

Formale Grundlagen der Informatik 3 –

4. Beweise, Kalküle, Herleitungen

Christoph Walther
TU Darmstadt

1 Beweiskalküle

- Grundlagenkrise der Mathematik ab 1900
- **Frage:** Wie vertrauenswürdig sind mathematische Methoden, wenn man sowohl eine Behauptung als auch deren Negat nachweisen kann ?
- **Frage:** Wie vertrauenswürdig sind Behauptungen über mathematische Sachverhalte, wenn die Methoden, mit denen diese nachgewiesen wurden, informell und nicht allgemein anerkannt sind ?
- **Beispiel Antinomie:**
 - *informell:* In einem Dorf gibt es einen Barbier, der alle Männer rasiert, die sich nicht selbst rasieren. Also rasiert sich der Barbier selbst, genau dann wenn er sich nicht selbst rasiert !
 - *formal:* Wenn $M := \{N \mid N \notin N\}$, dann $M \in M$ gdw. $M \notin M$ (denn falls $M \in M$, so auch $M \notin M$, und falls $M \notin M$, dann $M \in M$).
- **Beispiel Induktion:** “Wenn $P(0)$ gezeigt ist, und $P(n) \rightarrow P(n + 1)$ für jedes $n \in N$ gezeigt ist, so hat man auch $P(n)$ für jedes $n \in N$ gezeigt”.
Warum ist das überhaupt eine gültige Schlußweise? Warum darf man zum Nachweis von $P(n + 1)$ annehmen, daß $P(n)$ wahr ist ?

Lösung für diese Probleme:

- (Formale) *Logik* (= Meta-Mathematik = Mathematik über Mathematik) durch

(1) *Normierung formaler Sprachen*, in der mathematische Sachverhalte *repräsentiert* werden

- **Beispiel:** Sprache des Sequenzenkalküls (\Rightarrow FGdI 2)
- **Beispiel:** Klauselsprache des Resolutionskalküls (\Rightarrow FGdI 2)
- **Konsequenz der Normierung** (beispielsweise):
Definitionen wie “ $\{N \mid N \notin N\}$ ” sind verboten

(2) *Normierung* mathematischer *Schlußweisen* durch *Beweiskalküle*:

Mathematische Schlüsse werden durch *Regeln* beschrieben, die *Ausdrücke* einer Sprache in (andere) Ausdrücke dieser *Sprache* transformieren

- **Reine Symbolmanipulation:** Worte (über einem Alphabet Σ) werden in andere Worte transformiert, nichts weiter !
- **Absicht:** Nachweis von (wahren) mathematischen Behauptungen durch reine Symbolmanipulation
- **Damit beispielsweise:** Einfache Prüfung, ob Schlußweisen, die zum Nachweis einer Behauptung führen sollen, tatsächlich korrekt sind – dazu muß man kein Mathematiker sein, sondern nur die Beweisregeln kennen

- **Analogie:** Man muß kein Schachspieler sein, um zu überprüfen, ob die Zugfolgen in einer Schachpartie legal waren – man muß lediglich die Schachregeln kennen
 - **Beispiel:** Regeln des Sequenzenkalküls (\Rightarrow FGdI 2)
 - **Beispiel:** Regeln des Resolutionskalküls (\Rightarrow FGdI 2)
- (3) **Herleitungen:** *Formalisierung* des Begriffs “*Beweis*”
- **Definition von:** Was genau ist “*formaler Beweis*” in einem Kalkül?
 - **Beispiel:** Eine *Herleitung* aus einer Sequenz s im *Sequenzenkalkül* ist ein *Baum*, dessen Wurzel mit s und dessen Knoten mit Sequenzen markiert sind, die durch Anwendung der *Sequenzenregeln* entstehen;
ein *formaler Beweis* der *Allgemeingültigkeit* einer Sequenz s ist eine Herleitung aus s , deren *Blätter mit Axiomen* markiert sind (\Rightarrow FGdI 2)
 - **Beispiel:** Eine *Herleitung* aus einer *Klauselmenge* S im *Resolutionskalkül* ist ein *Baum*, dessen Knoten mit Klauseln markiert sind, die aus S stammen oder die durch Anwendung der Resolutions- und Faktorisierungsregel (auf Klauseln im Baum) entstehen ;
ein *formaler Beweis* der *Unerfüllbarkeit* von S ist eine Herleitung aus S , deren *Wurzel* mit der *leeren Klausel* markiert ist (\Rightarrow FGdI 2)

