Logik (2V+1Ü)

Übungen: im Rubikon anmelden
Tutorien: Di 10–12, Mi —, Do 14–16

Schriftl. Prüfungen: 22.7. $14-15^{00}$
18.10. $16-17^{00}$

Skript: Exzerpt aus: Logik für Informatiker.
Buch: Ebbinghaus-Flum-Thomas.

→ Semantik von Progr.-Sprachen / Verifikation von Hardware Software ; Korrektheit
KI: Planungssysteme. Wissensrepräsentation, Expertensysteme
(nicht-klassische Logiken) hier: nur klassische Logik (nur wahr/falsch)

---

bekannt: Boolesche Operationen
Wahrheitstafel

Aussagenlogik

A = „Ulm liegt an der Donau" (wahr) 1

B = „Schwefel ist ein Metall" (falsch) = 0

Aussagenvariablen: $A, B, C, \ldots, A_1, A_2, \ldots$
können noch wahr oder falsch sein.

A	B	$A \wedge B$	$A \vee B$	$A \oplus B$	$\neg A$	$A \to B$	$A \leftrightarrow B$
0	0	0	0	0	1	1	1
0	1	0	1	1	1	1	0
1	0	0	1	1	0	0	0
1	1	1	1	0	0	1	1
		„und"	„oder"	entweder-oder XOR	„nicht"	„daraus folgt"	gdw.

A = „Otto wird krank"
B = „Der Arzt verschreibt Otto eine Medizin"

} Vergleiche „A und B" mit „B und A"!

Wahrheitstafelmethode, anwenden auf

$$F = ((A \to B) \wedge (B \to C)) \to (A \to C)$$

A	B	C	$A \to B$	$B \to C$	$(\,) \wedge (\,)$	$A \to C$	$(\quad) \to (\,)$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	0	0	1	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	0	1
1	0	1	0	1	0	1	1
1	1	0	1	0	0	0	1
1	1	1	1	1	1	1	1

gültige Formel bzw. Tautologie

Schubfachschluss
(pidgeon hole principle)

$A_{i,j}$ Taube i in Taubenschlag j

$(i = 1, \dots, n+1; \quad j = 1, \dots, n)$

$$(A_{1,1} \vee A_{1,2} \vee \dots \vee A_{1,n})$$
$$\wedge (A_{2,1} \vee A_{2,2} \vee \dots \vee A_{2,n})$$
$$\vdots$$
$$\wedge (A_{n+1,1} \vee A_{n+1,2} \vee \dots \vee A_{n+1,n})$$
$$\wedge \underbrace{\neg(A_{1,1} \wedge A_{2,1}) \wedge \neg(A_{1,1} \wedge A_{3,1}) \wedge \dots \wedge \neg(A_{n,1} \wedge A_{n+1,1})}_{\binom{n+1}{2} \text{ viele}}$$
$$\vdots$$
$$\wedge \neg(A_{1,n} \wedge A_{2,n}) \wedge \neg(A_{1,n} \wedge A_{3,n}) \wedge \dots \wedge \neg(A_{n,n} \wedge A_{n+1,n})$$

... unerfüllbare Formel

Formale Definitionen: (Syntax)

Ind. anf.
- Die Aussagenvariablen sind Formeln
  z.B A, B, C
- Die Konstanten 0, 1 sind Formeln

Ind. schritt
- Wenn F bereits Formel ist, dann auch $\neg F$

| – Wenn F, G bereits Formeln sind, dann auch: $(F \wedge G), (F \vee G), (F \oplus G), (F \to G), (F \leftrightarrow G)$

--- übliche Klammerersparnisregeln:

$\neg AB \vee B(C \vee D)$

Eine Belegung ist eine Abbildung

$$\mathcal{A} : \underbrace{\{A_1, A_2, A_3, \ldots\}}_{\text{Aussagenvar.}} \longrightarrow \{0,1\}$$

(Semantik)

Gegeben eine Belegung $\mathcal{A}$, erweitere Anwendung von $\mathcal{A}$ auf Formeln:

$$\mathcal{A}(0) = 0 \quad , \quad \mathcal{A}(1) = 1$$

$$\mathcal{A}(\neg F) = \begin{cases} 1, & \mathcal{A}(F) = 0 \\ 0, & \mathcal{A}(F) = 1 \end{cases}$$

$$\mathcal{A}((F \wedge G)) = \begin{cases} 1, & \mathcal{A}(F) = 1 \text{ und } \mathcal{A}(G) = 1 \\ 0, & \text{sonst} \end{cases}$$

$$\mathcal{A}((F \vee G)) = \begin{cases} 1, & \mathcal{A}(F) = 1 \text{ oder } \mathcal{A}(G) = 1 \\ 0, & \text{sonst} \end{cases}$$

$$\mathcal{A}((F \to G)) = \begin{cases} 0, & \mathcal{A}(F) = 1 \text{ und } \mathcal{A}(G) = 0 \\ 1, & \text{sonst} \end{cases}$$

Def: Formel F heißt

- erfüllbar, falls es eine Belegung $\mathcal{A}$ gibt mit $\mathcal{A}(F) = 1$

- Tautologie, falls für alle Belegungen $\mathcal{A}$ gilt: $\mathcal{A}(F) = 1$

Satz: F ist Tautologie gdw. $\neg F$ ist unerfüllbar.

Beweis:

F ist Tautologie gdw. für alle $\mathcal{A}$ gilt $\mathcal{A}(F) = 1$
gdw: für alle $\mathcal{A}$ gilt $\mathcal{A}(\neg F) = 0$
gdw. für kein $\mathcal{A}$ gilt $\mathcal{A}(\neg F) = 1$
gdw. $\neg F$ ist nicht erfüllbar. □

Ähnlich:

$(F_1 \wedge \dots \wedge F_n) \to G$ Tautologie

gdw.

$\neg((F_1 \wedge \dots \wedge F_n) \to G)$ ist unerfüllbar

gdw.

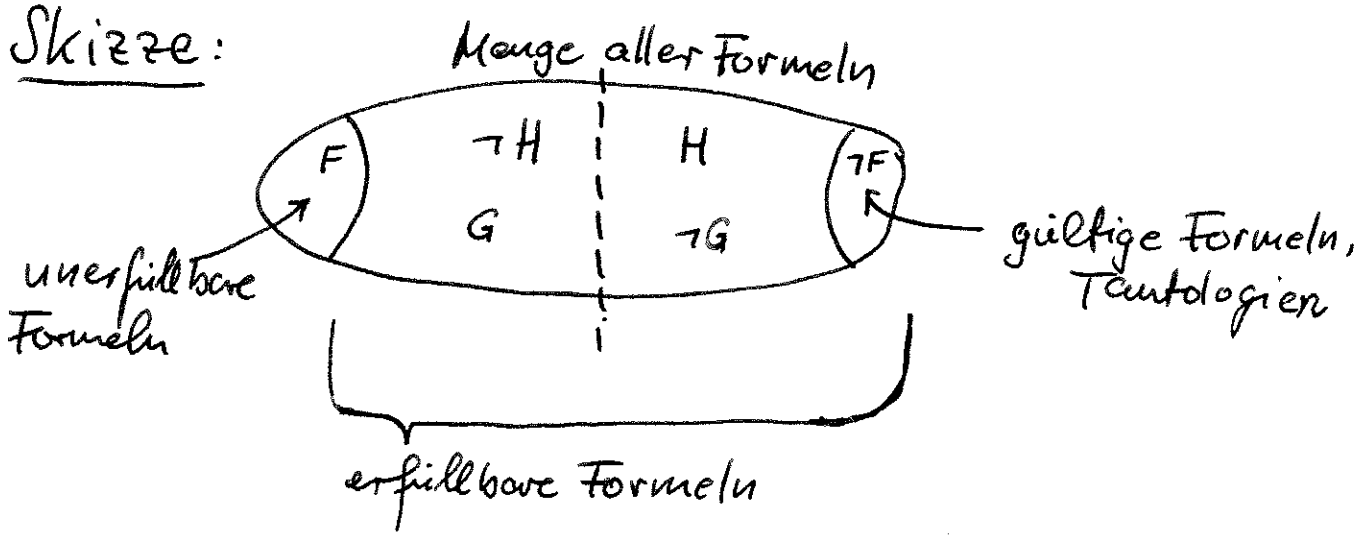
$\neg(\neg(F_1 \wedge \dots \wedge F_n) \vee G)$ ist unerfüllbar

gdw

$F_1 \wedge \dots \wedge F_n \wedge \neg G$ ist unerfüllbar

Wiederholung + Fragen zur letzten Vorlesung

Skizze:



Anwenden der Negation bedeutet Spiegeln an der gestrichelten Achse.

Frage: Gegeben eine Formel mit den Aussagenvariablen $A_1, A_2, \ldots, A_n$. Welchen Rechenaufwand erfordert es (im worst-case), um Festzustellen, ob die Formel erfüllbar oder unerfüllbar ist?

Antwort: Durchprobieren aller 2^n möglichen Belegungen, also exponentiellen Aufwand.
(Das Erfüllbarkeitsproblem der Aussagenlogik, kurz: SAT, ist NP-vollständig.)

Die Implikation $A \to B$; man sagt:
A impliziert B; aus A folgt B;
A ist hinreichend für B; B ist notwendig für A

Die Implikation $B \to A$ heißt die Umkehrung;
diese ist keineswegs äquivalent zu $A \to B$.

Dagegen nennt man $\neg B \to \neg A$ die Kontraposition
von $A \to B$; diese ist äquivalent zu $A \to B$.
(Tatsächlich ist dies das Prinzip des indirekten
Beweises.)

Aufgabe: Gib Wahrheitstafeln für die Formeln
$\neg(A \wedge B)$ sowie $(\neg A \vee \neg B)$ an und vergleiche
den Wahrheitswerteverlauf.

Antwort:

	A	B	$A \wedge B$	$\neg(A \wedge B)$	$\neg A$	$\neg B$	$(\neg A \vee \neg B)$
$\mathcal{A}_1$:	0	0	0	1	1	1	1
$\mathcal{A}_2$:	0	1	0	1	1	0	1
$\mathcal{A}_3$:	1	0	0	1	0	1	1
$\mathcal{A}_4$:	1	1	1	0	0	0	0

$=$

Die Wahrheitswerteverläufe sind identisch.
Dies beweist die de Morgan'sche Regel
$\neg(A \wedge B) = (\neg A \vee \neg B)$.

Frage: Unterschied klassische / nicht-klassische Logik, Beispiele?

