq ([\![\sigma_{m-1}, n]\!] \circ \ldots \circ [\![\sigma_0, n]\!])(\vec{a})$,

Semantik von WHILE

Definition (Semantik von WHILE, Fortsetzung)

6. Ist σ von der Form while $X_i \neq 0$ do π od, so unterscheiden wir:

   1. Es gibt eine natürliche Zahl p mit folgender Eigenschaft: Es existieren n-Tupel $(b_{0,0}, \ldots, b_{0,(n-1)}), \ldots, (b_{p,0}, \ldots, b_{p,(n-1)})$ mit

      (a) $[\![\pi, n]\!]^{(q)}(\vec{a}) = (b_{q,0}, \ldots, b_{q,(n-1)})$ für $0 \leq q \leq p$,

      (b) $b_{p,i} = 0$

      (c) $b_{r,i} > 0$ für $0 \leq r < p$.

      Dann ist dieses p eindeutig bestimmt, und wir setzen

      $[\![\sigma, n]\!](\vec{a}) := [\![\pi, n]\!]^{(p)}(\vec{a})$.

   2. Es gibt keine natürliche Zahl p mit dieser Eigenschaft:

      Dann sei $[\![\sigma, n]\!](\vec{a})$ undefiniert.

Semantik von WHILE

Beispiel

Wir betrachten noch einmal das While-Programm σ

$$\text{while } X_0 \neq 0 \text{ do } X_0 \leftarrow S(X_0) \text{ od}$$

und ordnen ihm eine Denotationsfunktion zu. Zuerst die Wertzuweisung $X_0 \leftarrow S(X_0)$: Sei $f := [\![X_0 \leftarrow S(X_0), 1]\!]$, also $f(a) = a + 1$.

Ist daher $a = 0$, so gibt es eine natürliche Zahl p mit

$$f^{(p)}(a) = 0 \qquad \text{und} \qquad f^{(q)}(a) > 0, \quad \text{für alle } q < p$$

nämlich $p = 0$. Ist $a \neq 0$, so gibt es kein solches p. Daher gilt:

$$[\![\sigma, 1]\!](a) = \begin{cases} 0, & \text{falls } a = 0; \\ \text{undefiniert}, & \text{falls } a \neq 0. \end{cases}$$

Semantik von WHILE

Bemerkungen

1. Wir haben bereits weiter oben erwähnt, dass alle im Prinzip berechenbaren partiellen zahlentheoretischen Funktionen durch WHILE-Programme berechnet werden können.
2. Es gibt eine Teilklasse der WHILE-Programme – die sogenannten LOOP-Programme – mit denen genau die primitiv-rekursiven Funktionen berechnet werden können.
3. Für die Informatik sind neben der Frage nach der prinzipiellen Berechenbarkeit natürlich auch die Zeit- und Platzkomplexität von Berechnungen von grosser Bedeutung. Auf dieses Thema wird in einer späteren Lektion eingegangen.

Drei Hauptströmungen

Grundlagen der Mathematik

- Logizismus (Frege, Russell): Logik ist die universelle Grundlage der Mathematik; Mathematik geht von Logik aus und baut auf ihr auf.
- Formalismus (Hilbert): Mathematik kann im Rahmen eines Formalismus realisiert werden und reduziert sich letztlich auf die Manipulation formaler Regeln; dieser Formalismus wird dadurch gerechtfertigt, dass seine Konsistenz nur durch endliches Schliessen nachgewiesen wird.
- Intuitionismus (Brouwer): Nur diejenigen Objekte existieren, die sich konstruieren lassen; mathematische Beweise sind in diesem Sinne ebenfalls Konstruktionen.

Konstruktive Beweise

Intuitionismus bildet die Grundlage der konstruktiven Mathematik; die Wahrheit einer mathematischen Aussage wird durch einen konstruktiven Beweis etabliert.

Zwei Hauptforderungen an konstruktive Beweise:

- Disjunktionseigenschaft – Ein Beweise $\mathcal{B}$ einer Disjunktion $(F \vee G)$ besteht darin, eines der beiden Disjunktionsglieder zu beweisen.
- Existenzeigenschaft – Ein Beweis $\mathcal{B}$ einer Existenzaussage $\exists x F[x]$ besteht in der Angabe eines Objekts t und in einem Beweis von $F[t]$.

Aufgrund der Disjunktionseigenschaft ist das berühmte *Tertium Non Datur* – d.h. die Aussage $(F \vee \neg F)$ – intuitionistisch (konstruktiv) nicht gültig.