Zusammenfassung: Ein *Kalkül* K ist *definiert* durch *Angabe*

(1) einer *formalen Sprache* L

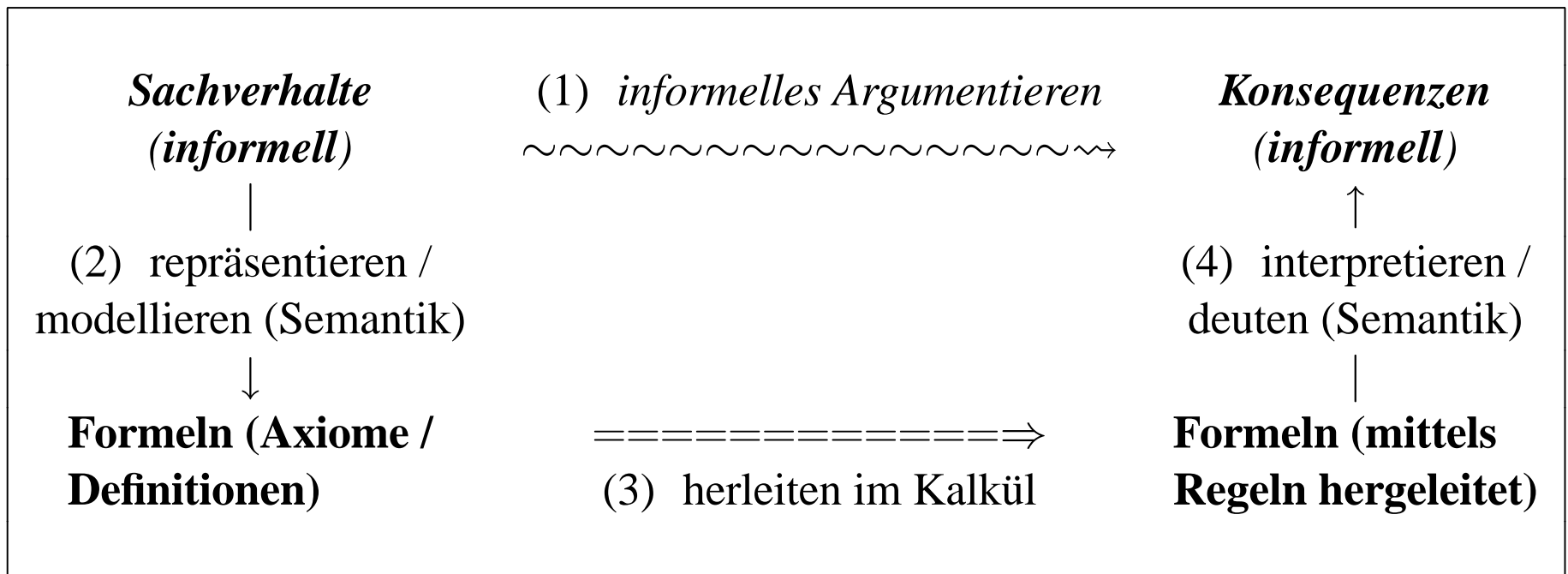
(2) den *Regeln* R von K

(3) den mittels R gebildeten *Herleitungen* in K

- **Jetzt weiter:** Um *informelles Argumentieren* durch *Herleitungen* eines Kalküls *zu ersetzen*, muß für die Sprache des Kalküls eine **Semantik** (= Bedeutung) definiert werden
 - Mit Definition der *Semantik* wird formal präzise festgelegt, was *Ausdrücke der Sprache bedeuten*
 - **Beispiel:** *Wahrheitstafeln* und \mathcal{V} -*Interpretationen* bzw. \mathcal{V} -*Belegungen* zur Deutung von Ausdrücken der Sprache der *Aussagenlogik* (\Rightarrow FGdI 2)
 - **Beispiel:** *S-Strukturen* und *S-Interpretationen* zur Deutung von Ausdrücken der Sprache der *Prädikatenlogik* (\Rightarrow FGdI 2)
 - **Offensichtliche Forderung:** Die *Kalkülregeln* müssen die *Sprachsemantik* “*respektieren*” (\Rightarrow nur Schlüsse, die bzgl. der Semantik gültig sind)
 - **Ergebnis:** Mit Kalkülen *Formalisierung* (= “*Mathematisierung*”) *informeller* mathematischer *Schlußweisen*

Also Vorgehen:

- (1) Anstatt ausgehend von informell gegebenen Sachverhalten durch *informelles Argumentieren* informelle Konsequenzen zu postulieren
- (2) werden (ausgehend von der Sprachsemantik) die Sachverhalte *durch Formeln modelliert / repräsentiert* (=> **Kapitel 3: Validierungsproblem !**)
- (3) daraus weitere *Formeln mittels der Kalkülregeln hergeleitet*, und diese dann
- (4) (ausgehend von der Sprachsemantik) in der *informellen “Welt” gedeutet / interpretiert* (=> **Kapitel 3: Validierungsproblem !**)



Nutzen: Jetzt kann bei (3) mathematisch präzise untersucht werden

- Was ist prinzipiell (nicht) beweisbar ?
- Was ist (nicht) automatisch beweisbar ?
- Ist jede bewiesene Behauptung auch wahr (Korrektheit) ?
- Kann jede wahre Behauptung auch bewiesen werden (Vollständigkeit) ?
- Ist entscheidbar, ob eine Behauptung beweisbar ist ?
- ...