Antw: klassisch: Wahrheitswerte sind 0 und 1.
nicht-klassisch: andere Wahrheitswerte möglich, z.B.
{wahr, falsch, möglich, notwendig} ... Modallogik
$[0,1]$ ... Fuzzy Logic

Aufgabe: Gib eine Formel mit den Aussagenvariablen $A_1, \ldots, A_n$ an, die genau 2 erfüllende Belegungen besitzt, nämlich entweder

$$\mathcal{A}(A_1) = \cdots = \mathcal{A}(A_n) = 1$$

oder

$$\mathcal{A}(A_1) = \cdots = \mathcal{A}(A_n) = 0$$

1. Lösung: $(A_1 \wedge A_2 \wedge \dots \wedge A_n) \vee (\neg A_1 \wedge \neg A_2 \wedge \dots \wedge \neg A_n)$

2. elegantere(?) Lösung:

$$(A_1 \to A_2) \wedge (A_2 \to A_3) \wedge \dots \wedge (A_{n-1} \to A_n) \wedge (A_n \to A_1)$$

Disjunktive Normalform, am Beispiel: (DNF)

$$F = (A_1 \wedge \bar{A}_2 \wedge A_5) \vee (\bar{A}_2 \wedge \bar{A}_5) \vee (\bar{A}_1 \vee \bar{A}_3 \vee A_5)$$

Literale — Disjunktionen (von Konjunktionen)

Konjunktive Normalform (KNF)

$$G = (A_1 \vee A_2 \vee \bar{A}_4) \wedge (A_3) \wedge (\bar{A}_3 \vee A_6) \wedge (\bar{A}_1 \vee \bar{A}_2)$$

Konjunktionen (von Disjunktionen)

Aus der Wahrheitstafel die DNF herstellen:

A	B	C	F	
0	0	0	0	F =
0	0	1	1	$\longrightarrow (\bar{A} \wedge \bar{B} \wedge C) \vee$
0	1	0	1	$\longrightarrow (\bar{A} \wedge B \wedge \bar{C}) \vee$
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$\longrightarrow (A \wedge B \wedge \bar{C})$
1	1	1	0	

(vollständige DNF)

vereinfachte DNF: $(\bar{A} \wedge \bar{B} \wedge \bar{C}) \vee (B \wedge \bar{C})$

Beliebige Formeln können in DNF umgeformt werden:

$\neg(A \vee B) = (\bar{A} \wedge \bar{B})$, $A \wedge (B \vee C) = (A \wedge B) \vee (B \wedge C)$

Aus der Wahrheitstafel die KNF herstellen:

A	B	C	G
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$G = (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C) \wedge (\overline{A} \vee B \vee \overline{C})$

„Klauseln"

keine Horn-Klauseln

„Literale"

<u>Hornformel</u> ist Formel in KNF, wobei in jeder „Klausel" höchstens ein positives Literal enthalten ist.

Hornklauseln zerfallen in verschiedene Kategorien:

- genau ein Literal, das positiv ist, : <u>Faktenklausel</u>
  kein negatives, z.B. (A)
- ein positives Literal, und einige negative Literale, z.B. $\underbrace{(A \vee \overline{B_1} \vee \overline{B_2})}_{\text{äquivalent: } (A \leftarrow (B_1 \wedge B_2))}$: <u>Regelklausel</u>
- rein negative Klausel, : <u>Zielklausel</u>
  z.B. $(\overline{B_1} \vee \overline{B_2})$

Effizienter Test auf Erfüllbarkeit/Unerfüllbarkeit, wenn die Eingabe eine Hornformel ist.

- Setze zunächst die Belegung aller Aussagenvariablen auf 0
- Für jede Faktenklausel (A) setze die Belegung von A auf 1.
- Wiederhole in einer Schleife:

  Wenn es eine Regelklausel $(A \vee \overline{B}_1 \vee \dots \vee \overline{B}_k)$ bzw $(A \leftarrow (B_1 \wedge \dots \wedge B_k))$

  gibt, wobei A noch auf 0 gesetzt ist und alle B_i auf 1 gesetzt sind,

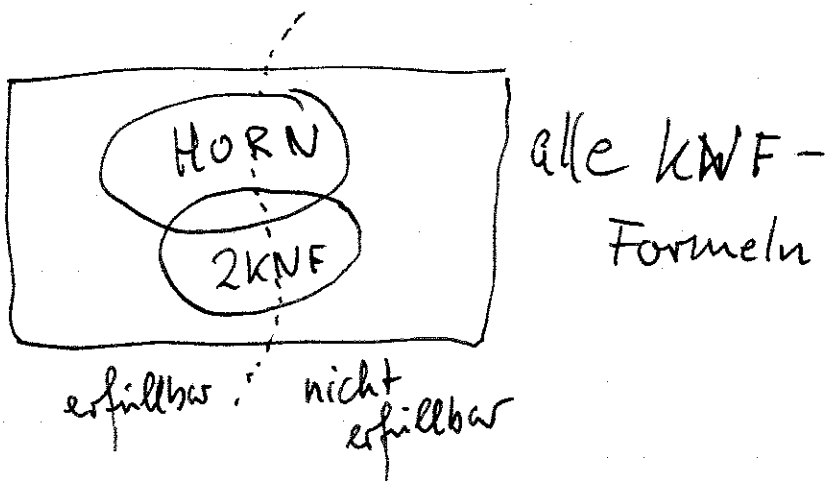
  dann setze A auf 1.

Nach diesem Algorithmus ist eine bestimmte Belegung $\mathcal{A}$ entstanden. Diese hat die Eigenschaft, dass sie alle Fakten- und Regelklauseln erfüllt, wobei so wenige 1-Setzungen wie möglich erfolgt sind.

Nun wird $\mathcal{A}$ auf die Zielklauseln angewandt. Die Ausgangsformel ist erfüllbar genau dann, wenn $\mathcal{A}$ alle Zielklauseln erfüllt. (weil $\mathcal{A}$ „Minimalbelegung" ist)

Anderer Spezialfall von KNF besitzt ebenfalls effizienten Erfüllbarkeitstest:

<u>2KNF</u>: alle Klauseln bestehen aus ≤ 2 Literalen.



<u>Bem:</u> Erfüllbarkeit bei 3KNF (kurz: 3SAT) ist NP-vollständig.

Resolution: $(A \vee B) \quad (\bar{A} \vee \bar{C})$

$\Big\vee$

$(B \vee \bar{C})$ ... Resolvent

Es gilt: Formel in KNF ist unerfüllbar

gdw.

es gelingt, per Resolution aus den Klauseln die leere Klausel abzuleiten.

Beispiel zur Resolution (ausführlich: später)

$F = (A \vee B) \wedge (B \vee \bar{C} \vee D) \wedge (\bar{D}) \wedge (C \vee \bar{A}) \wedge (\bar{B})$

$(B \vee \bar{C})$

$(B \vee \bar{A})$

(B)

$()$ andere Schreibweise: $\square$

Im Spezialfall 2KNF:

$(A \wedge \bar{B})\ (B \vee C)$

$(A \vee C)$

Auch Resolventen 2KNF

bei 3KNF

$(A \vee B \vee C)\ (\bar{A} \vee D \vee \bar{E})$

$(B \vee C \vee D \vee \bar{E})$

Resolvent der Länge 4

Anzahl der Klauseln in 2KNF:

$1 + 2n + \binom{2n}{2} = O(n^2)$ polynomial

bei n Aussagenvar.

Dies ergibt effizienten polynomialen Algorithmus.

Kalküle

Sammlung von Umformungsregeln, die gegebene Formel F manipulieren. (z.B. deMorgan, Distributivgesetz, ...)

Ziel der Umformung: Nachweis, dass F eine Tautologie / oder unerfüllbar ist.

Ein Kalkül birgt Nichtdeterminismus in sich: Mehrere Regeln können angewandt werden – manche führen in Sackgassen oder erzeugen viele überflüssige Schritte. Selbst der kürzeste Kalkül-Beweis kann evtl. exponentiell viele Schritte umfassen.

Resolutionskalkül

Nur anwendbar auf Formeln in KNF (Klauselmengen).

Ein Resolutionsschritt, angewandt auf 2 Klauseln, setzt voraus, dass eine Variable in der einen Klausel positiv und in der anderen Klausel negiert vorkommt:

$$K_1 = (A \vee \circ\circ\circ\circ) \qquad K_2 = (\bar{A} \vee \square\square\square\square)$$

Das Ergebnis ist der Resolvent K_3, der alles aus K_1 – bis auf A – und alles aus K_2 – bis auf $\bar{A}$ – enthält:

$$K_3 = (\circ\circ\circ\circ \vee \square\square\square\square)$$

In Formeln: $K_3 = (K_1 \setminus \{A\}) \cup (K_2 \setminus \{\bar{A}\})$

Bemerkung: Unzulässig ist

$$\begin{array}{c} (A \vee B \vee C) \quad (\bar{A} \vee \bar{B} \vee D \vee \bar{E}) \\ \diagdown\!\diagup \\ (C \vee D \vee \bar{E}) \end{array} \qquad (\text{f})$$

Begriffe/Notationen:

Mittels Kalkül $\mathcal{K}$ kann F zu G umgeformt werden: $F \vdash_{\mathcal{K}} G$.

Ein Kalkül bestimmt eine syntaktische Art der Umformung. Zur Beurteilung von Kalkülen muss man das syntaktische Umformen (Notation $\vdash_{\mathcal{K}}$) vergleichen mit semantischer Äquivalenz oder Folgerung:

Def: F und G sind (semantisch) äquivalent, falls für alle Belegungen $\mathcal{A}$ (welche alle Aussagenvariablen in F und G umfassen) gilt:

$$\mathcal{A}(F) = \mathcal{A}(G).$$

Notation für semantische Äquivalenz: $F \equiv G$

(Wir verwenden aber auch das normale Gleichheitszeichen). Beispiel:

$$\underbrace{\neg(A \vee B)}_{F} \quad = \quad \underbrace{\neg A \wedge \neg B}_{G}$$

Frage: Können Formeln F, G auch dann äquivalent sein, wenn sie unterschiedliche Mengen von vorkommenden Aussagenvariablen enthalten?

Antwort: Ja, aber nur im Fall, dass beides Tautologien oder beides unerfüllbare Formeln sind. Beispiel:

$$(A \vee \neg A) \quad \equiv \quad (B \wedge C) \vee (\neg B \wedge C) \vee (B \wedge \neg C) \vee (\neg B \wedge \neg C)$$

Def: Formel G ist eine (semantische) Folgerung von F (Notation: $F \models G$) falls für alle Belegungen $\mathcal{A}$ mit $\mathcal{A}(F)=1$ gilt, dass auch $\mathcal{A}(G)=1$.

Beispiel: $A \wedge B \models A \vee B$

Satz: Es gilt $F \models G$ genau dann, wenn $\models (F \to G)$ (d.h. $(F \to G)$ ist Tautologie).

Kalkül

$\mathcal{K}$ heißt **korrekt**, falls:

Wenn $F \vdash_{\mathcal{K}} G$, dann ist G Folgerung von F

Ein Kalkül

$\mathcal{K}$ heißt **vollständig**, falls:

Wenn $F \models G$, dann $F \vdash_{\mathcal{K}} \cdots \vdash_{\mathcal{K}} G$

$\mathcal{K}$ heißt **widerlegungsvollständig**, falls für alle unerfüllbaren F gilt, dass $F \vdash_{\mathcal{K}} \cdots \vdash_{\mathcal{K}} \square$

Satz: Der Resolutionskalkül Res ist korrekt und widerlegungsvollständig.

Beweis: (Korrektheit)

Es gelte $K_1 \quad K_2$ (mit Resolvente K_3). Sei $\mathcal{A}$ eine Belegung mit $\mathcal{A}(K_1 \wedge K_2) = 1$. Sei $K_1 = (A \vee \circ\circ\circ\circ)$, $K_2 = (\bar{A} \vee \square\square\square\square)$

Falls $\mathcal{A}(A) = 1$, so muss für ein $\square$-Literal in K_2 gelten: $\mathcal{A}(\square) = 1$. Also folgt auch $\mathcal{A}(K_3) = 1$.

Falls $\mathcal{A}(\bar{A}) = 1$, so muss für ein $\circ$-Literal in K_1 gelten: $\mathcal{A}(\circ) = 1$. Also folgt auch $\mathcal{A}(K_3) = 1$.

