Automatisierte Logik und Programmierung

Teil I

Formalisierung von
Logik, Berechnung und Datentypen

Automatisierte Logik und Programmierung

Einheit 2

Formale Logik

1. Evidenzbasierte Beweisführung
2. Formalisierung in Aussagenlogik
3. Evidenz für logische Aussagen
4. Evidenzkonstruktion mit Refinement Logik
5. Formale Prädikatenlogik

Logik präzisiert Schlussfolgerungen

- **Wissenschaft und Programmierung benötigt Vertrauen**
  - Warum sollten wir eine aufgestellte Behauptung akzeptieren?
    - gilt ein mathematisches Theorem wirklich?
    - ist es sicher, daß ein Programm wie vorgesehen funktioniert?
  - Welche Belege können wir für eine Behauptung geben?
    - was ist ein akzeptabler Beweis?
    - wie können wir Software verifizieren?
- **Logik trennt gültige Schlüsse von ungültigen**

  Tatsache: *Alle Säugetiere sind behaart*

  Tatsache: *Alle ~~Affen~~ Teddybären sind behaart*

  Konsequenz: *Also sind alle ~~Affen~~ Teddybären Säugetiere* falsch!

  Ursprüngliche Konsequenz ist zufällig richtig aber nicht allgemeingültig
- **Formalisierung offenbart Struktur logischer Argumente**
  - Begriffe werden durch symbolische Platzhalter ersetzt
  - Abstrakte Symbole beschreiben Zusammenhänge zwischen Aussagen

$$((M \Rightarrow H) \wedge (A \Rightarrow H)) \Rightarrow (A \Rightarrow M)$$ nicht allgemeingültig

Evidenz

Beleg für die Gültigkeit einer Aussage

- **Das Wort ist eigentlich eine Fehlübersetzung**
  - *evidence* (engl.) $\equiv$ Beweis, Beleg, (juristisch) Indiz, Zeugenaussage
    (Mit dieser Bedeutung hat es sich inzwischen festgesetzt)
  - Korrekt: "das dem Augenschein nach unbezweifelbar Erkennbare"
- **Evidenz für Alltagsargumenten**
  - Die Polizei behauptet, Sie wären zu schnell gefahren
    Was wäre ein Beweis für diese Behauptung? Radarmessung, Photo
  - Nach einem Unfall behauptet Ihr Gegner, Sie hätten Ihr Handy benutzt
    Welche Belege könnte es dafür geben? Daten des Providers, Uhrzeit
  - Vor einer Weile soll es geregnet haben - welches Indiz zeigt dies?
    Die Tatsache, daß die Straße naß ist, ist ein Indiz, aber kein Beweis
  - Wie können Sie beweisen, daß die Anzahl der Personen in diesem Raum entweder gerade oder ungerade ist?
    Das Argument "nichts anderes ist möglich" ist unbefriedigend – zählen ist besser

Evidenz für mathematische Aussagen

- **Beleg für** 2<3? Einfaches anschauliches Argument
  - Methodik nur für einfache Beispiele geeignet. Mathematik muß Wege beschreiben, alles auf wenige selbsterklärende Konzepte zu reduzieren
- **Evidenz für** 0=0**?** selbsterklärend
  - Fundamentale Aussage, nicht durch anderes zu beweisen (Axiom)
- **... für** 1=1, 2=2, 3=3, **... ?** auch selbsterklärend?
  - Im Prinzip könnte man dem zustimmen, aber was ist mit
    1111111111111111111111111111111 = 1111111111111111111111111111111?
  - Besser: Reduktion von Zahlenvergleichen auf Vorgänger und 0=0
- **... für** 0≠1**?** selbsterklärend?
  - Es gibt keinen Beweis für 0≠1 oder 0=1
  - 0=1 wäre absurd (Arithmetik würde kollabieren). Man muß 0≠1 fordern
  - 0≠2, 0≠3,... ist nicht reduzierbar, daher Postulat 0≠x+1 (0≠s(x))
- **... für "je zwei Zahlen sind gleich oder verschieden"?** selbsterklärend???
  - Zu sagen "nichts anderes ist möglich", wäre sehr unbefriedigend
  - Besserer Beleg: Reduktionsalgorithmus, der Vergleich durchführt

