Logik I - Zusamenfassung

Patrick Pletscher

19. Oktober 2003

1 Aussagenlogik

1.1 Sprachkonstrukte

$\vee$ Das logische OR (Disjunktion)

$\wedge$ Das logische AND (Konjunktion)

$\neg$ Das logische NOT

$\top$ Das logische TRUE

$\bot$ Das logische FALSE

$\rightarrow$ Die logische Implikation

$\leftrightarrow$ Die logische Äquivalenz

[bindet stark] $\neg \wedge \vee \rightarrow \leftrightarrow$ [bindet schwach]
Die Konjunktion ($\wedge$) und die Disjunktion ($\vee$) sind linksgeklammert, die Implikation ($\rightarrow$) ist rechtsgeklammert.

$A \rightarrow B$:

- A hinreichend für B
- B notwendig für A
- Wenn A dann B
- A nur wenn B
- $\neg B$ nicht, solange A
- B solange A
- nicht A ausser B

$A \leftrightarrow B$
A genau dann wenn B

$\neg A \wedge \neg B$
weder A noch B

1.2 Formale Beweise

$$\frac{A\ B}{A \wedge B}(\wedge i) \qquad \frac{A \wedge B}{A\ B}(\wedge e)$$

$$\frac{A}{A \vee B}(\vee i) \qquad \frac{B}{A \vee B}(\vee i)$$

$$\frac{A \vee B\ A \rightarrow C\ B \rightarrow C}{C}(\vee e)$$

$$\frac{A \rightarrow \bot}{\neg A}(\neg i) \qquad \frac{A\ \neg A}{B}(\neg e)$$

$$\frac{A \rightarrow B\ B \rightarrow A}{A \leftrightarrow B}(\leftrightarrow i)$$

$$\frac{A \leftrightarrow B}{A \rightarrow B}(\leftrightarrow e) \qquad \frac{A \leftrightarrow B}{B \rightarrow A}(\leftrightarrow e)$$

$$\frac{A\ A \rightarrow B}{B}(MP) \qquad \frac{}{\top}(\top i)$$

$$\frac{\bot}{A}(EFQ) \qquad \frac{}{A \vee \neg A}(TND)$$

1.3 Ausgewählte Gesetze

$$\begin{aligned}
(A \wedge (B \vee C)) &\leftrightarrow ((A \wedge B) \vee (A \wedge C)) \\
(A \vee (B \wedge C)) &\leftrightarrow ((A \vee B) \wedge (A \vee C)) \\
(A \rightarrow (B \vee C)) &\leftrightarrow ((A \rightarrow B) \vee (A \rightarrow C)) \\
\neg(A \vee B) &\leftrightarrow (\neg A \wedge \neg B) \\
\neg(A \wedge B) &\leftrightarrow (\neg A \vee \neg B) \\
(A \rightarrow B) &\leftrightarrow (\neg A \vee B)
\end{aligned}$$

1.4 Boolsche Funktionen

IO-Tabelle $\Rightarrow$ logische Funktion, zwei Wege

1. a) Betrachte nur die Zeilen, bei denen der Ausgabewert 1 ist.
   b) Bilde eine Konjunktion nach dem Muster: $1 \Rightarrow p_i$, $0 \Rightarrow \neg p_i$.

c) Bilde daraus eine Disjunktion (DNF)

2. a) Betrachte nur die Zeilen, bei denen der Ausgabewert 0 ist.
   b) Bilde eine Disjunktion nach dem Muster: $1 \Rightarrow \neg p_i$, $0 \Rightarrow p_i$.
   c) Bilde daraus eine Konjunktion (KNF)

Funktionale Vollständigkeit

Es gibt 2^4 mögliche Funktionen mit zwei Argumenten. Sie alle können durch die ausschliessliche Benutzung von $\neg, \wedge$, $\neg, \rightarrow$ oder auch nur durch das NAND ($p\ NAND\ q = \neg(p \wedge q)$) dargestellt werden, sie sind "funktional Vollständig"