Beispiele und Resultate

Aussagenlogik

- (1) *Sprache*: Formeln werden ausgehend von Aussagenvariablen mittels der üblichen logischen Verknüpfungen $\wedge, \vee, \rightarrow, \neg$ gebildet
 - (2) *Regeln*: Beispielsweise aussagenlogischer Sequenzenkalkül (\Rightarrow FGdI 2)
 - (3) *Herleitungen*: mit Formeln markierte Bäume (\Rightarrow FGdI 2)
- *Semantik*: Durch Belegung für Aussagenvariablen sowie Festlegung der Bedeutung der logischen Verknüpfungen (z.B. durch Wahrheitstafeln)
 - *Resultate*:
 - (a) es gibt *korrekte* und *vollständige* Kalküle für die Aussagenlogik (\Rightarrow FGdI 2);
 - (b) Wenn eine Formel *eine Tautologie* ist, so kann man das maschinell feststellen (\Rightarrow Menge der Tautologien ist *rekursiv aufzählbar*);
 - (c) Wenn eine Formel *keine Tautologie* ist, so kann man das maschinell feststellen (\Rightarrow Komplement der Menge der Tautologien ist *rekursiv aufzählbar*);
 - (d) Konsequenz aus (b) und (c): Menge der Tautologien ist *entscheidbar*.

Prädikatenlogik 1. Stufe (PL 1)

- (1) *Sprache*: Formeln werden ausgehend von Variablen-, Funktions- und Prädikatssymbolen mittels der üblichen logischen Verknüpfungen \wedge , \vee , \rightarrow , \neg sowie der Quantoren \forall und \exists gebildet
- (2) *Regeln*: Beispielsweise Sequenzenkalkül (\Rightarrow FGdI 2)
- (3) *Herleitungen*: mit Formeln markierte Bäume (\Rightarrow FGdI 2)
- *Semantik*: Formeln werden mittels *Interpretationen* gedeutet; eine Formel ist *allgemeingültig* gdw. sie in *jeder Interpretation* wahr ist (\Rightarrow FGdI 2)
- *Resultate*:
 - (a) es gibt *korrekte* und *vollständige* Kalküle für PL 1 (\Rightarrow FGdI 2)¹
 - (b) man kann maschinell feststellen, ob eine Formel *allgemeingültig* ist (\Rightarrow Menge der allgemeingültigen Formeln ist *rekursiv aufzählbar*)
 - (c) man kann *nicht* maschinell feststellen, ob eine Formel *nicht allgemeingültig* ist (\Rightarrow Menge der allgemeing. Formeln *nicht rekursiv aufzählbar*)
 - (d) Konsequenz aus (b) und (c): Menge der *allgemeingültigen Formeln* ist *nicht entscheidbar*²

¹ Kurt Gödel: Über die Vollständigkeit des Logikkalküls, Dissertation, 1929.

² Alonzo Church: A note on the Entscheidungsproblem, Journal of Symbolic Logic, 1936.

Alan Turing: On Computable Numbers, with an Application to the Entscheidungsproblem, Proceedings

Formale Arithmetik / Peano Arithmetik

- (1) *Sprache*: Wie PL 1 jedoch mit einer Signatur, die mindestens Funktionssymbole für 0, Nachfolgerfunktion, Addition und Multiplikation enthält
 - (2) *Regeln*: Wie PL 1 sowie weitere Axiome für *Induktion*, *Addition*, ...
 - (3) *Herleitungen*: Wie PL 1
- *Semantik*: Formeln werden mittels einer festen *Standardinterpretation* gedeutet, die die Symbole für 0, Nachfolgerfunktion, Addition und Multiplikation so wie die entsprechenden Funktionen in \mathbb{N} deuten; eine *Formel gilt* gdw. sie *in der Standardinterpretation* wahr ist
 - *Resultate*:
 - (a) es gibt *korrekte* Kalküle für die formale Arithmetik
 - (b) es gibt *keine korrekten und vollständigen* Kalküle für die formale Arithmetik³
 - * für jeden korrekten Kalkül kann man eine Formel angeben, von der gezeigt werden kann, daß diese *in der Standardinterpretation wahr* jedoch *nicht im Kalkül beweisbar* ist

of the London Mathematical Society, Second Series 42:230–265, 1936.

³ Kurt Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, Monatshefte für Mathematik und Physik 38, 1931, S.173–198.

- (c) *Damit*: Es kann maschinell *nicht* festgestellt werden, ob eine Formel der Arithmetik *wahr oder falsch* ist (\Rightarrow weder ist die Menge der wahren Formeln der Arithmetik rekursiv aufzählbar, noch ist es deren Komplement)
- (d) *Insbesondere dann*: Die Menge der wahren Formeln der Arithmetik ist *nicht entscheidbar*.

Bemerkung 1

- Die Ergebnisse für die *Formale Arithmetik* gelten sinngemäß auch für Kalküle zur Programmverifikation:
Grund: Da Programmiersprachen *Schleifen* und/oder *Rekursion* besitzen kann die Formale Arithmetik mittels Programmen modelliert werden.
- **Konsequenz 1**: Jeder korrekte Kalkül für Programmbeweise ist unvollständig (denn es gibt wahre Aussagen, die nicht beweisbar sind)
- **Konsequenz 2**: Es ist nicht entscheidbar, ob Behauptungen über ein Programm wahr sind.