Ein nichtkonstruktiver Beweis

Theorem

Es gibt irrationale Zahlen a und b, so dass a^b rational ist.

Beweis. Wir wissen, dass $\sqrt{2}$ irrational ist (Schulstoff) und unterscheiden nun zwei Fälle:

$$\sqrt{2}^{\sqrt{2}} \text{ ist rational } \textbf{oder } \sqrt{2}^{\sqrt{2}} \text{ ist irrational.}$$

1. Fall: Dann setze $a := \sqrt{2}$ sowie $b := \sqrt{2}$, und unsere Behauptung ist bewiesen.

2. Fall: Dann setze $a := \sqrt{2}^{\sqrt{2}}$ sowie $b := \sqrt{2}$. Da dann aber

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2$$

gilt, haben wir auch in diesem Fall zwei irrationale Zahlen, deren Potenz rational ist. □

Ein nichtkonstruktiver Beweis

Der obige Beweis verletzt die Disjunktionseigenschaft. Obwohl er zeigt, dass es irrationale Zahlen a und b gibt, deren Potenz rational ist, liefert er keinerlei Information darüber, ob wir als a die Zahl $\sqrt{2}$ oder die Zahl $\sqrt{2}^{\sqrt{2}}$ wählen müssen.

Der Grund dafür ist die Verwendung der Tertium Non Datur in Form der Fallunterscheidung zu Beginn des Beweises.

Programme aus Beweisen

Grundidee

Gegeben seien die beiden Datentypen D_1 und D_2 sowie die formale Spezifikation *Spec*. Ausserdem sei $\mathcal{B}$ ein konstruktiver Beweis der Aussage, dass für jedes Element x aus D_1 ein Element y aus D_2 existiert, so dass die Spezifikation *Spec* erfüllt ist,

$$\mathcal{B} \vdash (\forall x \in D_1)(\exists y \in D_2) Spec[x, y].$$

Dann kann der Beweis $\mathcal{B}$ in ein Programm $\pi_{\mathcal{B}}$ transformiert werden, das für jeden Input $x \in D_1$ einen Output $\pi_{\mathcal{B}}(x) \in D_2$ berechnet, für den

$$Spec[x, \pi_{\mathcal{B}}(x)].$$

Im Folgenden stellen wir verschiedene Ansätze vor, die versuchen, diese sehr einfache Grundidee zu präzisieren.

Brouwer-Heyting-Kolmogorov-Interpretation (BHK)

Entsprechend der BHK-Interpretation ist ein Beweis von

$(F_1 \vee F_2)$	ein Paar $(i, \mathcal{B})$, so dass $i \in \{1, 2\}$ gilt und $\mathcal{B}$ ein Beweis von F_i ist;
$(F_1 \wedge F_2)$	ein Paar $(\mathcal{B}_1, \mathcal{B}_2)$, so dass $\mathcal{B}_1$ ein Beweis von F_1 und $\mathcal{B}_2$ ein Beweis von F_2 ist;
$(F_1 \to F_2)$	eine Methode, die jeden Beweis von F_1 in einen Beweis von F_2 transformiert;
$(\exists x \in D)F[x]$	ein Paar $(d, \mathcal{B}_d)$, das aus einem Element $d \in D$ und einem Beweis $\mathcal{B}_d$ von $F[d]$ besteht;
$(\forall x \in D)F[x]$	eine Methode, die für jedes $d \in D$ einen Beweis $\mathcal{B}_d$ von $F[d]$ liefert.

Brouwer-Heyting-Kolmogorov-Interpretation (BHK)

Unmittelbare Folgerungen

- Es gibt einen Beweis von $(F \vee \neg F)$ im Sinne der BHK-Interpretation nur dann, wenn F oder $\neg F$ BHK-beweisbar ist; im Allgemeinen ist aber $(F \vee \neg F)$ nicht BHK-beweisbar.
- Ein BHK-Beweis einer Aussage

$$(\forall x \in D_1)(\exists y \in D_2)F[x, y]$$

liefert eine Methode (ein Programm), die für jedes Objekt $x \in D_1$ ein Objekt $y \in D_2$ sowie einen BHK-Beweis von $F[x, y]$ bereitstellt.

Curry-Howard Formulas as Types

Ein verwandter Ansatz, der auf Curry und Howard zurückgeht, besteht darin, Formeln als Typen zu interpretieren. Primär betrachtet man dabei Ausdrücke der Form

$$t : T,$$

die bedeuten, dass T ein Datentyp und t ein Term vom Typ T ist.