1. $\neg, \wedge$
   a) NAND: $\neg(p \wedge q)$
   b) XOR: $\neg(p \wedge q) \wedge \neg(\neg p \wedge \neg q)$
   c) $p \leftrightarrow q$: $\neg(p \wedge \neg q) \wedge \neg(q \wedge \neg p)$
2. $\rightarrow, \neg$
   a) $p \vee q$: $\neg p \rightarrow q$
   b) $p \wedge q$: $\neg(p \rightarrow \neg q)$
   c) $p \leftrightarrow q$: $\neg((p \rightarrow q) \rightarrow \neg(q \rightarrow p))$

1.5 Normalformen

Negationsnormalform (NNF)

1. Elimination der Aequivalenz
   $A \leftrightarrow B$ wird überall ersetzt durch $(A \rightarrow B) \wedge (B \rightarrow A)$
2. Elimination der Implikation
   $A \rightarrow B$ wird überall ersetzt durch $\neg A \vee B$
3. Hineinziehen der Negation
   $\neg\neg A$ wird ersetzt durch A
   $\neg(A \wedge B)$ wird ersetzt durch $\neg A \vee \neg B$ [DeMorgan]
   $\neg(A \vee B)$ wird ersetzt durch $\neg A \wedge \neg B$ [DeMorgan]

konjunkte Normalform (KNF)

Eine Konjunktion von Disjunktionen von Literalen. Alternativ auch mit IO-Tabelle.

1. $A \vee (B \wedge C)$ wird ersetzt durch $(A \vee B) \wedge (A \vee C)$
2. $(B \wedge C) \vee A$ wird ersetzt durch $(B \vee A) \wedge (C \vee A)$

disjunkte Normalform (DNF)

Eine Disjunktion von Konjunktionen von Literalen. Alternativ auch mit IO-Tabelle.

1. $A \wedge (B \vee C)$ wird ersetzt durch $(A \wedge B) \vee (A \wedge C)$
2. $(B \vee C) \wedge A$ wird ersetzt durch $(B \wedge A) \vee (C \wedge A)$

1.6 Resolution

$S = \{\{p, r\}, \{q, \neg r\}\}$ entspricht $(p \vee r) \wedge (q \vee \neg r)$
Ein Beweisschema mit nur einer Regel, um Formeln aus Literalen (p oder $\neg p$) und Klauseln (Menge von Literalen) zu beweisen. Eine Klausel $C = \{L_1, \ldots, L_n\}$ entspricht der Disjunktion $L_1 \vee \ldots \vee L_n$. Ihr Wahrheitswert ist:

$$[[C]]_a := \begin{cases} 1; \exists L \in C, [[L]]_a = 1 \\ 0; else \end{cases}$$

Die leere Klausel ist immer Falsch, so fällt das Beweisen durch zurückführen des Gegenteils auf die leere Klausel leicht. Eine Klauselmenge entspricht einer Formel in der KNF. Der Wahrheitswert ist:

$$[[S]]_a := \begin{cases} 1; \forall C \in S, [[C]]_a = 1 \\ 0; else \end{cases}$$

$$\frac{C\ D\ (L \in C)\ (\overline{L} \in D)}{(C \backslash \{L\}) \cup (D \backslash \{\overline{L}\})}$$

1. Wähle $L \in C$, $\overline{L} \in D$
2. Streiche L in C und $\overline{L}$ in D.
3. Vereinigt die verbleibenden Literale.

Solange die Klauselmenge S nicht leer ist, mache die folgenden Schritte:

1. Wähle eine Klausel C aus S und bewege sie von S nach U.
2. Bestimme alle Resolventen von C und Klauseln in U und füge die Resolvente zu S hinzu.
3. Falls eine Resolvente eine Tautologie ist, streiche sie.
4. Falls eine Resolvente von einer Klausel in S oder U subsumiert wird, streiche die Resolvente.

5. Falls eine Klausel in S oder U von einer Resolvente subsumiert wird, streiche die Klausel.

Falls man die leere Klausel erhält ist S unerfüllbar.

Beispiel

$p \wedge \neg q \wedge \neg r$ logische Konsequenz von Klauselmenge $S = \{\{\neg p, \neg q\}, \{q, \neg r\}, \{p, r\}, \{p, \neg q, \neg r\}\}$ ist.
Also nimmt man das Gegenteil nämlich $\neg(p \wedge \neg q \wedge \neg r) = \neg p \vee q \vee r$ an.