Formalisierung in Aussagenlogik

- **Syntax präzisiert natürlichsprachige Konzepte**
  - Sprachdefinition analog zur Beschreibung Programmiersprachen
    - Frei wählbare Symbole als Platzhalter für Aussagen
    - Reservierte Schlüsselwörter/Symbole für logische Bezüge
  - Komplexe Formeln werden induktiv aus einfacheren zusammengesetzt
  - Eindeutige Syntaxdefinition unterstützt spätere Implementierung
- **Erlaubte Symbole**
  - Aussagenvariablen $P, Q, R, P_o, Q_0, R_0,, \ldots$ (Großbuchstaben mit Index)
  - Logische Symbole $\neg$, $\wedge$, $\vee$, $\Rightarrow$ und Klammern
- **Syntax der Aussagenlogik**
  - Jede Aussagenvariable ist eine Formel
  - Sind A und B Formeln, dann auch $\neg A$, $(A \Rightarrow B)$, $(A \wedge B)$, $(A \vee B)$
  - Man kann Konventionen einführen, um Klammern zu sparen (später)

Die Symbole A und B sind Platzhalter für Formeln, d.h. Objekte der Meta-Sprache

Formalisierung umgangssprachlicher Aussagen

- *Wenn es friert, fährt die S-Bahn nicht. Die S-Bahn fährt. Also friert es nicht*
  - Mit memnonischen Abkürzungen $(((Friert \Rightarrow \neg SBahn) \wedge SBahn) \Rightarrow \neg Friert)$
  - Aussschließlich mit erlaubten Symbolen $\boldsymbol{(((P \Rightarrow \neg Q) \wedge Q) \Rightarrow \neg P)}$
- *Wenn es friert, fährt die S-Bahn nicht. Es friert es nicht. Also fährt die S-Bahn*
  - $\boldsymbol{(((P \Rightarrow \neg Q) \wedge \neg P) \Rightarrow Q)}$ (leider nicht wahr)
- *Wenn es nicht keinen Frost gibt, dann friert es*
  - $\boldsymbol{\neg\neg P \Rightarrow P}$ (Achtung, regional andere Lesweise “nicht kein”= “absolut nicht”)
- *Sein oder nicht sein*
  - $\boldsymbol{P \vee \neg P}$ (der ganze Rest dieses inhaltsschweren Satzes geht verloren)
- *Dieser Satz ist wahr*
  - Nicht formulierbar in Aussagen- oder Prädikatenlogik

Semantik der Aussagenlogik

- **Logik ist mehr als nur eine formale Kurzschreibweise**
  - Formeln haben eine intendierte Bedeutung (Semantik)
  - Man kann feststellen, ob eine Aussage/Schlußfolgerung gültig ist
  - Hierzu muß man die Bedeutung von Formeln präzise festlegen
- **Wahrheitstabellen sind kein sinnvoller Weg**
  - Mischen Mathematik mit metaphysischem Konzept der Wahrheit
  - Nehmen an, daß philosphischen Frage '*Was ist Wahrheit?*' geklärt ist
- **Interpretation in Zielsprache ist unintuitiv**
  - Verlagern das Problem nur auf eine andere Sprache
  - Was ist die Bedeutung dieser Zielsprache?
- **Evidenzbasierte Semantik**
  - Beschreibt die Bedeutung logischer Formeln durch Angabe von Belegen (d.h. Rechtfertigungen) für ihre Gültigkeit
  - Umgeht Notwendigkeit philosphische Fragestellungen zu klären

Evidenzsemantik der Aussagenlogik

- **Konstruktion von Rechtfertigungen für Formeln**
  - Rechtfertigungen werden als formale Evidenz(terme) ausgedrückt
  - Induktive Konstruktion von Evidenz folgt syntaktischen Aufbau

  Achtung: Wie bei Programmiersprachen dürfen ausschließlich bereits erklärte Konstrukte verwendet werden. Gesetze der Boolesche Algebra können erst angewandt werden, wenn hierfür Evidenz vorliegt

- **Formale Schreibweisen**
  - $[\boldsymbol{A}]$: Menge (oder Typ) der Evidenzen für die Formel A
  - $\boldsymbol{a} \in [\boldsymbol{A}]$: Term a ist Evidenz für die Formel A (mathematische Notation)

    $\boldsymbol{a} : [\boldsymbol{A}]$: a ist Evidenz für die Formel A (Programmiersprachennotation)

- **Evidenz für Aussagenvariablen** $[A]$ **bleibt unspezifiziert**
  - Aussagenvariablen sind Platzhalter für unbekannte Aussagen
  - Keine feste Evidenz möglich