Die Grundidee besteht nun darin, Formeln als Typen und Beweise als geeignete Beweisterme zu interpretieren. So gesehen bedeutet dann $t : T$:

- T ist ein Datentyp, der einer Formel F zugeordnet wird;
- t ist ein Term, der einen Beweis von F kodiert;
- $t : T$ drückt also sowohl aus, dass t einen Beweis von F kodiert, als auch, dass t ein Objekt vom Datentyp T ist.

Curry-Howard Formulas as Types

Um diese Korrespondenz zwischen Formeln und Typen deutlich zu machen, ordnen wir Formeln A auf folgende Weise Typen $\|A\|$ zu:

$$\begin{aligned}
\|F \vee G\| &:= \{(0,x) : x \in \|F\|\} \cup \{(1,x) : x \in \|G\|\};\\
\|F \wedge G\| &:= \|F\| \times \|G\|;\\
\|F \to G\| &:= \text{geeignete Teilmenge von } \|G\|^{\|F\|};\\
\|(\exists x \in D)F[x]\| &:= \{(x,f) : x \in D \text{ und } f \in \|F[x]\|\};\\
\|(\forall x \in D)F[x]\| &:= \prod_{x \in D} \|F[x]\|.
\end{aligned}$$

Dabei wird die Interpretation der atomaren Formeln vorläufig noch offengelassen.

Curry-Howard Formulas as Types

Bei der Interpretation von Implikationen $F \to G$ gewisse Flexibilität; je nach intendierter Anwendung kann man sich z.B. beschränken auf:

- alle konstruktiven Funktionen von $\|F\|$ nach $\|G\|$;
- alle im Sinne einer geeigneten Topologie stetigen Funktionen von $\|F\|$ nach $\|G\|$.

Ist die Typenstruktur fixiert, können die entsprechenden Terme im Rahmen des sogenannten getypten Lambdakalküls oder der sogenannten getypten kombinatorischen Logik definiert werden.

Die so erhaltenen Terme können direkt als funktionale Programme gelesen werden.

Curry-Howard Formulas as Types

Theorem (Curry-Howard-Isomorphismus)

Für die konstruktive Logik erster Stufe besteht der folgende Isomorphismus zwischen Beweisen und entsprechend getypten Termen:

Einem Beweis $\mathcal{B}$ einer Formel G ausgehend von den Annahmen $F_1, \ldots, F_n$ entspricht ein Term $t_{\mathcal{B}}(x_1, \ldots, x_n)$, so dass

$$x_1 : \|F_1\|, \ldots, x_n : \|F_n\| \quad \Rightarrow \quad t_{\mathcal{B}}(x_1, \ldots, x_n) : \|G\|.$$

Ersetzen wir also die Variablen $x_1, \ldots, x_n$ durch Beweisterme $t_{F_1}, \ldots, t_{F_n}$, die Beweise der Formeln $F_1, \ldots, F_n$ kodieren, so kodiert $t_{\mathcal{B}}(t_{F_1}, \ldots, t_{F_n})$ einen Beweis von G.

Curry-Howard Formulas as Types

Basierend auf dem Curry-Howard-Isomorphismus ergibt sich eine Reihe interessanter Korrespondenzen:

- Überprüfung Beweiskorrektheit $\approx$ Überprüfung Typenkorrektheit
- Beweissuche für F $\approx$ Suche eines Elements von $\|F\|$
- Ein Beweis $\mathcal{B}$ einer Aussage

$$(\forall x \in D_1)(\exists y \in D_2)F[x, y]$$

liefert einen Beweisterm $t_{\mathcal{B}}$, der als Code eines (funktionales) Programm $\pi_{\mathcal{B}}$ verstanden werden kann, das für jedes $x \in D_1$ ein $\pi_{\mathcal{B}}(x) \in D_2$ berechnet, für das $F[x, \pi_{\mathcal{B}}(x)]$ gilt. Dieser Beweis $\mathcal{B}$ ist zugleich eine Verifikation von $\pi_{\mathcal{B}}$!

Ausblick

Die in dieser Lektion skizzierten Ideen und die entsprechenden Konzepte sind in einer ganzen Reihe von Projekten realisiert. Wir beschränken uns zum Abschluss darauf, einige Beispiele zu nennen:

- Automath (De Bruijn)
- Systeme (konstruktiver) Typentheorie (e.g. Martin-Löf Type Theory) und Lambda-Kalkül als theoretische Basis
- Logical Frameworks (zu viele Beitragende, um sie aufzulisten)
- General-Purpose Theorem Proving Systems für die Formalisierung (konstruktiver) Mathematik, z.B.: Coq, HOL, LEGO, Mizar und Nuprl.