1:	$\{\neg p, \neg q\}$	
2:	$\{q, \neg r\}$	
3:	$\{p, r\}$	
4:	$\{p, \neg q, \neg r\}$	
5:	$\{\neg p, q, r\}$	
6:	$\{\neg p, r\}$	Res von 1,5
7:	$\{r\}$	Res von 3,6
8:	$\{\neg p, \neg r\}$	Res von 1,4
9:	$\{\neg r\}$	Res von 2,8
10:	$\{\}$	Res von 7,9

Somit ist das Gegenteil unerfüllbar und somit $p \wedge \neg q \wedge \neg r$ eine logische Konsequenz von S.

1.7 Davis-Putnam Prozedur

Dient dem effizienten Testen von Erfüllbarkeit von Klauselmengen, man findet alle Belegungen von S, die sie wahr machen. Der Algoritmus ist:

1. Wenn S leer, gib die erfüllende Belegung aus. (BackTracking)
2. Sonst wähle eine Aussagenvariable $p \in S$.
3. Ergänze die Belegung durch $p := 1$, entferne alle Klauseln von S mit p, streiche in den anderen alle $\neg p$.
4. Gehe rekursiv zu 1.
5. Mache die Änderungen von 3. rückgängig und setze $p := 0$, entferne alle Klauseln von S mit $\neg p$, streiche in den anderen alle p.
6. Gehe rekursiv zu 1.

Für den manuellen Gebrauch wird oft eine Baumdarstellung gewählt, wobei der Baum beim Auftreten von leeren Klauseln früh abgeschnitten werden können. So lässt sich das Vorgehen stark optimieren.
leere Klausel $\{\square\}$: keine erfüllende Belegung
leere Klauselmenge $\{\}$: erfüllende Belegung

1.8 Boolsche Algebra

$* = \wedge$, $+ = \vee$, $' = /$
Boolsche Algebren sind algebraische Strukturen $\langle A, *, +, ', 0, 1\rangle$, wobei A eine Menge (Universum), 0 und 1 Elemente, *, + zweistellige und ' ein einstelliger Operatoren in A sind. Es gelten folgende Gesetze:

Kommutativität $x * y = y * x$, $x + y = y + x$

Assoziativität $x*(y*z) = (x*y)*z$, $x+(y+z) = (x + y) + z$

Verschmelzung $x * (x + y) = x$, $x + (x * y) = x$

Distributivität $x * (y + z) = (x * y) + (x * z)$, $x + (y * z) = (x + y) * (x + z)$

Komplement $x * x' = 0$, $x + x' = 1$

Mit Boolschen Algebren lässt sich ähnlich rechnen wie mit den üblichen logischen Formeln. Alle logischen Konsequenzen lassen sich aus den Axiomen ableiten.

0-1 Algebra $\langle\{0, 1\}, \wedge, \vee, \neg, 0, 1\rangle$

Potenzmengen $\langle\mathcal{P}(M), \cap, \cup, -, [], M\rangle$

Jede endliche Boolsche Algebra ist isomorph zu einer Potenzmenge $\rightarrow$ nur 2^x Elemente möglich.

2 Prädikatenlogik

2.1 Sprachkonstrukte

$\forall x A$ Für alle x gilt A

$\exists x A$ Es gibt ein x, so dass A gilt

$A \approx B$ A ist das selbe wie B

$f(x)$ Funktion: Operieren im Universum

$G(x)$ Prädikat: Testen eine Bedingung

$\forall x, y(x * y) \approx 1 \rightarrow y * x \approx 1)$

$\forall x(1 * x \approx x)$

$\forall x, y, z(x * y \approx 1 \wedge x * z \approx 1 \rightarrow y \approx z)$

2.2 Quantorenregeln

Anzahlaussagen
es gibt mindestens ein x mit P $\leftrightarrow \exists x P(x)$
es gibt höchstens ein x mit P $\leftrightarrow \forall x \forall y (P(x) \wedge P(y) \rightarrow x \approx y)$
es gibt genau ein x mit P $\leftrightarrow \exists x (P(x) \wedge \forall y (P(y) \rightarrow x \approx y))$

Negation von Quantoren
$\neg \forall x P(x) \leftrightarrow \exists x \neg P(x)$
$\neg \exists x P(x) \leftrightarrow \forall x \neg P(x)$