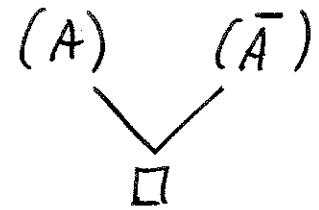
Das bedeutet, wenn aus $F = K_1 \wedge K_2 \wedge \ldots \wedge K_m$ durch Resolutionsschritte die leere Klausel $()$ abgeleitet werden kann, dann ist F unerfüllbar. $\square$

(Widerlegungsvollständigkeit)

Angenommen, F ist unerfüllbar. Wir zeigen, dass es eine Abfolge von Resolutionsschritten gibt, die mit den Klauseln von F startet und mit der leeren Klausel endet.

Beweis durch Induktion über die Anzahl n der in F vorkommenden Aussagenvariablen.

(n=1) Wenn F unerfüllbar ist und nur 1 Variable A enthält, so muss F die Klauseln (A) und $(\bar{A})$ enthalten. Dann haben wir



(n→n+1) Sei F eine KNF-Formel mit n+1 Variablen, die unerfüllbar ist.
Sei A eine Variable in F. Dann sind sowohl

$F|_{A=0} =: F_0$ als auch $F|_{A=1} =: F_1$

unerfüllbare KNF-Formeln.

Bsp: $(A, \bar{C})\ (A, B, C)\ (\bar{A}, \bar{B}, \bar{C})\ (\bar{A}, B)\ (\bar{B}, C) = F$

ist unerfüllbar.

Dann ist

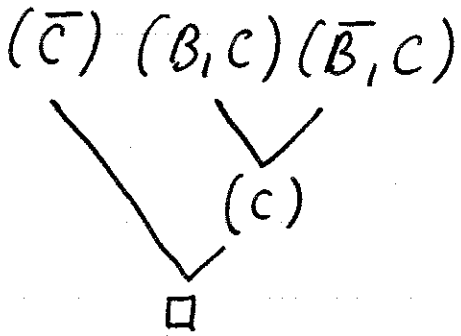
$F|_{A=0} = (\bar{C})\ (B, C)\ (\bar{B}, C)$

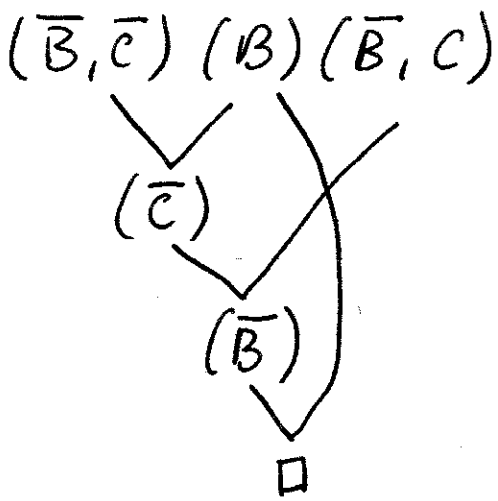
sowie

$F|_{A=1} = (\bar{B}, \bar{C})\ (B)\ (\bar{B}, C)$ unerfüllbar.

Nach Ind. voraus. besitzt sowohl $F|_{A=0}$ als auch $F|_{A=1}$ eine Resolutionsherleitung der leeren Klausel, da sie nur noch $\leq n$ Variablen enthalten.

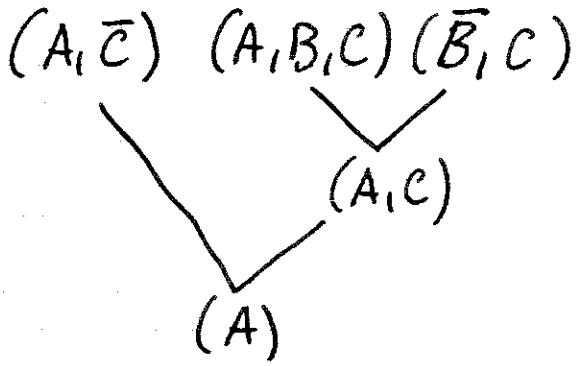
Am Beispiel:



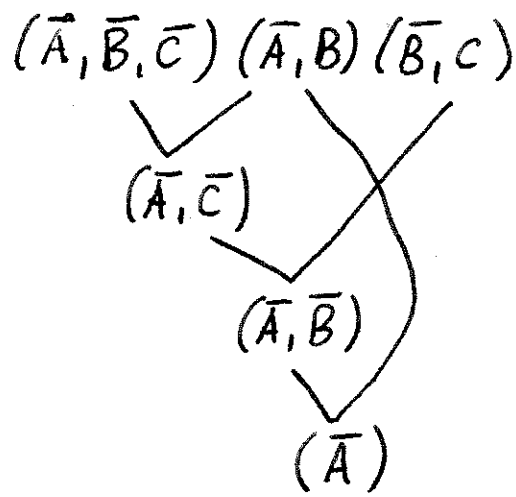


Stellt man nun die ursprünglichen Klauseln wieder her (Einfügen von A in $F|_{A=0}$ und von $\overline{A}$ in $F|_{A=1}$).

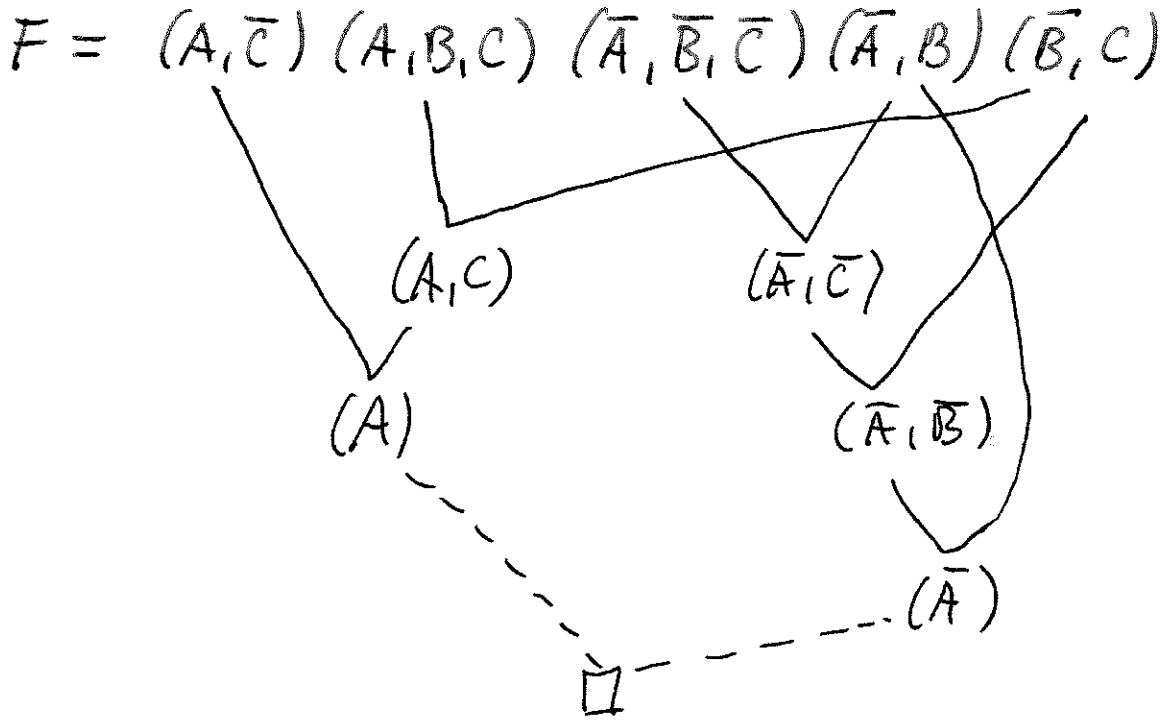
So erhält man, am Beispiel:



sowie:



Zusammengefügt als Resolutionsschritte von F aus ergibt:



Ein weiterer Resolutionschritt (A) (Ā) ergibt die leere Klausel. ▨

Länge der so erzeugten Res.beweise, bei n Aussagenvariablen, ist exponentiell ($\geq 2^n - 1$)

Interessante Tatsache: Sowohl Erfüllbarkeit als auch Unerfüllbarkeit kann durch eine Existenzaussage ausgedrückt werden:

F erfüllbar gdw. es gibt eine erfüllende Belegung $\mathcal{A}$ für F.

F unerfüllbar gdw. es gibt einen Resolutionsbeweis, startend mit F, endend mit $\square$.

Die erste Aussage impliziert: $SAT \in NP$

(→ Vorlesung Berechenbarkeit + Komplexität)

Bedeutet die 2. Aussage, dass $\overline{SAT} \in NP$?

(das wäre eine Sensation, fast wie $P = NP$)

Nein, und zwar deshalb nicht, weil Resolutionsbeweise exponentielle Länge haben können.

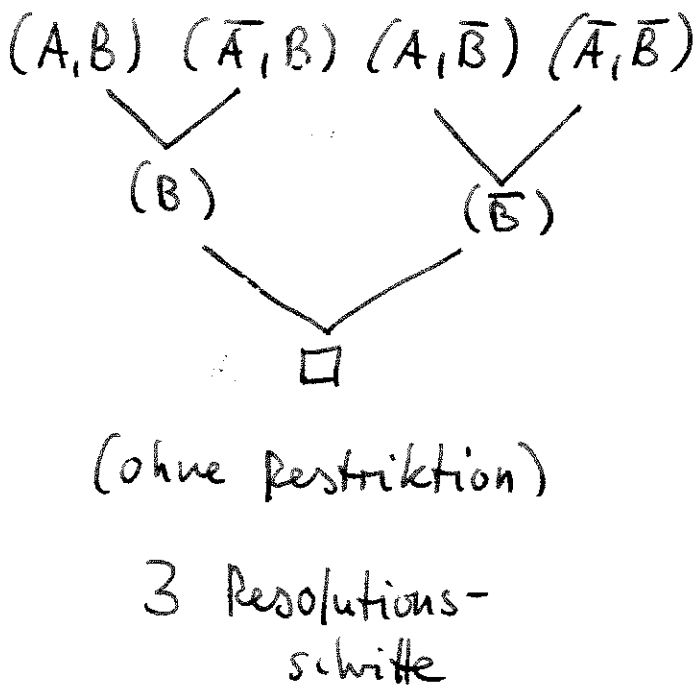
Satz: Jeder Resolutionsbeweis für das unerfüllbare Pidgeonhole-Problem mit $n+1$ Tauben und n Taubenschlägen benötigt $\geq 2^{n/20}$ viele Resolutionsschritte.

(→ Vorlesung SAT-Solving, Master)

Weitere Begriffe:

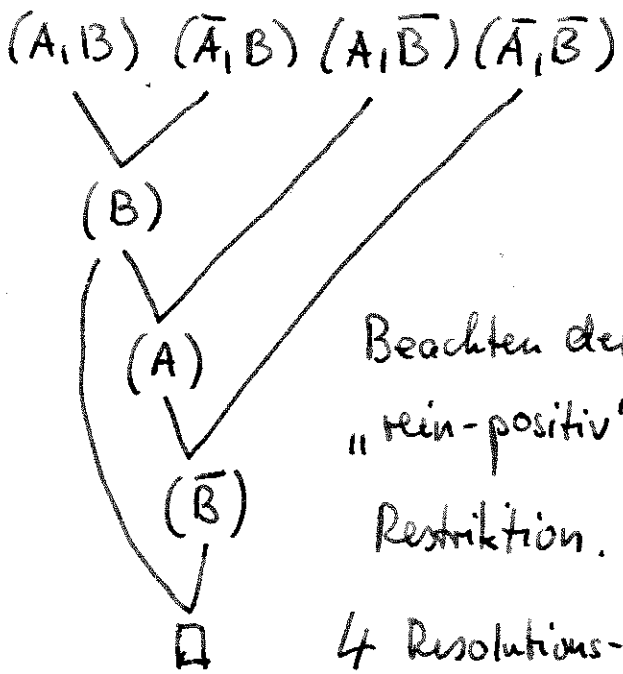
Schränkt man den Resolutionskalkül per Definition ein (z.B. so: Eine der beiden Elternklauseln muss rein-positiv sein), so spricht man von einer Resolutionsrestriktion. Hierdurch wird der Grad an Nichtdeterminismus eingeschränkt. Andererseits kann es sein, dass durch eine zu starke Restriktion die Widerlegungsvollständigkeit verloren geht — oder es kann sein, dass die Beweise bei Einhalten der Restriktion länger werden.