Evidenz für Implikationen

- **$A \Rightarrow B$: "Aus A folgt B"** $[A \Rightarrow B] = [A] \rightarrow [B]$
  - Aus Evidenz a für A muß eine Evidenz b für B konstruiert werden
  - Evidenz für $A \Rightarrow B$ muß Funktion f sein mit $f(a) : [B]$, falls $a : [A]$
  - Typ der Evidenzen für $A \Rightarrow B$ ist Typ der Funktionen von $[A]$ nach $[B]$
- **Konkrete Evidenz für $P \Rightarrow P$**
  - Evidenz ist Funktion $f : [P] \rightarrow [P]$ mit $f(p) : [P]$, falls $p : [P]$
  - Einfachste Funktion dieser Art ist Identität, d.h. $f(p) = p$
  - Einfachste Notation ist Schreibweise des λ-Kalküls: $f = \boldsymbol{\lambda p.\, p}$
- **Konkrete Evidenz für $P \Rightarrow (Q \Rightarrow P)$**
  - Evidenz ist $f : [P] \rightarrow ([Q] \rightarrow [P])$ mit $f(p) = g_p$ für eine Funktion $g_p : [Q] \rightarrow [P]$, falls $p : [P]$. Dabei ist $q_p(q) : [P]$, falls $q : [Q]$
  - Einfachste Lösung ist $g_p = \lambda q.\, p$, also $f = \boldsymbol{\lambda p.\, (\lambda q.\, p)}$
- **Konkrete Evidenz für $P \Rightarrow ((P \Rightarrow Q) \Rightarrow Q)$**
  - Evidenz ist Funktion f mit $f(p) = g_p$ für ein $g_p : ([P] \rightarrow [Q]) \rightarrow [Q]$, falls $p : [P]$. Dabei ist $q_p(h) : [Q]$, falls $h : [P] \rightarrow [Q]$
  - Einfachste Lösung ist $g_p(h) = h(p)$, also $f = \boldsymbol{\lambda p.\, (\lambda h.\, h(p))}$

Evidenz für Konjunktionen

- **$A \wedge B$: "A und B"** $\qquad [A \wedge B] = [A] \times [B]$
  - Um Evidenz für $A \wedge B$ zu konstruieren, braucht man Evidenzen a für A und b für B, also ein Paar (a, b) mit $a : [A]$ und $b : [B]$
  - Typ der Evidenzen für $A \wedge B$ ist das Produkt von $[A]$ und $[B]$
- **Konkrete Evidenz für $P \Rightarrow (Q \Rightarrow (P \wedge Q))$**
  - Evidenz ist Funktion f mit $f(p) = g_p$ für ein $g_p : [Q] \rightarrow ([P] \times [Q])$, falls $p : [P]$. Dabei ist $q_p(q) : [P] \times [Q]$, falls $q : [Q]$
  - Einfachste Lösung ist $g_p(q) = (p, q)$, also $\qquad f = \lambda p.\ (\lambda q.\ (p, q))$
- **Konkrete Evidenz für $P \Rightarrow (Q \Rightarrow (Q \wedge P))$**
  - Dasselbe Argument liefert $\qquad f = \lambda p.\ (\lambda q.\ (q, p))$
- **Konkrete Evidenz für $(P \wedge Q) \Rightarrow P$**
  - Evidenz ist Funktion f mit $f(x) : [P]$ für ein $x : [P] \times [Q]$,
  - x muß ein Paar sein, dessen erste Komponente $x.1$ in $[P]$ liegt und dessen zweite Komponente $x.2$ zu $[Q]$ gehört
  - Der gesuchte Evidenzterm ist damit $\qquad f = \lambda x.\ x.1$

Evidenz für Disjunktionen

- **$A \vee B$: "A oder B"** $[A \vee B] = [A] + [B]$
  - Um Evidenz für $A \vee B$ zu konstruieren, braucht man Evidenz $a : [A]$ oder Evidenz $b : [B]$ und eine Kennzeichnung, welches von beiden man hat
  - Mögliche Evidenzen sind $\texttt{inl}(a)$ mit $a : [A]$ und $\texttt{inr}(b)$ mit $b : [B]$
  - Typ der Evidenzen für $A \vee B$ ist die direkte Summe von $[A]$ und $[B]$
- **Konkrete Evidenz für $P \Rightarrow (P \vee Q)$**
  - Evidenz ist Funktion f mit $f(p) : [[P] + [Q]$
  - Einfachste Lösung ist $f = \lambda p.\, \texttt{inl}(p)$
- **Konkrete Evidenz für $(P \vee Q) \Rightarrow (Q \vee P)$**
  - Evidenz ist Funktion f mit $f(x) : [Q] + [P]$ für ein $x : [P] + [Q]$
  - x is entweder $\texttt{inl}(p)$ mit $p : [P]$ oder $\texttt{inr}(q)$ mit $q : [Q]$
    Im ersten Fall ist $f(x) = \texttt{inr}(p)$, im zweiten $f(x) = \texttt{inl}(q)$
  - Als geschlossener Term $f = \lambda x.\, (\texttt{case}\ x\ \texttt{of}\ \texttt{inl}(p) \rightarrow \texttt{inr}(p) \mid \texttt{inr}(q) \rightarrow \texttt{inl}(q))$
    (Pattern-Matching Notation)