Negation von beschränkten Quantoren
$\neg \forall x (P(x) \rightarrow Q(x)) \leftrightarrow \exists x (P(x) \wedge \neg Q(x))$
$\neg \exists x (P(x) \wedge Q(x)) \leftrightarrow \forall x (P(x) \rightarrow \neg Q(x))$

Vertauschung von Quantoren der gleichen Art
$\forall x \forall y R(x, y) \leftrightarrow \forall y \forall x R(x, y)$
$\exists x \exists y R(x, y) \leftrightarrow \exists y \exists x R(x, y$

Hineinziehen von Quantoren
$\forall x (P(x) \wedge Q(x)) \leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
$\exists x (P(x) \vee Q(x)) \leftrightarrow \exists x P(x) \vee \exists x Q(x)$

Nullquantifikation
x kommt nicht als freie Variable in A vor.
$\forall x A \leftrightarrow A$
$\exists x A \leftrightarrow A$
$\forall x (A \vee P(x)) \leftrightarrow A \vee \forall x P(x)$
$\exists x (A \wedge P(x)) \leftrightarrow A \wedge \exists x P(x)$

2.3 Formale Beweise

$$\frac{\forall x\ A}{A\frac{t}{x}}(\forall e) \qquad \frac{A}{\forall x\ A}(\forall i)$$

$$\frac{A\frac{t}{x}}{\exists x\ A}(\exists i)$$

$$\frac{A}{t \approx t}(ID) \qquad \frac{s \approx t\ A\frac{s}{x}}{A\frac{t}{x}}(SUB)$$

3 Logikprogrammierung

3.1 Grundlagen

Prolog/Datalog arbeiten mit Klauseln, welche durch Backtracking ausgewertet werden. Eingegeben werden diese durch `Fakten`, welche als Wahrheitsaussage gelten, und `Regeln`, welche der Implikation ähneln. Ist ein Programm dadurch bestimmt, kann man durch die Formulierung von Zielen etwas beweisen lassen. Dabei werden Grossbuchstaben als Variablen, Kleinbuchstaben und Zahlen als Konstanten betrachtet.

Faktum $A.$

Regel $A :- B_1, \ldots, B_n.$

Ziel $R(t_1, \ldots, t_n).$

Prolog eignet sich für das Lösen von Problemen durch Backtracking, aber man kann auch relationale Datenbanken nachbilden.

3.2 Datalog

Ein Beispiel aus Serie 10, Aufgabe 3 (gekürzt). Durch die Datenbank aus Fakten (unterer Teil) und die Wissensbasis aus Regeln lassen sich Fragen wie "Ist r2 ein Auto" (`interpretation(r2, car)`) oder "Was ist r3" (`interpretation(r3, X)`) schnell durch Prolog beantworten.

```
large(Region, yes) :-
    size(Region, X), X >= 10000.
large(Region, no) :-
    size(Region, X), X =< 500.

interpretation(Region, forrest) :-
    large(Region, yes),
    vegetation(Region, yes),
    water(Region, no).

interpretation(Region, road) :-
    elongated(Region, yes),
    vegetation(Region, no),
    water(Region, no).

interpretation(Region, car) :-
    large(Region, no),
    vegetation(Region, no),
    water(Region, no).

size(r2, 632).
size(r3, 56).
elongated(r2, yes).
elongated(r3, no).
vegetation(r2, no).
vegetation(r3, no).
water(r2, no).
water(r3, no).
```

3.3 Unifikation

Unifikationsalgorithmus

Eingabe: Zwei Terme s und t.

U1: Setze $\theta_0 := \epsilon$ und $k := 0$. Gehe zu U2.

U2: Falls $s\theta_k = t\theta_k$, dann stoppe mit Resultat θ_k. Sonst gehe zu U3.

U3: Gehe von links her durch $s\theta_k$ und $t\theta_k$. Seien a und b die Teilterme, die an der ersten Position stehen, wo sich $s\theta_k$ und $t\theta_k$ unterscheiden.
Falls b eine Variable ist und a keine, dann vertausche a und b.
Falls a eine Variable ist und a nicht in b vorkommt, dann setze $\theta_{k+1} := \theta_k\{b/a\}$, addiere 1 zu k und gehe zu U2.
Sonst stoppe mit Antwort "nein".