Beispiel:



(ohne Restriktion)

3 Resolutions-schritte



Beachten der „rein-positiv"-Restriktion.

4 Resolutions-schritte

Bei einer Resolutionsstrategie wird die Reihenfolge des Resolutionsschritte vorgeschrieben. Dadurch wird der Nichtdeterminismus des Kalküls eliminiert (oder zumindest eingeschränkt). Aus dem Kalkül wird ein Algorithmus.

Beispiel: Davis-Putnam-Strategie/Algorithmus:
Die vorkommenden Variablen seien $A_1, A_2, \ldots, A_n$:

for $i := 1$ to n do

Bilde alle Resolventen, die sich aus Elternklauseln bilden lassen, die A_i bzw. $\overline{A_i}$

→ Entferne danach alle beteiligten Elternklauseln

dies ist eine Restriktion

Beispiel:

$$(A_1, \overline{A_3})\ (A_1, A_2, A_3)\ (\overline{A_1}, \overline{A_2}, \overline{A_3})\ (\overline{A_1}, A_2)\ (\overline{A_2}, A_3)$$

- Alle Resolventen bzgl. A_1 bilden:

$(\bar{A}_2, \bar{A}_3)\ (A_2, \bar{A}_3)\ (A_2, A_3\ \bar{A}_2, \bar{A}_3)\ (A_2, A_3)$

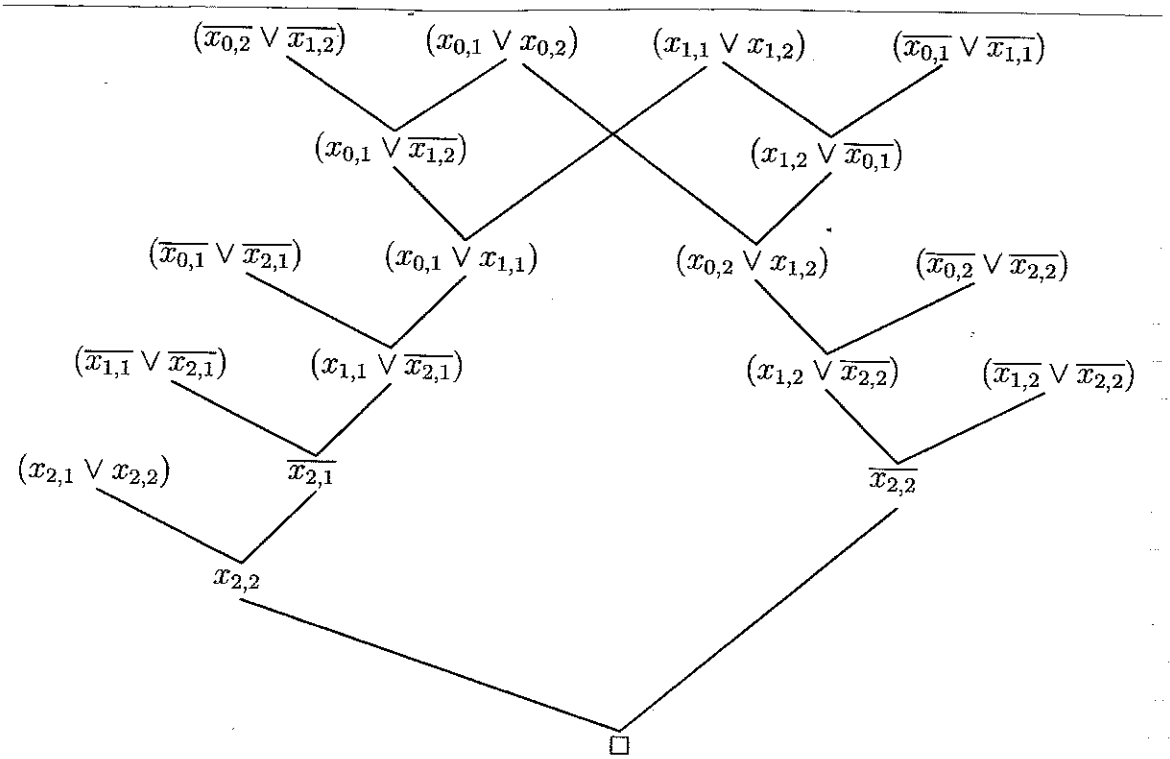
Aus F verbleibt außerdem noch: $(\bar{A}_2, A_3)$

- Alle Resolventen nach A_2 bilden:

  $(\bar{A}_3)$, (A_3), und weitere

- Unter den Resolventen mittels A_3 befindet sich die leere Klausel $\square$.

3 Tauben und 2 Taubenschläge:



-------- PROOF ---------

1 -p11 | -p12.
2 -p11 | -p13.
3 -p11 | -p14.
4 -p12 | -p13.
5 -p12 | -p14.
6 -p13 | -p14.
7 -p21 | -p22.
8 -p21 | -p23.
9 -p21 | -p24.
10 -p22 | -p23.
11 -p22 | -p24.
12 -p23 | -p24.
13 -p31 | -p32.
14 -p31 | -p33.
15 -p31 | -p34.
16 -p32 | -p33.
17 -p32 | -p34.
18 -p33 | -p34.
19 p11 | p21 | p31.
20 p12 | p22 | p32.
21 p13 | p23 | p33.
22 p14 | p24 | p34.

Pidgeon-Hole-Klauseln 4 Tauben und 3 Taubenschläge

23 [res:19,3] p21 | p31 | -p14.
24 [res:19,2] p21 | p31 | -p13.
25 [res:19,1] p21 | p31 | -p12.
26 [res:19,9] p11 | p31 | -p24.
27 [res:19,8] p11 | p31 | -p23.
30 [res:19,14] p11 | p21 | -p33.
32 [res:20,5] p22 | p32 | -p14.
33 [res:20,4] p22 | p32 | -p13.
34 [res:20,1] p22 | p32 | -p11.
41 [res:21,6] p23 | p33 | -p14.
42 [res:21,4] p23 | p33 | -p12.
43 [res:21,2] p23 | p33 | -p11.
52 [res:22,3] p24 | p34 | -p11.
53 [res:22,12] p14 | p34 | -p23.
54 [res:22,11] p14 | p34 | -p22.
55 [res:22,9] p14 | p34 | -p21.
57 [res:22,17] p14 | p24 | -p32.
60 [res:23,8] p31 | -p14 | -p23.
77 [res:25,14] p21 | -p12 | -p33.
82 [res:26,13] p11 | -p24 | -p32.
84 [res:27,15] p11 | -p23 | -p34.
97 [res:32,10] p32 | -p14 | -p23.
98 [res:32,7] p32 | -p14 | -p21.
107 [res:33,7] p32 | -p13 | -p21.
113 [res:34,11] p32 | -p11 | -p24.
114 [res:34,10] p32 | -p11 | -p23.
160 [res:41,16] p23 | -p14 | -p32.
161 [res:41,14] p23 | -p14 | -p31.
181 [res:43,16] p23 | -p11 | -p32.
265 [res:52,18] p24 | -p11 | -p33.
277 [res:53,17] p14 | -p23 | -p32.
289 [res:54,15] p14 | -p22 | -p31.
298 [res:55,18] p14 | -p21 | -p33.
299 [res:55,17] p14 | -p21 | -p32.
337 [res:60,13] -p14|-p23|-p32.
888 [res:337,277] -p23 | -p32.
889 [res:337,160] -p14 | -p32.
890 [res:337,97] -p14 | -p23.
891 [res:888,181] -p32 | -p11.
894 [res:888,114] -p23 | -p11.
897 [res:889,299] -p32 | -p21.
899 [res:889,57] -p32 | p24.
900 [res:889,98] -p14 | -p21.
903 [res:890,53] -p23 | p34.
904 [res:890,161] -p14 | -p31.
907 [res:891,113] -p11 | -p24.
909 [res:891,82] -p32 | -p24.
914 [res:894,43] -p11 | p33.
915 [res:894,84] -p23 | -p34.
918 [res:897,107] -p21 | -p13.
923 [res:900,298] -p21 | -p33.
932 [res:904,289] -p31 | -p22.
939 [res:907,265] -p11 | -p33.
944 [res:909,899] -p32.
946 [res:944,20] p12 | p22.
953 [res:915,903] -p23.
954 [res:953,42] p33 | -p12.
955 [res:953,21] p13 | p33.
957 [res:918,24] -p13 | p31.
959 [res:923,77] -p33 | -p12.
960 [res:923,30] -p33 | p11.
975 [res:939,914] -p11.
995 [res:959,954] -p12.
997 [res:995,946] p22.
1001 [res:997,932] -p31.
1002 [res:1001,957] -p13.
1003 [res:1002,955] p33.
1004 [res:960,1003] p11.
1006 [res:1004,975].

------- end of proof --------

Aufgabe: Man zeige, dass folgender „Resolutionsschritt" inkorrekt ist:

$$k_1 = \{A, B, C\} \qquad \{\bar{A}, \bar{B}, C, \bar{D}\} = k_2$$

$$\{C, \bar{D}\}$$

Antwort: Betrachte die Belegung

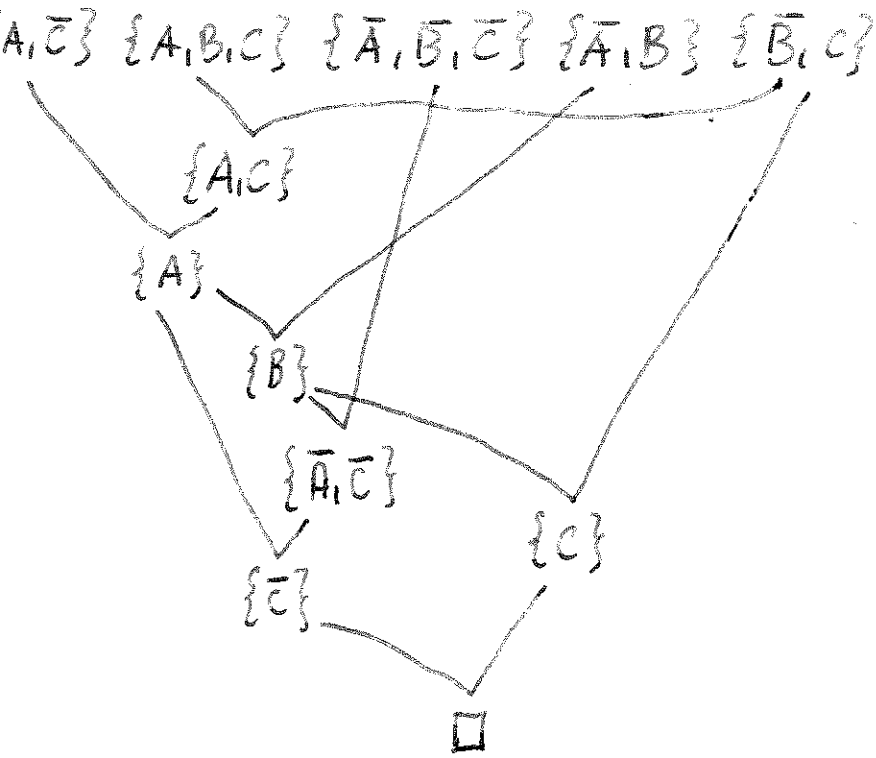
$\mathcal{A}$: $A \mapsto 1$, $B \mapsto 0$, $C \mapsto 0$, $D \mapsto 1$. Diese Belegung erfüllt die beiden Ausgangsklauseln k_1 und k_2, aber nicht den „Resolventen".