Weitere Beispiele für Beweiskalküle

- Die *Symbolische Auswertung* in *VeriFun* wird durch einen Beweiskalkül für *Gleichheitsbeweise* implementiert
- Der *HPL-Kalkül* in *VeriFun* ist ein Beweiskalkül für *Induktionsbeweise*
- Der π -*Kalkül* ist ein Kalkül für Beweise über *nebenläufige Systeme*
- Der μ -*Kalkül* ist ein Kalkül für Beweise in *temporaler Logik*
- Der *Hoare-Kalkül* ist ein Kalkül für Korrektheitsbeweise über Programme *imperativer Programmiersprachen*

2 Beweise in der Mathematik

Unterscheide: *Verwendung* von Kalkülen

- um mathematische Schlußweisen *zu untersuchen*
(\Rightarrow *Mathematik / formale Logik*)
- um Beweise tatsächlich *zu führen*
(\Rightarrow *Informatik*)

Nachteile von *Kalkülbeweisen* (= *formalen Beweisen*) :

- Da *jeder einzelne Beweisschritt* mittels einer Kalkülregel *protokolliert* werden muß, sind Kalkülbeweise
 - aufwendig zu führen
 - umfangreich
 - unübersichtlich
 - umständlich
 - schwer lesbar
 - ...

- **Konsequenz:**
 - In Mathematikbüchern (und anderen Veröffentlichungen) findet man keine Kalkülbeweise
 - “Offensichtliche” Schlüsse werden weggelassen (= *Makroschritte*)
 - * “wie man leicht sieht, gilt ...”
 - * “ohne Beschränkung der Allgemeinheit nehmen wir an, daß ...”
 - Intuitive Notationen
 - * “...” anstatt rekursive Definition

- Ist das jetzt wieder unpräzise ???
 - *Das geschulte Auge erkennt:* Man könnte den Beweis auch streng formal führen, also sind die Schlüsse gültig.
 - *Allerdings:* Fehler können schon mal übersehen werden

Ein Beispiel

- Zu zeigen sei “ $\sqrt{2}$ ist irrational”
- **Also:** $\forall q \in \mathbb{Q} q \neq \sqrt{2}$
- **Umgeformt (1):** $\forall x, y \in \mathbb{Z} y \neq 0 \curvearrowright x/y \neq \sqrt{2}$
- **Umgeformt (2):** $\forall x, y \in \mathbb{Z} y \neq 0 \curvearrowright x \neq \sqrt{2} y$
- **Umgeformt (3):** $\forall x, y \in \mathbb{Z} y \neq 0 \curvearrowright x^2 \neq 2y^2$
- **Umgeformt (4):** $\forall x, y \in \mathbb{N} y \neq 0 \curvearrowright x^2 \neq 2y^2$

Jetzt Beweis dafür ...

Behauptung (*) $\forall x, y \in \mathbb{N} \ y \neq 0 \leadsto x^2 \neq 2y^2$

Beweis:

(1) *Annahme:* Behauptung ist falsch, d.h. es gelte

$$(i) \ \exists a, b \in \mathbb{N} \ b \neq 0 \wedge a^2 = 2b^2$$

(2) *O.B.d.A.:* $a + b$ ist minimal, d.h. es gilt

$$(ii) \ \forall x, y \in \mathbb{N} \ x + y < a + b \wedge y \neq 0 \leadsto x^2 \neq 2y^2$$

(3) Wegen (i) ist a^2 eine gerade Zahl $\neq 0$, und folglich gilt

$$(iii) \ a \text{ ist eine gerade Zahl } \neq 0$$

(4) Wegen (iii) gilt für ein $k \in \mathbb{N}$

$$(iv) \ a = 2k + 2 = 2(k + 1)$$

(5) Mit (i) und (iv) folgt $4(k + 1)^2 = 2b^2$, also $2(k + 1)^2 = b^2$, also

$$(v) \ 2(a/2)^2 = b^2$$

(6) Setze in (ii) $x := b$ und $y := a/2$: Dann $b^2 \neq 2(a/2)^2$ mit (ii) (denn $b + a/2 < a + b$) im Widerspruch zu (v) !

(7) *Konsequenz:* Annahme (i) ist falsch, und folglich ist (*) wahr. ■

Beweisanalyse (1)

Beweis von Behauptung $(*)$ ist ein *indirekter Induktionsbeweis*
(= Induktionsbeweis, der als Widerspruchsbeweis geführt wird)

- Prinzip für *direkten Induktionsbeweis* einer Behauptung $\forall x \in \mathbb{N} \phi[x]$:
 - Zeige $\phi[0]$ (= Induktionsanfang)
 - Zeige $\forall x \in \mathbb{N} \phi[x] \rightsquigarrow \phi[x + 1]$ (= Induktionsschritt)
 - Mit dem Induktionsaxiom gilt dann $\forall x \in \mathbb{N} \phi[x]$
- Prinzip für *indirekten Induktionsbeweis* einer Behauptung $\forall x \in \mathbb{N} \phi[x]$:
 - *Annahme*: Behauptung ist falsch.
 - Dann ist $\exists x \in \mathbb{N} \neg\phi[x]$ wahr
 - Damit ist $M := \{x \in \mathbb{N} \mid \neg\phi[x]\}$ nicht leer.
 - Zeige $0 \notin M$ (= Induktionsanfang)
 - Mit $0 \notin M$ können wir ein $a \in M$ wählen, so daß $a \neq 0$ und $a - 1 \notin M$
 - Zeige, daß aus $\neg\phi[a]$ auch $\neg\phi[a - 1]$ folgt (= Induktionsschritt)
 - Dann gilt $a - 1 \in M$, also Widerspruch !