Das Resultat des Unifikationsalgorithmus ist entweder eine Substitution oder die Meldung ñein.

Definiton MGU: Falls zwei Terme s und t unifizierbar sind, dann bezeichnen wir mit mgu(s,t) den vom Unifikationsalgorithmus berechneten allgemeinsten Unifikator von s und t (most general unifier).

Beispiel

k	$s\theta_k$	$t\theta_k$	a	b	θ_{k+1}
0	$f(a, x, f(g(y)))$	$f(z, f(z), f(w))$	a	z	$\frac{a}{z}$
1	$f(a, x, f(g(y)))$	$f(a, f(a), f(w))$	x	$f(a)$	$\frac{f(a)}{x}$
2	$f(a, f(a), f(g(y)))$	$f(a, f(a), f(w))$	$g(y)$	w	$\frac{g(y)}{w}$
3	$f(a, f(a), f(g(y)))$	$f(a, f(a), f(g(y)))$			

3.4 Prolog

Aber auch komplexere Probleme wie das Zusammenhängen einer Liste oder das Umdrehen ihrer Elemente lassen sich in Prolog Formulieren. [] entspricht dabei einer leeren Liste, $[A|R]$ einer Liste mit Kopfelement A, und Restliste R und $[a, b, c]$einer Liste mit den genannten Elementen. `app(X,Y,A)` hängt die Listen X und Y zu A zusammen, `rev(X,Y)` dreht X in Y um. `list(L)` ist eine Liste (ein geschlossener Term). `select(X,L1,L2)` löscht das erste Vorkommen von X in L1 und gibt L2 zurück. `delete(X,L1,L2)` löscht alle Vorkommen.

```
app([], L, L).
app([X|L1],L2,[X|L3]) :- app(L1,L2,L3).

rev([], []).
rev([X],[X]).
rev(L1,L2) :- app([X|LA], [Y], L1),
              app([Y|LB], [X], L2),
rev(LA, LB).

/*2te Variante*/
rev([],[]).
rev([X|L],M):-
 rev(L,N),
 app(N,[X],M).

/*1ter und letzter Eintrag loeschen*/
r(L1,L2) :- app([_|L2],[_],L1).

/*Permutation*/
perm([],[]).
perm([X|L1],[Y|L2]):-
 sel([X|L1],Y,L3),
 perm(L3,L2).

sel([X|L],X,L).
sel([Y|L1],X,[Y|L2]):-
 sel(L1,X,L2).

suffix(L1,L2) :- append(_,L1,L2).
prefix(L1,L2) :- append(L1,_,L2).

list([]).
list([_|L]) :- list(L).

sublist(L1,L2) :-
 suffix(L3,L2),
 prefix(L1,L3).

member(X,[X|_]).
member(X,[_|L]) :- member(X,L).

select(X,[X|L],L).
select(X,[Y|L1],[Y|L2]) :- select(X,L1,L2).

delete([],_,[]).
delete([Y|L1],X,L2) :-
 ( X=Y ->
  delete(L1,X,L2)
 ; L2=[Y|L3],
  delete(L1,X,L3)
 ).

/*loescht den zweiten Eintrag*/
second(X,[_,X|L]).
```

```
/*X letztes Element der Liste?*/
final(X,L) :- append(L1,[X],L).

/*oder*/
final(X,[X]).
final(X,[_|L]):-final(X,L).

/*L1 identisch mit L2, bis auf
Vertauschung von letztem und
erstem? */
swapf1([],[]).
swapf1([X],[X]).
equalminuslast([X],[Y]).
equalminuslast([X|L1],[X|L2]) :-
 equalminuslast(L1,L2).
swapf1([X|L1],[Y|L2]) :-
 final(X,[Y|L2]),
 final(Y,[X|L1]),
 equalminuslast(L1,L2).

/*auf 2x vorkommen testen*/
twice(X,L) :-
 append(_,[X|L1],L),
 member(X,L1).
```

Deklarative Konstrukte in Prolog

Rule = Atom ':-' Goal'.'

Goal = 'true' | 'fail' | Atom | Term '=' Term |
Goal ',' Goal | Goal ';' Goal | '\+' Goal |
Goal '->' Goal ';' Goal

= : "gleich"
, : "und"
; : "oder"
\+ : "nicht"
->; : "if-then-else"