Aufgabe: Gib einen Resolutionsbeweis, der auf die leere Klausel führt, für folgende Klauselmenge an:

$$\{A, \bar{C}\}\ \{A, B, C\}\ \{\bar{A}, \bar{B}, \bar{C}\}\ \{\bar{A}, B\}\ \{\bar{B}, C\}$$

Dieser Resolutionsbeweis soll die Restriktion „ein Elternteil muss rein-positiv sein" einhalten.

Antwort:



Aufgabe: Bei einer erfüllbaren Klauselmenge kann man nicht die leere Klausel ableiten. Für jede Variable x kann man $\{x\}$ oder $\{\bar{x}\}$ (ggf. die Klausel) ableiten, aber nicht beide.
Entwickle mit dieser Tatsache einen Algorithmus, der mittels Resolution eine erfüllende Belegung findet.

Antwort: Finde (möglichst wenige) Resolutionsschritte, die für eine Variable x entweder $\{x\}$ oder $\{\bar{x}\}$ ableiten können. Setze $x \mapsto 1$ im ersten Fall und $x \mapsto 0$ im 2. Fall. Setze diesen Wert in die Formel ein, simplifiziere, und fahre so fort.

Beispiel: $\{x,z\}\ \{\bar{x},z\}\ \{\bar{x},y,z\}\ \{\bar{y},\bar{z}\}\ \{x,\bar{z}\}$

Aus den ersten 2 Klauseln lässt sich $\{z\}$ ableiten. Setze also $z \mapsto 1$ und setze ein. Dies ergibt:

$$\{x\}\ \{\bar{y}\}$$

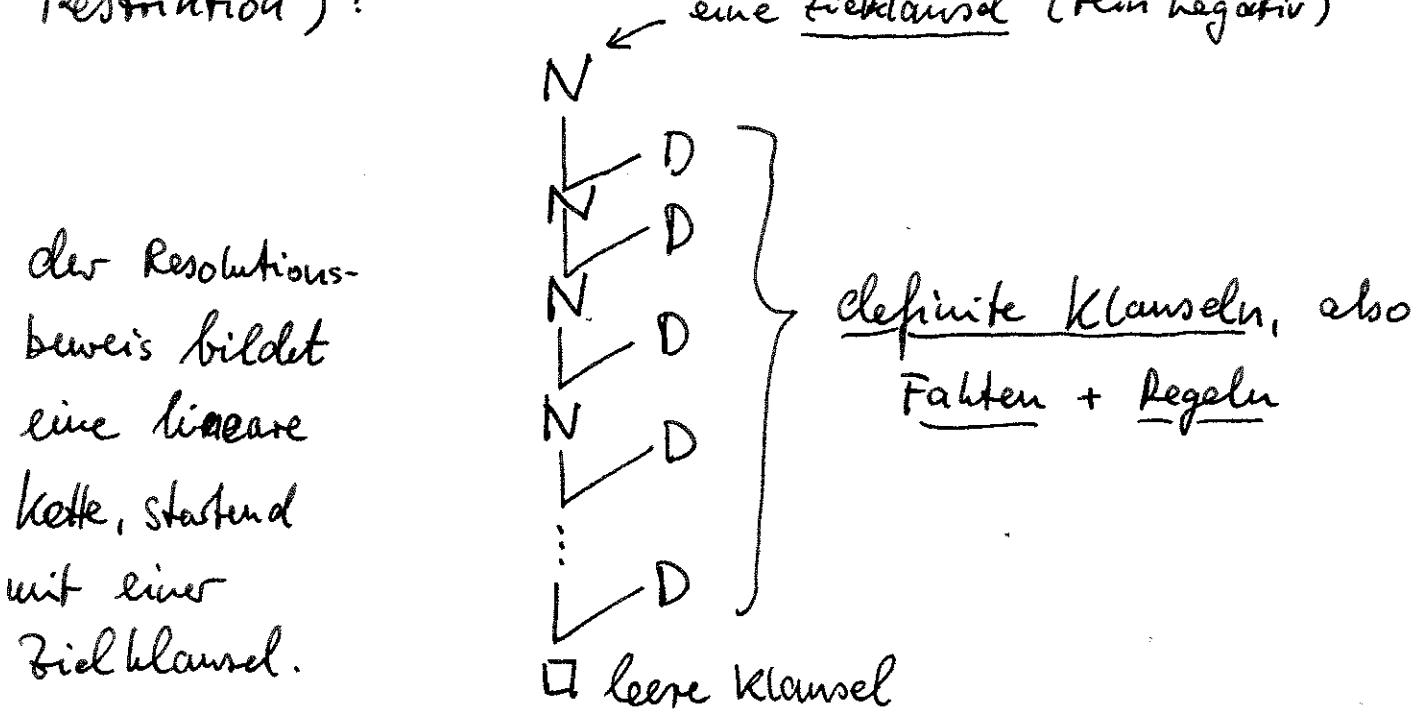
Nun kann man noch $x \mapsto 1$ und $y \mapsto 0$ setzen. Dies ergibt eine erfüllende Belegung.

Aufgabe: Zeige an einem Beispiel, dass die Resolutionsrestriktion „eine der Elternklauseln muss einelementig sein" nicht widerlegungsvollständig ist.

Antwort: $\{A, B\}$ $\{\bar{A}, B\}$ $\{A, \bar{B}\}$ $\{\bar{A}, \bar{B}\}$

Diese Klauselmenge ist unerfüllbar, aber es gibt keine einelementige Klausel. (Diese Restriktion ist jedoch für Hornklauseln widerleg-vollst.)

Um die Unerfüllbarkeit einer Horn-Formel nachzuweisen, kann man folgende spezielle Art eines Resolutionsbeweises verwenden (also eine auf Hornformeln zugeschnittene Art der Resolutions-Restriktion):



Diese Art der Resolution (mit prädikatenlogischen Hornklauseln) ist die Basis für die (Interpretation der) Programmiersprache PROLOG. Sie ist widerlegungsvollständig (für Hornformeln).

Eine Zwischenüberlegung:

Man kann das Konzept der Erfüllbarkeit/Unerfüllbarkeit auch auf Mengen von unendlich vielen Formeln bzw. Klauseln anwenden:

$$M = \{K_1, K_2, K_3, \dots\}$$

Eine solche Menge enthält dann auch unendlich viele Aussagenvariablen $A_1, A_2, A_3, \dots$

M ist erfüllbar, falls es Belegung $\mathcal{A}: \{A_1, A_2, \dots\} \to \{0,1\}$ gibt, so dass $\mathcal{A}(K_i) = 1$ für alle i gilt.

Andernfalls heißt M unerfüllbar.

Kann man die Unerfüllbarkeit hier auch mittels Resolution zeigen, denn ein Resolutionsbeweis besteht nur aus endlich vielen Klauseln und fußt auf einer endlichen Teilmenge von M.

Ja, man kann: **Endlichkeitssatz:** (im Skript-Anhang)

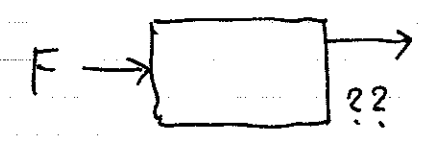
compactness theorem

Eine solche unendliche Menge M ist unerfüllbar genau dann, wenn es eine endliche Teilmenge von M gibt, die unerfüllbar ist.

Vorschau: Wir werden im Kapitel über Prädikatenlogik zeigen, dass die Frage nach der Erfüllbarkeit / Unerfüllbarkeit einer einzelnen prädikatenlogischen Formel zurückgeführt werden kann auf eine unendliche Menge M von aussagenlogischen Formeln. Diese Menge M muss dann auf Erfüllbarkeit / Unerfüllbarkeit getestet werden. Sollte Unerfüllbarkeit vorliegen, so kann man dies nach endlicher Zeit feststellen. Wegen des Endlichkeitssatzes muss es eine endliche unerfüllbare Teilmenge von M geben, deren Unerfüllbarkeit man dann mittels Resolution nachweisen kann.

Sollte M erfüllbar sein (dann auch die ursprüngliche prädikatenlog. Formel), so wird der betreffende Algorithmus nie stoppen

(→ Semi-Entscheidbarkeit, Vorlesung Berechenbarkeit)



Prädikaten- oder Quantorenlogik

Bisher: Aussagenvariablen $A, B, C, \ldots$ die einen festen (aber evtl. unbekannten) Wahrheitswert $\in \{0,1\}$ besitzen.

In der Prädikatenlogik kann der Wahrheitswert noch dynamisch vom Wert einer (oder mehrerer) Variablen x abhängen. Statt $A(x)$ schreiben wir nun $P(x), Q(x), R(x)$ etc. und nennen diese Objekte Prädikate. Es ist noch komplizierter: die Variablen x, y, z können zunächst noch mittels Funktionen f, g, h Wertveränderungen unterworfen sein, so dass die Prädikate dann folgende Formen haben können:

$$P(x, f(y)), \quad Q(\underbrace{f(g(x))}), \quad R(x, y, \underbrace{h(z, f(x))})$$