- Induktionsschema für direkten Induktionsbeweis von Behauptung $(*)$:
 - $\forall x, y \in \mathbb{N} \ x = 0 \wedge y \neq 0 \curvearrowright x^2 \neq 2y^2$ (= Induktionsanfang)
 - $\forall x, y \in \mathbb{N} \ x \neq 0 \wedge [x/2 \neq 0 \curvearrowright y^2 \neq 2(x/2)^2] \curvearrowright [y \neq 0 \curvearrowright x^2 \neq 2y^2]$
(= Induktionsschritt; Beobachtung: $y + x/2 < x + y$ wenn $x \neq 0$)
 - Damit gilt auch $\forall x, y \in \mathbb{N} \ y \neq 0 \curvearrowright x^2 \neq 2y^2$
- Sehr ungewöhnliches Induktionsprinzip !
Warum gilt das überhaupt ??? (Antwort => **Kapitel 6**)

Beweisanalyse (2)

Beweis von Behauptung (*) enthält zahlreiche *Makroschritte* (= “Lücken”)

- “O.B.d.A.: $a + b$ ist minimal” (warum darf man das annehmen ?)
- Folgerung “ a^2 ist gerade Zahl $\neq 0$ ” (Beweis ?)
- Folgerung “ a^2 ist gerade $\curvearrowright a$ ist gerade” (Beweis ?)
- Folgerung “ $a/2 \neq 0$ ” (Beweis ?)
- Folgerung “ $2(a/2)^2 = b^2$ ” (Beweis ?)
- ...

\Rightarrow *In Kalkülbeweisen müssen diese Lücken gefüllt werden, selbst triviale Schritte wie “ $4(k + 1)^2 = 2b^2 \curvearrowright 2(k + 1)^2 = b^2$ ” müssen durch Regelanwendungen protokolliert werden*

3 Warum Logik in der Informatik ?

- Die Mathematik ist ein mächtiges Werkzeug mit dem viele Gegenstandsbereiche *formal präzise modelliert und untersucht* werden können
 - *“In jeder Wissenschaft steckt soviel Wahrheit, wie Mathematik in ihr steckt.”*
(Immanuel Kant)
 - *“Die Logik ist keine Lehre, sondern ein Spiegelbild der Welt.”*
(Ludwig Wittgenstein)
 - Allerdings auch: *“Wenn man aber einmal mit Logik beginnt, wo ein Gedanke von selbst aus dem vorangehenden folgt, weiß man zum Schluß nie, wie das endet.”* (Robert Musil)
- Für jede Ingenieurdisziplin fundierte mathematische Grundlagen erforderlich:
 - Welche Mathematikdisziplin jeweils gut geeignet ist, hängt von dem konkreten Gegenstandsbereich ab (Bauingenieurwesen, Maschinenbau, Elektrotechnik, u.s.w.).
- Die formale Logik ist die Mathematikdisziplin, die für die Informatik gut geeignet ist, denn
 - Informatiker müssen einen Gegenstandsbereich formal erfassen, damit Aufgaben aus diesem Bereich durch einen Rechner überhaupt verarbeitet werden können.

- Damit werden formale Sprachen und deren Semantik definiert, also Begriffe, die uns schon die formale Logik bereitstellt.
- Die Verarbeitung durch einen Rechner wird durch Kalkülregeln modelliert, also ebenfalls ein Begriff, den uns schon die formale Logik bereitstellt.
- Damit kann man jetzt für eine konkrete Anwendung untersuchen:
 - (i) Sind alle Berechnungen korrekt, d.h. ist eine berechnete Lösung auch wirklich die Lösung eines Problems (Korrektheit) ?
 - (ii) Kann für jedes lösbare Problem auch eine Lösung berechnet werden (Vollständigkeit) ?
 - (iii) Können lösbare und unlösbare Probleme durch einen Rechner immer erkannt werden (Entscheidbarkeit) ?
 - (iv) ...

Also:

- *Formale Repräsentation* eines Gegenstandsbereichs durch (syntaktisch korrekte) Ausdrücke der Sprache (*Modellierung* => FOC Kanonik).
- *Semantik* der Sprache legt fest, was ein Ausdruck *im Gegenstandsbereich* bedeutet.
- *Gültige Schlüsse* im Gegenstandsbereich werden durch *Kalkülregeln* modelliert.
- **Zweck:**
 - Durch formale Modellierung werden Unklarheiten, Mehrdeutigkeiten, Widersprüchlichkeiten in der Erfassung des Gegenstandsbereichs aufgedeckt (=> FOC Kanonik).
 - Präzise nachvollziehbare Aussagen über Gegenstandsbereich (Inspektion einer Kalkülherleitung bzgl. der Aussage).
 - Kalkülherleitungen können durch Rechner durchgeführt werden, also anstatt Schlüsse im Gegenstandsbereich selbst durchzuführen überlassen wir das einem Rechner.