Evidenz für Negationen

- **$\neg A$: "nicht A"** $[\neg A] = [A] \to \{\}$
  - Evidenz für $\neg A$ ist Nachweis(!), daß keine Evidenz für A möglich ist
  - Aus Annahme $a : [A]$ muß Widerspruch konstruiert werden
  - Aussagenlogik hat keine fundamentalen Widerspruche (wie z.B. `0=1`)
    Ergänze konstante Formel f und das Postulat $[\mathsf{f}] = \{\}$ ($\hat{=}$ der leere Typ)
  - Typ der Evidenzen für $\neg A$ ist Typ der Funktionen von $[A]$ nach $\{\}$
- **Konkrete Evidenz für $P \Rightarrow \neg\neg P$**
  - Evidenz ist Funktion f mit $f(p) = g_p$ für ein $g_p : ([P] \to \{\}) \to \{\}$,
    falls $p : [P]$. Dabei ist $q_p(h) : \{\}$, falls $h : [P] \to \{\}$
    Einfachste Lösung ist $g_p(h) = h(p)$, also $f = \lambda p.\, (\lambda h.\, h(p))$
    Dies bedeutet, daß es keine Funktion $h : [P] \to \{\}$ geben kann, also $\neg P$
    keine Evidenz hat, da man sonst ein Element des leeren Typs konstruieren könnte
- **Konkrete Evidenz für $\neg(P \vee Q) \Rightarrow \neg P$**
  - Evidenz ist Funktion f mit $f(h) = g_h : [P] \to \{\}$ falls $h : ([P] + [Q]) \to \{\}$.
    Dabei ist $q_h(p) : \{\}$, falls $p : [P]$
  - Lösung ist $g_h(p) = h(\mathtt{inl}(p))$, also $f = \lambda h.\, (\lambda p.\, h(\mathtt{inl}(p)))$

Evidenz für komplexere Formeln

- $((P \vee Q) \wedge ((P \Rightarrow R) \wedge (Q \Rightarrow R))) \Rightarrow R$
  - Intuitiv einfach: wenn P gilt, beweise R mit $P \Rightarrow R$, sonst mit $Q \Rightarrow R$
  - Evidenzkonstruktion ist wie Programmierung mit Typbedingungen:
    Finde einen Term in $(([P]+[Q])\times(([P]\to[R])\times([Q]\to[R]))) \to [R]$
  - Evidenz ist Funktion f mit $f(x) = r : [R]$ für eine Eingabe
    x aus $([P]+[Q])\times(([P]\to[R])\times([Q]\to[R]))$
  - x ist ein Tripel $(z,(g,h))$ mit $z : [P]+[Q]$, $g : [P]\to[R]$, $h : [Q]\to[R]$
  - z ist entweder $\texttt{inl}(p)$ mit $p : [P]$ oder $\texttt{inr}(q)$ mit $q : [Q]$
    Im ersten Fall ist $r = g(p)$, im zweiten $r = h(q)$
  - Lösung wäre $\lambda(z,(g,h)).\ \texttt{case}\, z\, \texttt{of}\ \texttt{inl}(p) \to g(p) \mid \texttt{inr}(q) \to h(q)$
  - Dies ist kein korrekter Term (kein Pattern Matching bei λ erlaubt)
    Verwende $z = x.1,\ g = x.2.1,\ h = x.2.2$
  - Evidenz ist also $\lambda x.(\texttt{case}\, x_1\, \texttt{of}\ \texttt{inl}(p) \to x_{21}(p) \mid \texttt{inr}(q) \to x_{22}(q)$
- $(P \vee Q) \Rightarrow ((P \Rightarrow R) \Rightarrow ((Q \Rightarrow R) \Rightarrow R))$
  - Evidenz ist $\lambda z.\lambda g.\lambda h.\ \texttt{case}\, z\, \texttt{of}\ \texttt{inl}(p) \to g(p) \mid \texttt{inr}(q) \to h(q)$
  - Argumente kommen der Reihe nach statt im Block (currying)
    Alternativ $\lambda z.\ \texttt{case}\ z\ \texttt{of}\ \texttt{inl}(p) \to (\lambda g.\lambda h.\, g(p)) \mid \texttt{inr}(q) \to (\lambda g.\lambda h.\, h(q))$