diese Objekte werden Terme genannt

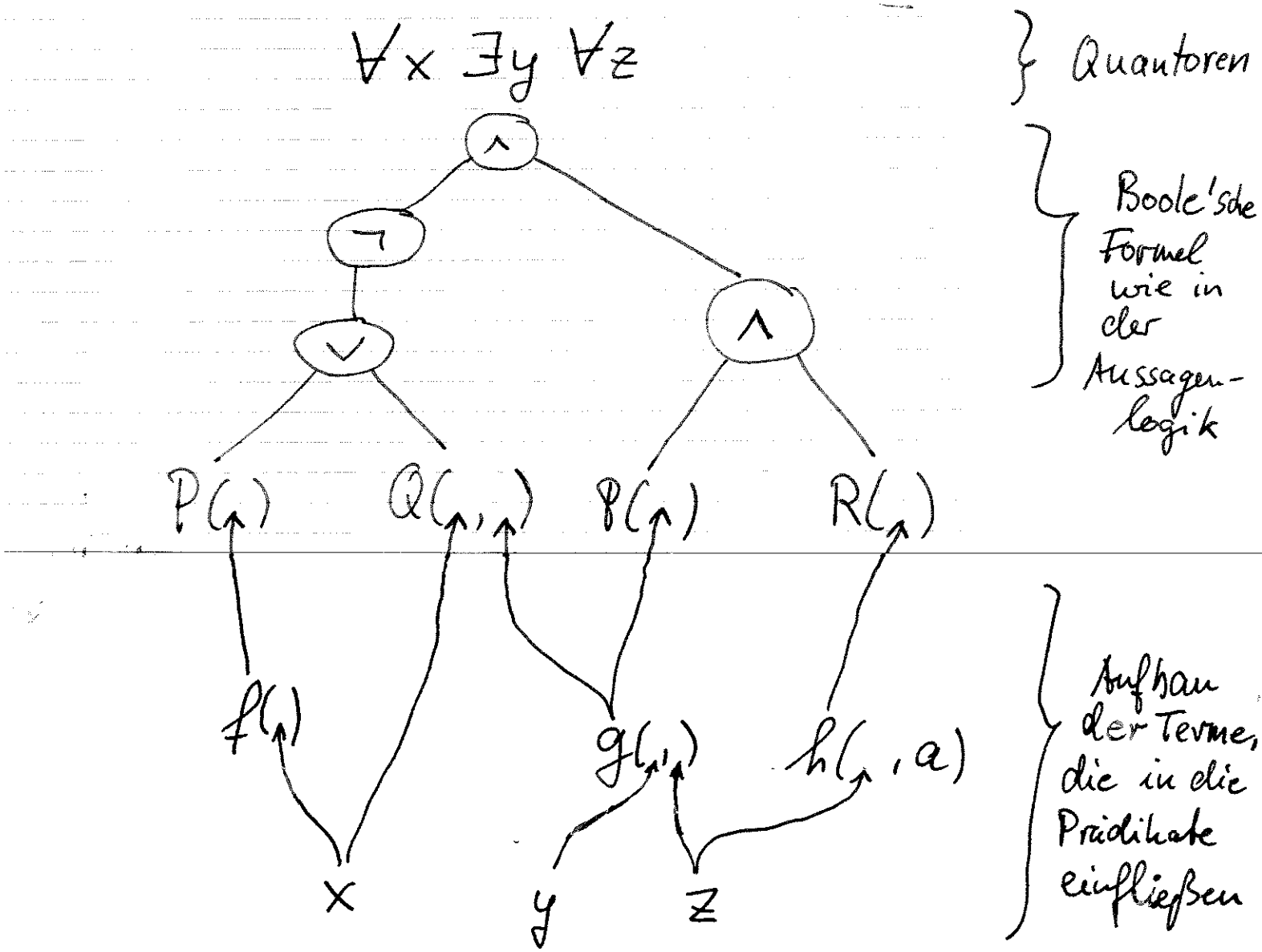
Welchen Wertebereich die Variablenwerte von x, y, z durchlaufen, ebenso welche Funktion f, g, h dies konkret sind, bleibt bei der Angabe einer prädikatenlogischen Formel zunächst offen.

In der Aussagenlogik wird durch Angabe einer Belegung $\mathcal{A}$ der Wahrheitswert der verschiedenen Aussagenvariablen A, B, C festgelegt. Das Pendant zur Belegung heißt in der Prädikatenlogik <u>Struktur</u>. Bei der Angabe einer Struktur $\mathcal{A}$ muss festgelegt werden:

- die Grundmenge U, aus der die Variablen und Funktionen ihre Werte beziehen, z.B. $U = \mathbb{N}$.
- Welche konkrete Funktion den vorkommenden Symbolen f, g, h (über der Grundmenge U) zugeordnet werden sollen, z.B. Additionsfunktion
- Welche konkreten Prädikate (also Relationen) den vorkommenden Symbolen P, Q, R zugeordnet werden sollen, z.B. Primzahleigenschaft

$\mathcal{A} = (U;$ konkrete Funktionen über $U;$ konkrete Prädikate über $U)$

Grundsätzlicher Aufbau einer prädikatenlogischen Formel:



Dieselbe Formel in linearer Schreibweise:

$$\forall x \exists y \forall z \neg\Big(P(f(x)) \vee Q(x, g(y,z))\Big) \wedge \Big(P(g(y,z)) \wedge R(h(z,a))\Big)$$

Eine mögliche Struktur $\mathcal{A}$ für diese Formel:

$\mathcal{A} = (\mathbb{N}$; f ist Nachfolgerfkt, g ist Additionsfkt, h ist Multiplikation, a ist Konstante 10, P... Primzahleigenschaft, Q... Kleiner-Relation, R bedeutet: Wert ist > 5)

In der Formel kommen folgende Symbole vor:

Prädikate: P ... 1-stellig
Q 2-stellig
R ... 1-stellig

Funktionen: f ... 1-stellig
g ... 2-stellig
h ... 2-stellig
a ... 0-stellig (Konstante)

Kurz: $\Sigma = \{P/1, Q/2, R/1, f/1, g/2, h/2, a/0\}$

Eine solche Angabe heißt auch Signatur (oder Vokabular) (der Formel F, oder auch: der Struktur $\mathcal{A}$)

Formale Definition von Formeln und Termen der Prädikatenlogik:

- Jede Variable x ist ein Term
- Wenn f ein k-stelliges Funktionszeichen ist, ~~dann~~ und wenn $t_1, \ldots, t_k$ bereits Terme sind, dann ist $f(t_1, \ldots, t_k)$ ein Term.

- Wenn $t_1, \ldots, t_k$ Terme sind und P ein k-stelliges Prädikatsymbol, dann ist $P(t_1, \ldots, t_k)$ eine Formel.
- Wenn F eine Formel ist, dann ist auch $\neg F$ eine Formel
- Wenn F und G Formeln sind, dann ist auch $(F \wedge G)$ ~~als~~ auch $(F \vee G)$ eine Formel.
- Wenn x eine Variable ist und F eine Formel, dann ist auch $\forall x F$ bzw. $\exists x F$ eine Formel.

  (Zusatz: Falls x in einer derartigen Teil-Formel $\exists x F$ bzw. $\forall x F$ vorkommt, so heißt dieses Vorkommen gebunden. Die anderen Vorkommen von x heißen frei.)

Wenn eine Struktur $\mathcal{A} = (U, f_i^{\mathcal{A}}, P_j^{\mathcal{A}})$ gegeben ist:

f_i: konkrete Fkt auf U; P_j: konkretes Prädikat auf U

dann kann man $\mathcal{A}$ auf eine Formel anwenden, —sofern sie keine freien Variablen enthält—

die dadurch einen Wahrheitswert bekommt,

entweder $\mathcal{A}(F) = 1$ bzw. $\mathcal{A} \models F$

oder $\mathcal{A}(F) = 0$ bzw. $\mathcal{A} \not\models F$.

(Vorausgesetzt die Signatur von $\mathcal{A}$ und die Signatur von F stimmen überein, d.h. in F kommen genau diejenigen Funktions- und Prädikat-Symbole vor, die in $\mathcal{A}$ konkret definiert werden.)

Def: F (ohne freie Variablen) heißt erfüllbar, wenn es eine Struktur $\mathcal{A}$ gibt mit $\mathcal{A} \models F$. Andernfalls: unerfüllbar.

Eine unerfüllbare prädl. log. Formel ist z.B. Folgende:

$$\forall x\, P(x, f(x)) \wedge \exists t\, \forall y\, \neg P(t, y)$$

Aufgabe: Formuliere die Aussage „die Funktion f ist stetig an der Stelle x" mit Hilfe von Quantoren und analysiere, welche Funktionen und Prädikate in der Formel vorkommen, sowie welche Variablen-Vorkommen frei bzw. gebunden sind.

Antwort:

$$\forall \varepsilon > 0 \; \exists \delta > 0 \; \forall y \; (|x-y| < \delta \rightarrow |f(x) - f(y)| < \varepsilon)$$

frei geb geb frei geb. geb.

Vorkommende Funktionen: f 1-stellig, noch uninterpretiert

$-$ Subtraktion, 2-stellig

$| \; |$ Betrag, 1-stellig

Relationen: $>$ Größer-Relation

$<$ Kleiner-Relation

Die Schreibweise: $\forall \varepsilon > 0 \; F$ bedeutet:

$$\forall \varepsilon \; (\varepsilon > 0 \rightarrow F)$$

$\exists \delta > 0 \; F$ bedeutet: $\exists \delta \; (\delta > 0 \wedge F)$

Die Formel kann erst dann mit einem Wahrheitswert versehen werden, wenn f und x spezifiziert werden.

Frage (Formulierung von prädikatenlog. Formeln)

Gegeben sei folgende Struktur:

$$\mathcal{A} = (M; Gl, Ki, Gr, Fl)$$

wobei M = Menge aller Drachen

$Gl(x)$ bedeutet: x ist glücklich

$Gr(x)$ ——#—— : x ist grün

$Fl(x)$ ——#—— : x kann fliegen

$Ki(x,y)$ ——#—— : x ist Kind von y

Formalisieren Sie:

(a) „Jeder Drache ist glücklich, wenn alle seine Kinder fliegen können."

$$\forall x \left(\forall y \left(Ki(y,x) \rightarrow Fl(y) \right) \rightarrow Gl(x) \right)$$

(b) „Grüne Drachen können fliegen."

$$\forall z \left(Gr(z) \rightarrow Fl(z) \right)$$

(c) „Ein Drache ist grün, wenn er Kind mindestens einen grünen Drachens ist."

$$\forall u \left(\exists v \left(Ki(u,v) \wedge Gr(v) \right) \rightarrow Gr(u) \right)$$

Frage:
Welche Variablen-Vorkommen in F sind frei und welche sind gebunden?

$$F = \forall x\, P(x,y) \wedge \exists y\, (Q(x,y,z) \vee \neg R(y))$$

x: geb, y: frei; y: frei, y: geb., z: frei, y: geb.

Bemerkung: Die Auswertung einer Formel ändert sich nicht, wenn man gebundene Variablen systematisch umbenennt. Auf diese Weise kann man dafür sorgen, dass jede Variable entweder gebunden oder frei vorkommt (aber nicht beides) und keine 2 verschiedenen Quantoren sich auf denselben Variablennamen beziehen. (Formel bereinigen)

Beispiel:

$$\forall x\, P(x,y) \vee \exists x (Q(x) \wedge \neg P(y,x)) \vee R(x)$$

$$= \forall u\, P(u,y) \vee \exists v (Q(v) \wedge \neg P(y,v)) \vee R(x)$$

u: frei; y: frei; x: frei

Eine Aussage ist eine Formel ohne freie Variablen. Im Folgenden wollen wir nur Aussagen betrachten. Freie Variablen sind meist implizit all-quantifiziert.

Bsp: „$x+y=y+x$" bedeutet dasselbe wie

$$\forall x \forall y \, (x+y=y+x)$$

Anwenden einer Struktur $\mathcal{A}$ auf eine Formel:

Gegeben sei eine Struktur $\mathcal{A}$ und eine Formel F ohne freie Variablen. Die Signatur von $\mathcal{A}$ stimmt mit der Signatur von F überein (d.h. die in F vorkommenden Funktions- und Prädikatsymbole f_i, P_j werden in $\mathcal{A}$ durch konkrete Funktionen und Prädikate $f_i^{\mathcal{A}}, P_j^{\mathcal{A}}$ auf einer Grundmenge M interpretiert).