4 Beweise in der Informatik

Nochmal: Ein *Kalkül* K ist *definiert* durch *Angabe*

- (1) einer *formalen Sprache* L
 - Das kennen wir schon aus der Informatik
- (2) den *Regeln* R von K
 - Reine *Symbolmanipulation*, also auch bekannt aus der Informatik
 - Kalkülregeln = *Erzeugungsvorschrift* für *Bäume*
- (3) den mittels R gebildeten *Herleitungen* in K
 - (mit Formeln markierte) *Bäume*, also auch bekannt aus der Informatik

Fazit:

- Der *Kalkülbegriff* verwendet *Konzepte der Informatik*
- (*Historisch korrekt:* Die Informatik verwendet Konzepte der formalen Logik)
- **Also:** *Formeln* und *Herleitungen* sind *Datenobjekte*, die durch *Algorithmen* bearbeitet werden können
- **Damit:** Der *Kalkülbegriff* verwendet *Konzepte*, die eine *maschinelle Verarbeitung* erlauben

Konsequenz:

- *Nachteile von Kalkülbeweisen aufgehoben durch Rechnerunterstützung*
- *Triviale Beweisschritte (= Lücken bei Makroschritten) können automatisch berechnet werden*
- **Damit:** absolute Fehlerfreiheit (relativ zur korrekten Implementierung des verwendeten Beweissystems)

Beispiel: Beweis von “ $\forall x, y \in \mathbb{N} y \neq 0 \hookrightarrow x^2 \neq 2y^2$ ” in *VeriFun*

Benötigte Prozeduren:

```
function [infixr,10] +(x : ℕ, y : ℕ) : ℕ <=  
if ?0(x) then y else +(^(x) + y) end_if  
berechnet Summe zweier natürlichen Zahlen
```

```
function 2*(x : ℕ) : ℕ <=  
if ?0(x) then 0 else +(^(2*(^(x)))) end_if  
berechnet Verdopplung einer natürlichen Zahl
```

```
function [postfix] ^2(x : ℕ) : ℕ <=  
if ?0(x) then 0 else +((^(x))^2 + 2*(^(x))) end_if  
berechnet Quadrat einer natürlichen Zahl
```

```

function  $\frac{1}{2}(x : \mathbb{N}) : \mathbb{N} \leq =$ 
if ?0(x)
  then 0
  else if ?0( $\neg(x)$ )
    then 0
    else  $+(\frac{1}{2}(\neg(\neg(x))))$ 
  end_if
end_if

```

berechnet nach unten gerundete Division durch 2

```

function  $\mathfrak{R}(x : \mathbb{N}, y : \mathbb{N}) : \mathbb{N} \leq =$ 
if ?0(x) then 0 else  $\mathfrak{R}(y, \frac{1}{2}(x))$  end_if

```

Ergebnis irrelevant; Aus \mathfrak{R} gewinnt man das *Induktionsschema* für den Beweis der Behauptung.

Anders gesagt: Das verwendete Induktionsschema wird *indirekt* durch die Prozedur \mathfrak{R} angegeben. (*Wie & Warum* => **Kapitel 7**)

Benötigte Lemmata:

lemma + is associative <= $\forall x, y, z : \mathbb{N}$

$$(x + y) + z = x + y + z$$

automatischer Beweis

lemma + is commutative <= $\forall x, y : \mathbb{N} \ x + y = y + x$

automatischer Beweis

lemma + is left-cancelable <= $\forall x, y, z : \mathbb{N}$

$$\text{if}\{x + y = x + z, y = z, \text{true}\}$$

automatischer Beweis

lemma $x > y \rightarrow x + z > y + z$ <= $\forall x, y, z : \mathbb{N}$

$$\text{if}\{x > y, x + z > y + z, \text{true}\}$$

automatischer Beweis

lemma $x + x = 2*(x)$ <= $\forall x : \mathbb{N} \ x + x = 2*(x)$

automatischer Beweis

lemma 2^* is injective $\leq \forall x, y : \mathbb{N}$
 $\text{if}\{2^*(x) = 2^*(y), x = y, \text{true}\}$

automatischer Beweis

lemma $2^*(x + y) = 2^*(x) + 2^*(y) \leq \forall x, y : \mathbb{N}$
 $2^*(x + y) = 2^*(x) + 2^*(y)$

automatischer Beweis

lemma $(2^*(x))^2 = 2^*(2^*(x^2)) \leq \forall x : \mathbb{N}$
 $(2^*(x))^2 = 2^*(2^*(x^2))$

automatischer Beweis

lemma $x = 2^*(\frac{1}{2}(x)) \vee x = {}^+(2^*(\frac{1}{2}(x))) \leq \forall x : \mathbb{N}$
 $\text{if}\{x = 2^*(\frac{1}{2}(x)), \text{true}, x = {}^+(2^*(\frac{1}{2}(x)))\}$

automatischer Beweis

lemma $2*(x) \neq +(2*(y)) \leq \forall x, y : \mathbb{N}$

$\neg 2*(x) = +(2*(y))$

1 Benutzerinteraktion

Zu zeigende Behauptung:

lemma $y \neq 0 \rightarrow 2*((y)^2) \neq (x)^2 \leq \forall x, y : \mathbb{N}$

$\text{if}\{\neg ?0(y), \neg 2*((y)^2) = (x)^2, \text{true}\}$

6 Benutzerinteraktionen

Kalkülbeweis:

- *Keine Beweislücken* (= Makroschritte) !
- *Keine Fehler* (die Korrektheit von *VeriFun* vorausgesetzt)