Ein paar überraschende Ergebnisse

- **Evidenz für $P \vee \neg P$**
  - Evidenz muß $\texttt{inl}(p)$ für ein $p : [P]$ oder $\texttt{inr}(\lambda p.\, x_p)$ für ein $x_p : \{\}$ sein.
  - Solange man nichts über $[P]$ weiß, kann man beides nicht konstruieren
  - **Keine universelle Evidenz**
- **Evidenz für $\neg\neg P \Rightarrow P$**
  - Evidenz ist Funktion f mit $f(h) = p : [P]$, falls $h : ([P] \to \{\}) \to \{\}$
  - Es gibt keinen allgemeinen Weg, p aus h zu konstruieren
  - **Keine universelle Evidenz**
- **Evidenz für $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$**
  - Evidenz ist Funktion f mit $f(h) = x : ([P] \to \{\}) + [Q]$ falls $h : [P] \to [Q]$.
  - x muß $\texttt{inl}(\lambda p.\, x_p)$ für ein $x_p : \{\}$ oder $\texttt{inr}(q)$ für ein $q : [Q]$ sein
  - Solange man nichts über $[P]$ weiß, kann man beides nicht konstruieren
  - **Keine universelle Evidenz**

Was ist das denn für eine Logik?

- **Bekannte logische Gesetze gelten nicht?**
  - Gesetz vom ausgeschlossenen Dritten $P \vee \neg P$
  - Gesetz der Doppelten Negation $\neg\neg P \Rightarrow P$
  - Zusammenhang Implikation und Disjunktion $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$

  ... und viele andere mehr
- **Was bedeutet $P \vee \neg P$ denn genau?**
  - "*Jede beliebige Aussage ist ~~wahr oder nicht~~*"? Das steht da nicht!

    "*Jede beliebige Aussage ist gültig oder ihre Negation ist gültig*"
  - Begriff der "Wahrheit" verwendet metaphysische Ideen und Einsichten

    Mit solchen Konzepten kann man in der Informatik nicht arbeiten
  - $P \vee \neg P$ würde bedeuten, daß man jede (!) Aussage entscheiden kann

    Aus der theoretischen Informatik wissen wir, daß dies nicht stimmt

    Heißt das, daß die Formel $\neg(P \vee \neg P)$ gültig ist?
- **Logik heißt "Intuitionistische (konstruktive) Logik"**
  - Vermeidet Wahrheitsbegriff der "klassischen" Logik
  - Akzeptiert nur mathematische Aussagen, die belegbar sind
  - Ist für das Schließen über Programme besser geeignet

Evidenzsemantik – Zusammenfassung

Aussage A	Evidenztyp $[A]$	Evidenzkonstruktion	Dekompositionsterm
$A \Rightarrow B$	$[A] \to [B]$	$\lambda a.b$	$f(a)$
$A \wedge B$	$[A] \times [B]$	(a, b)	$x.1$, $x.2$
$A \vee B$	$[A] + [B]$	$\texttt{inl}(a)$, $\texttt{inr}(b)$	$\texttt{case}\ x\ \texttt{of}\ \texttt{inl}(a) \to s \mid \texttt{inr}(b) \to t$
$\neg A$	$[A] \to \{\}$	$\lambda a.b$	$f(a)$
f	$\{\}$	–	–

- **Aussagen korrespondieren mit Datentyp der Evidenzen**
  - Es gibt Terme, um Evidenz zu konstruieren
  - Es gibt Terme, um komplexe Evidenz zu zerlegen
- **Evidenzterme bilden eine Art Programmiersprache**
  - Logische Formeln sind Spezifikationen dieser Programme
  - Ausdruckstarke Logiken ermöglichen sehr genaue Spezifikationen
- **Man kann mit Evidenztermen rechnen**
  - Konstruktions- und Dekompositionsterme sind invers zueinander
  - $(a, b).1 \hat{=} a$, $(\lambda a.b)(c) \hat{=} b[c/a]$, ...

...Mehr dazu in der Einheit zum λ-Kalkül