Wir schreiben $\mathcal{A}(F)=1$ oder $\mathcal{A} \models F$ („$\mathcal{A}$ erfüllt F"), falls Folgendes gilt:

oder

„$\mathcal{A}$ ist Modell für F"

- Falls $F = \neg G$ so gilt $\mathcal{A} \models F$ genau dann wenn $\mathcal{A} \not\models G$.
- Falls $F = (G \wedge H)$, dann ist $\mathcal{A} \models F$ genau dann wenn $\mathcal{A} \models G$ und $\mathcal{A} \models H$ gilt.
- Falls $F = (G \vee H)$, dann ist $\mathcal{A} \models F$ genau dann wenn $\mathcal{A} \models G$ oder $\mathcal{A} \models H$.
- Falls $F = \forall x\, G$, dann ist $\mathcal{A} \models F$ genau dann wenn für alle Elemente $m \in M$ (die Grundmenge von $\mathcal{A}$) gilt, dass $\underbrace{\mathcal{A}[x/m]}_{\text{abgeänderte Struktur } \mathcal{A}:} \models G$.

  bei der Auswertung soll die freie Variable x in G durch die Konstante m interpretiert werden (kurz: $x^{\mathcal{A}} = m$)
- Falls $F = \exists x\, G$, dann ist $\mathcal{A} \models F$ genau dann wenn ein $m \in M$ existiert mit $\mathcal{A}[x/m] \models G$

Die Struktur $\mathcal{A}$ ~~gibt~~ weist den Funktions- und Symbolen konkrete Funktionen zu, sowie allen Variablen durch die $[x/m]$-Festlegungen. Daher hat jeder Term t unter der Struktur $\mathcal{A}$ einen festen Wert $t^{\mathcal{A}} \in M$.

- Falls F die Form hat $P(t_1, \ldots, t_k)$, so existieren – wie gesagt – bereits feste Werte der Terme: $t_1^{\mathcal{A}}, \ldots, t_k^{\mathcal{A}}$. An diesen Werten kann das Prädikat ausgewertet werden und liefert den Wahrheitswert $P^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_k^{\mathcal{A}})$. Falls dieser Wahrheitswert $= 1$ ist, so gilt $\mathcal{A} \models F$, sonst $\mathcal{A} \not\models F$.

Beispiel: $F = \forall x \forall y \left(P(x,y) \rightarrow \exists z \left(P(x,z) \wedge P(z,y) \right) \right)$

Eine mögliche Struktur ist $\mathcal{A} = (\mathbb{R};$ $P^{\mathcal{A}}$ ist die kleiner-Relation auf $\mathbb{R})$

Unter dieser Interpretation, in moderner Schreibweise geschrieben, lautet die Formel:

$$\forall x \in \mathbb{R} \; \forall y \in \mathbb{R} \left(x < y \rightarrow \exists \underset{\in \mathbb{R}}{z} \left(x < z \wedge z < y \right) \right)$$

d.h: Zwischen je zwei reellen Zahlen x, y wobei $x < y$ gibt es echt dazwischen eine weitere reelle Zahl z.

Diese Aussage ist wahr, also $\mathcal{A} \models F$.

Betrachten wir die Struktur $\mathcal{B} = (\mathbb{N}, P^{\mathcal{A}}$ ist kleiner-Relation auf $\mathbb{N})$

Unter dieser Struktur gilt die Formel F nicht, also $\mathcal{B} \not\models F$.

Definition: Formel F heißt erfüllbar, falls es eine Struktur $\mathcal{A}$ gibt mit $\mathcal{A} \models F$, ansonsten unerfüllbar.

Formel F heißt Tautologie (oder allgemeingültig) falls für alle (passenden) Strukturen $\mathcal{A}$ gilt: $\mathcal{A} \models F$.

Bsp für eine Tautologie:

$$\exists y \forall x P(x,y) \rightarrow \forall u \exists v P(u,v)$$

Problem: Je nach gegebener Struktur kann die Grundmenge endlich sein, unendlich (sogar überabzählbar so wie $\mathbb{R}$) sein.
Wenn die Grundmenge endlich ist, so kann die Feststellung ob $\mathcal{A} \models F$ algorithmisch in endlicher Zeit durchgeführt werden.

Aufgabe: In der monadischen Prädikatenlogik dürfen die Formeln keine Funktionssymbole enthalten und die Prädikatsymbole müssen einstellig sein.

Man zeige: Wenn eine Formel der monadischen Prädikatenlogik erfüllbar ist, so gibt es mit den Prädikaten $P_1, \ldots, P_k$ bereits eine Struktur mit 2^k Elementen, die die Formel erfüllt. (Hieraus folgt, dass das Erfüllbarkeits/Gültigkeits-Problem der monadischen Prädikatenlogik entscheidbar ist.)

Lösung: Der Grundbereich einer Struktur $\mathcal{A}$, die die Formel erfüllt, kann in $\leq 2^k$ Äquivalenzklassen unterteilt werden.

$m \in M$ und $m' \in M$ sind dabei äquivalent, wenn $(P_1^{\mathcal{A}}(m), \ldots, P_k^{\mathcal{A}}(m)) = (P_1^{\mathcal{A}}(m'), \ldots, P_k^{\mathcal{A}}(m'))$

Seien $[m_1], \ldots, [m_\ell]$, $\ell \leq 2^k$, die hierbei entstehenden Äquivalenzklassen. Dann erfüllt auch

folgende Struktur $\mathcal{B} = (\{[m_1], \ldots, [m_e]\}, P_i^{\mathcal{B}})$

die Formel, wobei

$P_i^{\mathcal{B}}([m_j])$ gilt gdw. $P_i^{\mathcal{A}}(m_j)$ gilt.

Also: $\mathcal{B} \models F$.

Viele Konzepte und Sätze aus der Aussagenlogik übertragen sich in die Prädikatenlogik, z.B:

F ist Tautologie gdw. $\neg F$ ist unerfüllbar.

Dasselbe gilt für Äquivalenzumformungen wie Assoziativgesetz, Distributivgesetz, de Morgan, etc.

Eine prädikatenlog. Formel ohne Quantoren und damit ohne Variablen kann wie eine aussagenlog. Formel verstanden werden (trotz Vorkommen von Funktionssymbolen und damit Termen).

Beispiel: $F = \underbrace{Q(f(a))}_{=:A} \wedge (\neg \underbrace{P(f(f(b)), c)}_{=:B} \vee \underbrace{R(c)}_{=:C})$

$$F = A \wedge (\neg B \vee C)$$

Erfüllbarkeit/Tautologie-Eigenschaft identisch.

Wir behandeln hier die engere Prädikatenlogik. Bei der allgemeinen Prädikatenlogik wird auch das Gleichheitszeichen zugelassen, z.B.

$$\forall x \forall y \; (x = y \rightarrow f(x) = f(y))$$

In dieser Vorlesung nur Prädikatlogik der 1. Stufe d.h. die Quantoren beziehen sich auf 1. stufige Objekte, die Variablen, deren Werte einzelne Elemente der Grundmenge sind.

Bei der Prädikatlogik der 2. Stufe dürfen sich Quantoren auch auf Prädikate und Funktionen beziehen, z.B. $\forall P \exists f \forall x \; P(f(x), x)$

Aussagen, die mit Hilfe der Prädikatlogik 1. Stufe formuliert werden können, heißen elementar.

Äquivalenzumformungen für Formeln mit Quantoren:

$$\neg \exists x \, F = \forall x \neg F$$
$$\neg \forall x \, F = \exists x \neg F$$

verallg. de Morgan

$$\forall x \, F \wedge G = \forall x \, (F \wedge G)$$
$$\forall x \, F \vee G = \forall x \, (F \vee G)$$
$$\exists x \, F \wedge G = \exists x \, (F \wedge G)$$
$$\exists x \, F \vee G = \exists x \, (F \vee G)$$

x darf in G nicht frei auftreten

//

$$\forall x F \wedge \forall x G = \forall x (F \wedge G)$$

$$\exists x F \vee \exists x G = \exists x (F \vee G)$$

Falsch ist:

$$\forall x F \vee \forall x G = \forall x (F \vee G)$$

$$\exists x F \wedge \exists x G = \exists x (F \wedge G)$$

Ziel dieser Umformungen: Herstellen der

Pränex-Normalform:

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \underbrace{(\dots)}_{\text{Restformel, ohne Quantoren}} \quad Q_i \in \{\exists, \forall\}$$

Korrekt:

$$\forall x F \vee \forall x G = \forall x \forall y (F \vee G[x/y])$$

$$\exists x F \wedge \exists x G = \exists x \exists y (F \wedge G[x/y])$$

Gegeben: eine prädikatenlog. Formel. Diese soll aufbereitet werden, um dann ggf. „automatisch" ihre Unerfüllbarkeit bzw. Tautologieeigenschaft feststellen zu können.

Es zeigt sich, dass nur die Eigenschaft unerfüllbar zu sein, am Ende bestätigt werden kann.
Möchte man bestätigen, dass F eine Tautologie ist, so starte man das Verfahren stattdessen mit $\neg F$ und überprüfe $\neg F$ auf Unerfüllbarkeit.

---

1. Schritt: Formel bereinigen durch gebundene Umbenennung. (Keine 2 Quantoren sollten sich auf denselben Variablennamen beziehen)
Freie Variablen sollten keine vorkommen; alle Variablen sollen gebunden sein.

2. Schritt: Herstellen der Pränexform, d.h. alle Quantoren sollen vorne stehen. Dies erreicht man durch Anwenden der (verallgemeinerten) de Morgan-Regeln: $\neg \forall x F = \exists x \neg F$ und $\neg \exists x F = \forall x \neg F$.

Ferner durch Regeln wie $\forall x F \wedge G \equiv \forall x (F \wedge G)$

Die Formel hat jetzt die Form

$$F = \underbrace{Q_1 x_1 Q_2 x_2 \dots Q_k x_k}_{\text{Quantorenpräfix}} \underbrace{G(x_1, \dots, x_k)}_{\text{die sog. Matrix der Formel } F.}$$

$Q_i \in \{\exists, \forall\}$

Abkürzung: BPF ... bereinigt und in Pränexform

evtl. 3. Schritt: Die Matrix der Formel F, also G, kann mit den Methoden der Aussagenlogik (Anwenden von de Morgan und Distributivgesetz) in konjunktive Normalform gebracht werden. Die KNF benötigen wir, sofern später (prädikatenlogische) Resolution angewandt werden soll.

(Es gibt alternative Verfahren, die nicht auf Resolution beruhen, dann wird KNF nicht benötigt.)

Eine Notation: Sei F eine Formel, die Variable x komme in F frei vor, t sei ein Term. Dann bezeichnet $F[x/t]$ die neue Formel, die entsteht, indem man jedes solche freie Vorkommen von x in F durch t ersetzt.

Beispiel: Sei $F = P(x,y) \wedge Q(f(x))$

Dann ist $F[x/g(a)] = P(g(a),y) \wedge Q(f(g(a)))$

Solche Substitutionen können kombiniert werden, z.B.

$$sub = [x/t_1][y/t_2]$$

Zuerst werden die freien Vorkommen von x durch t_1 ersetzt, dann die freien Vorkommen von y durch t_2.

Bemerkung: Beachte den Unterschied zwischen

$\underbrace{\mathcal{A}[x/m]}$ und $\underbrace{F[x/t]}$

die Struktur $\mathcal{A}$ wird (semantisch) so abgeändert, dass beim Auswerten von x der Wert $m \in M$ eingesetzt wird

die Formel F wird (syntaktisch) so abgeändert, dass die Vorkommen von x durch t ersetzt werden.

Ein Zusammenhang zwischen beiden Notationen stellt das Überführungslemma dar:

Sei F eine Formel mit der freien Variablen x. Sei $t^{\mathcal{A}} \in M$ der Wert, den der Term t bei Auswertung mittels der Struktur $\mathcal{A}$ erhält. (M ist die Grundmenge von $\mathcal{A}$.) Dann gilt:

$$\mathcal{A} \models F[x/t] \quad \text{gdw.} \quad \mathcal{A}[x/t^{\mathcal{A}}] \models F$$

Dies wird gelegentlich in Beweisen als technisches Hilfsmittel benötigt.