5 Kalküle als Rechenmodelle

- **Kalkül:** lat. *calcularare* = dt. *rechnen*
- Mit *Kalkülen* können allgemein *Rechenmodelle* (= Vorgehensmodelle zur algorithmischen Lösung eines Problems) definiert werden
- Unterschied zu Algorithmen (als Rechenmodelle):
Kalkül indeterministisch, Algorithmus deterministisch
- Mit Kalkül *Abstraktion* von *Details* einer algorithmischen Lösung
- **Sonderfall Beweiskalküle:** Modell für die Berechnung von Beweisen

Beispiel: Lösung des Matchingproblems

- **Definition:** $\mathcal{T}(\Sigma, \mathcal{V})$ ist die Menge der *Terme* über
 - einer *Signatur* Σ für Funktionssymbole (= Angabe der Menge der verwendeten Funktionssymbole sowie deren Stelligkeit)
 - einer Variablenmenge \mathcal{V}
- *Spezialfall:* Menge der *Grundterme* $\mathcal{T}(\Sigma) := \mathcal{T}(\Sigma, \emptyset)$
- **Definition:** Eine Substitution σ ist eine totale Abbildung $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$ mit (*) “ $\sigma(x) \neq x$ für nur endlich viele $x \in \mathcal{V}$ ”
 - Wegen (*) kann jede Substitution σ durch eine *endliche* Menge $\sigma_{rep} := \{(x, \sigma(x)) \in \mathcal{V} \times \mathcal{T}(\Sigma, \mathcal{V}) \mid \sigma(x) \neq x\}$ repräsentiert werden
 - Schreibweise: $\sigma_{rep} = \{x_1/t_1, \dots, x_n/t_n\}$ falls
 - * $\sigma(x_i) = t_i$ für alle $i \in \{1, \dots, n\}$,
 - * $y \notin \{x_1, \dots, x_n\} \Leftrightarrow \sigma(y) = y$ für alle $y \in \mathcal{V}$
 - **Konvention:**
 - (i) Man schreibt σ anstatt σ_{rep} wenn der Bezug aus dem Kontext offensichtlich ist.
 - (ii) *SUB* bezeichnet die Menge aller σ_{rep} .

- **Definition:** Für eine Substitution σ ist $\hat{\sigma}$ eine totale Abbildung $\hat{\sigma} : \mathcal{T}(\Sigma, \mathcal{V}) \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$ definiert durch
 - $\hat{\sigma}(x) := \sigma(x)$ für alle $x \in \mathcal{V}$
 - $\hat{\sigma}(f(t_1, \dots, t_n)) := f(\hat{\sigma}(t_1), \dots, \hat{\sigma}(t_n))$
- **Konvention:** Man schreibt σ anstatt $\hat{\sigma}$ wenn der Bezug aus dem Kontext offensichtlich ist

Das *Matchingproblem*

- **Gegeben:**
 - Term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ (genannt *Pattern*) sowie
 - Grundterm $q \in \mathcal{T}(\Sigma)$ (genannt *Target*)
- **Gesucht:** Substitution σ mit
 - $\sigma(t) = q$
 - * *Sprechweise:* σ ist ein *Matcher* von t und q
 - und $\theta(t) \neq q$ für alle Substitutionen θ mit $\theta_{rep} \subsetneq \sigma_{rep}$
 - * *Sprechweise:* σ ist ein *minimaler* *Matcher* von t und q
- **Damit:** Ein *Matcher* σ repräsentiert eine *Lösung* für die *Gleichung* $t \doteq q$ in der *freien Termalgebra*

Beispiel 1 (*Matching*)

- (1) Für $t = h(g(x), y)$ und $q = h(g(f(a)), h(a, f(b)))$ (mit $x, y \in \mathcal{V}$ und $f, g, h, a, b \in \Sigma$) ist $\sigma = \{x/f(a), y/h(a, f(b))\}$ ein *minimaler Matcher* von t und q , denn
- $\sigma(h(g(x), y)) = h(g(f(a)), h(a, f(b)))$ (Beweis ?) und
 - $\theta(t) \neq q$ für alle Substitutionen θ mit $\theta_{rep} \subsetneq \sigma_{rep}$ (Beweis ?)
- (2) Für t und q wie in (1) ist $\theta = \{x/f(a), y/h(a, f(b)), z/f(x)\}$ ein *Matcher* von t und q , denn $\theta(h(g(x), y)) = h(g(f(a)), h(a, f(b)))$. Wegen (1) und $\sigma_{rep} \subsetneq \theta_{rep}$ ist θ jedoch *nicht minimal*.
- (3) Für $t = h(g(x), x)$ und $q = h(g(f(a)), h(a, f(b)))$ ist das *Matchingproblem unlösbar*, denn für einen *Matcher* σ müßte
- sowohl $\sigma(x) = f(a)$
 - als auch $\sigma(x) = h(a, f(b))$
- gelten (aber $f(a) \neq h(a, f(b))$ in der freien Termalgebra !).

Jetzt: Lösung des Matchingproblems mittels eines Kalküls ...

Definition 1 (*Matchingkalkül*)

(1) *Sprache*: Mengen $(A, \sigma) \in 2^{\mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)} \times SUB$ mit $|A| < \infty$.⁴

(2) *Regeln*:

(a) *Solve*:
$$\frac{A \uplus \{x \doteq r\}, \sigma}{\theta(A), \theta \cup \sigma}, \text{ falls } x \in \mathcal{V} \text{ und } \theta = \{x/r\}$$
⁵

(b) *Decompose*:
$$\frac{A \uplus \{f(t_1 \dots t_n) \doteq f(q_1 \dots q_n)\}, \sigma}{A \cup \{t_1 \doteq q_1, \dots, t_n \doteq q_n\}, \sigma}$$

(c) *Eliminate*:
$$\frac{A \uplus \{t \doteq t\}, \sigma}{A, \sigma}$$

(3) *Herleitungen*:

- Für $A, A' \subseteq_{fin} \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)$ und $\sigma, \sigma' \in SUB$ ist \Rightarrow eine binäre Relation auf $2^{\mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)} \times SUB$ definiert durch “ $(A, \sigma) \Rightarrow (A', \sigma')$ gdw. (A', σ') aus (A, σ) durch Anwendung einer der Kalkülregeln entsteht”.⁶
- *Damit*: Ein *Matching-Problem* A ist *lösbar* und σ ist ein *Lösung* von A gdw. $(A, \varepsilon) \Rightarrow^* (\emptyset, \sigma)$.⁷

⁴ $2^M =$ Potenzmenge von M .

⁵ $A \uplus \{a\}$ steht für $A \cup \{a\}$ mit $a \notin A$. Zwecks Lesbarkeit wird $t \doteq q$ anstatt (t, q) für Paare aus $\mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)$ geschrieben. $\theta(A) := \{\theta(t) \doteq q \mid t \doteq q \in A\}$

⁶ $M \subseteq_{fin} N$ steht für $M \subseteq N$ und $|M| < \infty$.

⁷ ε bezeichnet die *identische Substitution*, d.h. $\varepsilon_{rep} = \emptyset$.

Bemerkung 2 *Mit der Anwendungsbedingung “falls $f(t_1 \dots t_n) \neq f(q_1 \dots q_n)$ ” für die Decompose-Regel erhält man eine Optimierung, da dann identische Grundterme nicht unnötigerweise zerlegt werden – solche Paare von Grundtermen können in einem Schritt mittels der Eliminate-Regel eliminiert werden.*

Beispiel 2 (Matchingkalkül)

(1) Für die Terme t und q aus Beispiel 1(1) erhält man

$$\begin{aligned} & (\overline{\{h(g(x), y) \doteq h(g(f(a)), h(a, f(b)))\}}, \emptyset) \\ \Rightarrow & (\overline{\{g(x) \doteq g(f(a)), y \doteq h(a, f(b))\}}, \emptyset) && \text{, mit Regel (b) Decompose} \\ \Rightarrow & (\overline{\{g(x) \doteq g(f(a))\}}, \{y/h(a, f(b))\}) && \text{, mit Regel (a) Solve} \\ \Rightarrow & (\overline{\{x \doteq f(a)\}}, \{y/h(a, f(b))\}) && \text{, mit Regel (b) Decompose} \\ \Rightarrow & (\emptyset, \{x/f(a), y/h(a, f(b))\}) && \text{, mit Regel (a) Solve} \end{aligned}$$

(2) Für die Terme t und q aus Beispiel 1(3) erhält man

$$\begin{aligned} & (\overline{\{h(g(x), x) \doteq h(g(f(a)), h(a, f(b)))\}}, \emptyset) \\ \Rightarrow & (\overline{\{g(x) \doteq g(f(a)), x \doteq h(a, f(b))\}}, \emptyset) && \text{, mit Regel (b) Decompose} \\ \Rightarrow & (\overline{\{g(h(a, f(b))) \doteq g(f(a))\}}, \{x/h(a, f(b))\}) && \text{, mit Regel (a) Solve} \\ \Rightarrow & (\overline{\{h(a, f(b)) \doteq f(a)\}}, \{x/h(a, f(b))\}) && \text{, mit Regel (b) Decompose} \\ \nRightarrow & && \text{, keine Regel anwendbar} \end{aligned}$$

Für den Matchingkalkül gilt:

Satz 2 (*Eigenschaften des Matchingkalküls*)

- (1) (*Korrektheit*) Sei $\sigma \in SUB$ eine Lösung von $A \subseteq_{fin} \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)$. Dann gilt $\sigma(t) = q$ für alle $t \doteq q \in A$.
- (2) (*Vollständigkeit*) Sei $A \subseteq_{fin} \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)$ und sei $\theta \in SUB$ mit $\theta(t) = q$ für alle $t \doteq q \in A$. Dann gibt es eine Lösung von A .
- (3) (*Minimalität*) Sei $A \subseteq_{fin} \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma)$, sei $\theta \in SUB$ mit $\theta(t) = q$ für alle $t \doteq q \in A$ und sei $\sigma \in SUB$ eine Lösung von A . Dann gilt $\sigma \subseteq \theta$.

Weitere Beispiele für Kalküle als Rechenmodell:

- Berechnungskalkül für die Semantik von \mathcal{L} -Programmen (\Rightarrow **Kapitel 5**)
- Kalkül zur Überprüfung der Typkorrektheit von Termen
- λ -Kalkül (\Rightarrow Berechenbarkeitstheorie)