Ein wichtiger weiterer Umformungsschritt ist das Herstellen der Skolemform. Hierdurch werden alle Existenzquantoren einer Formel F eliminiert; es verbleibt danach eine neue Formel F', die nur noch Allquantoren enthält. Die neue Formel ist allerdings nicht im strengen Sinne äquivalent zur alten, sondern es gilt:

F ist erfüllbar gdw. F' ist erfüllbar.

(sog. Erfüllbarkeitsäquivalenz)

Dies ist für unsere Zwecke jedoch gut genug, denn der Sinn all dieser Umformungsschritte besteht darin, die ja/nein-Frage zu beantworten, ob die (ursprüngliche) Formel un/erfüllbar ist.

Herstellen der Skolemform („Skolemisieren"):

- Finde die erste Stelle in F, wo ein $\exists$-Quantor auftaucht:

$$F = \forall x_1 \dots \forall x_n \exists x_{n+1} G \overset{\text{Umformung}}{\rightsquigarrow} \forall x_1 \dots \forall x_n G[x_{n+1} / f(x_1, \dots, x_n)]$$

$(n \geq 0)$

- Diesen Schritt solange wiederholen, bis alle Existenzquantoren eliminiert sind.

Hierbei ist f ein neues (bisher in F nicht vorkommendes) n-stelliges Funktionssymbol.

Beispiel: $F = \exists x \, \forall y \, \exists u \, \forall v \, \exists w \; P(x,y,u,v,w)$

$\rightsquigarrow \forall y \, \exists u \, \forall v \, \exists w \; P(a,y,u,v,w)$

$\rightsquigarrow \forall y \, \forall v \, \exists w \; P(a,y,f(y),v,w)$

$\rightsquigarrow \forall y \, \forall v \; P(a,y,f(y),v,g(y,v))$

Hierbei sind:

a ... 0-stelliges Fkt.symbol (Konstante)

f ... 1-stelliges Fkt. Symbol

g ... 2-stelliges Fkt. Symbol

neu eingeführte sog. Skolemfunktionen.

Intuitiv: Betrachte den Übergang

$$\forall x \, \exists y \; P(x,y) \rightsquigarrow \forall x \; P(x,f(x))$$

Links: für jedes x gibt es ein y mit $P(x,y)$.

Rechts: „Definiere" f so dass für jedes x $f(x)$ dann irgendso ein existierendes y sein soll.

Mathematisches Problem: Ist dies eine sinnvolle Definition der Funktion f?

Aus der Tatsache der Existenz eines y mit $P(x,y)$ ergibt sich, dass man ein solches y auswählen kann und man $f(x)=y$ per Definition setzen kann.
Diese Vorgehensweise erfordert das sog. Auswahlaxiom.

Den Satz:

> F ist erfüllbar gdw. die Skolemform F' von F ist erfüllbar

kann man im Spezialfall $F = \forall x \exists y\, P(x,y) \rightsquigarrow F' = \forall x\, P(x,f(x))$ wie folgt beweisen:

Wenn F' erfüllbar ist durch $\mathcal{A}$ mit Grundbereich M,
Dann gilt für alle $m \in M$: $\mathcal{A}[x/m] \models P(x,f(x))$

Nun Überführungslemma: für alle $m \in M$ ist

$$\mathcal{A}[x/m][y/m'] \models P(x,y)$$

wobei $m' = f^{\mathcal{A}}(m)$. Es folgt: für alle $m \in M$ existiert ein m' mit $\mathcal{A}[x/m][y/m'] \models P(x,y)$

Das bedeutet: $\mathcal{A} \models \forall x \exists y\, P(x,y)$.

Umgekehrt:

Sei $\mathcal{A}$ ein Modell für $\forall x \exists y\, P(x,y)$, also $\mathcal{A} \models \forall x \exists y\, P(x,y)$. Das bedeutet: für alle $m \in M$ existiert ein $m' \in M$ mit $\mathcal{A}[x/m][y/m'] \models P(x,y)$

Definiere erweiterte Struktur $\mathcal{A}'$ mit einem zusätzlichen Funktionssymbol f (es findet also eine Erweiterung der Signatur von $\mathcal{A}$ nach $\mathcal{A}'$ statt.)

Mittels <u>Auswahlaxiom</u> definieren wir für jedes $m \in M$

$$f(m) = m'$$

wobei wir hierzu ein m' wie oben <u>auswählen</u>.

Somit: für alle $m \in M$,
$$\mathcal{A}'[x/m][y/f^{\mathcal{A}'}(m)] \models P(x,y)$$

Überführungslemma: für alle $m \in M$,

$$\mathcal{A}'[x/m] \models P(x, f(x)).$$

Also: $\mathcal{A}' \models \forall x\, P(x, f(x))$. □

Zwischenergebnis: Nach all diesen Umformungsschritten haben wir eine prädikatenlogische Formel, bei der alle Variablen all-quantifiziert sind. Ferner sind neue Skolemfunktion hinzugekommen. Wir lassen die Allquantoren jetzt weg.

Beispiel: $(P(x, f(x)) \vee \neg Q(a)) \wedge (P(g(z), u) \vee R(v))$
(in KNF)

Beispiel für eine unerfüllbare Formel:

$$P(x) \wedge \neg P(f(y))$$

Wie zeigt man dies?

Herbrand-Theorie:

Die Grundmenge M einer Struktur $\mathcal{A}$ kann im Prinzip jede beliebige Menge sein — auch überabzählbar: $M = \mathbb{R}$. Die Elemente von M müssen nicht notwendigerweise Zahlen sein, z.B.

$M = \{\diamond, \#, *, \natural, @\}$ etc. Wichtig ist die Art der Relationen/Fkten auf M.

Herbrands Idee war, die Grundmenge M rein syntaktisch aus den Bestandteilen der Formel F aufzubauen.

Beispiel: F enthalte a ... 0-stellig
f ... 1-stellig

Dann könnte man folgende Grundmenge $M = D(F)$ bilden:

$$\underbrace{D(F)}_{\substack{\text{sog.} \\ \text{Herbrand-Universum}}} = \{ a, f(a), f(f(a)), f(f(f(a))), \dots \}$$

Stattdessen könnte man auch schreiben:

$$M = \{ 0 \xrightarrow{f} 1 \xrightarrow{f} 2 \xrightarrow{f} 3, \dots \}$$

Was ist die natürlichste Art und Weise, auf der Grundmenge $D(F)$, die Funktion f zu interpretieren?

$$f^A: \begin{cases} a \longmapsto f(a) \\ f(a) \longmapsto f(f(a)) \\ f(f(a)) \longmapsto f(f(f(a))) \\ \vdots \\ \subseteq D(F) \end{cases} \qquad \subseteq D(F)$$

Eine Herbrand-Struktur ist „Normalform" für Strukturen:

$$\mathcal{A} = (D(F), f_i^{\mathcal{A}}, P_j^{\mathcal{A}})$$

- $D(F)$: Grundmenge
- $f_i^{\mathcal{A}}$: sind fest vorgegeben, so wie oben
- $P_j^{\mathcal{A}}$: frei definierbar – bezogen auf die Grundmenge $D(F)$

Es gilt:

> F (in Skolemform) ist erfüllbar
> gdw.
> F ist durch eine Herbrand-Struktur erfüllbar.

Korollar:

Satz von Löwenheim-Skolem:

Jede erfüllbare präd. log Formel besitzt bereits ein abzählbares Modell.

Herbrand-Expansion eines (Skolem-) Formel F:

$$E(F) = \{ F[x_1/t_1] \cdots [x_n/t_n] \mid t_1, \ldots, t_n \in D(F) \}$$

Aufgabe: Gegeben ist ein Ausschnitt aus einem Prolog-Programm, das einige Verwandschafts-beziehungen aufzeigt:

ist-vater (anton, maria).

geschwister (maria, fritz).

ist-vater (horst, anton).

ist-grossvater (X, Y) ← ist-vater (X, Z), ist-vater (Z, Y).

Konvention in Prolog: Variablen gross schreiben, Prädikate und Funktionen klein schreiben. Jede Zeile entspricht einer Klausel. (Eine Klausel der Form $f \leftarrow g, h.$ entspricht dabei $f \vee \neg g \vee \neg h$).

Frage: Gib das Herbrand-Universum dieses Prolog-Programms an.

Antwort: Es kommen nur 0-stellige Funktionszeichen, also Konstanten vor. Diese bilden daher D(F):

$D(F) = \{anton, maria, fritz, horst\}$

Bemerkung: Diese einfache Version der Prädikatenlogik, in der keine ineinander verschachtelten Funktionssymbole vorkommen, nur Konstanten, aber auch Variablen, heißt Datalog. Es ist eine Art Datenbank-Sprache. Sie erlaubt gewisse deduktive Abfragen wie z.B. „Wer ist Großvater von Maria?".

Bemerkung: In manchen Formeln kommen keine Konstanten vor, wohl aber (≥ 1)-stellige Funktionssymbole. Um in einem solchen Fall sinnvoll $D(F)$ definieren zu können, fügt man „künstlich" eine Konstante, z.B. a, zur Signatur von F hinzu und bildet dann $D(F)$ in gewohnter Weise.

Beispiele:

- F enthält a, b, c $\Rightarrow$ $D(F) = \{a, b, c\}$
  (0-stellig)

- F enthält a, b (0-stellig)
  und f (1-stellig) $\Rightarrow$ $D(F) = \{a, f(a), f(f(a)), \ldots$
  $b, f(b), f(f(b)), \ldots\}$

- F enthält f (1-stellig)
  und g (2-stellig), aber
  keine Konstanten $\Rightarrow$

$D(F) = \{a, f(a), g(a,a), f(f(a)), f(g(a,a)),$
$g(f(a),a), g(a,f(a)), \ldots\}$

künstlich hinzu gefügt

$D(F)$ ist immer abzählbar.

Beispiel: Formel $F = (\forall y \exists x P(y,x) \wedge \exists z \forall w \neg P(w,z)$

Diese Formel ist erfüllbar. Eine mögliche Struktur $\mathcal{A}$, die die Formel erfüllt (ein sog. Modell) ist

$\mathcal{A} = (\mathbb{N}; P^{\mathcal{A}})$ wobei $P^{\mathcal{A}}$ die Kleiner-Relation auf $\mathbb{N}$ ist. Dann „besagt" die Formel, das es für jedes $n \in \mathbb{N}$ eine größe natürliche Zahl m gibt (etwa $m = n+1$).

Ferner gibt es eine Zahl, die man für z einsetzen kann, nämlich die $0 \in \mathbb{N}$, die nicht größer als irgendein anderes Wort w aus $\mathbb{N}$ ist.

Wir bilden die Skolemform + Pränexform von F:

$$\rightsquigarrow F' = (\forall y\, P(y, f(y)) \wedge \forall w \neg P(w, a))$$

$$= \forall y \forall w\, (P(y, f(y)) \wedge \neg P(w, a))$$

Das Herbrand-Universum dieser Formel ist

$D(F') = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$

Das Herbrand-Universum ist die Grundmenge einer Herbrand-Struktur, in diesem Fall also

$$\mathcal{B} = (D(F'); P^B; f^B, a^B)$$

Die Interpretation der Funktionssymbole in einer Herbrand-Struktur ist ebenfalls fest vorgegeben: Sie folgt gewissermaßen der Definition des Herbrand-Universums. In diesem Fall ist $a^B = a$ und

$$f^B(\overbrace{f(f\cdots f(a)\cdots)}^{\in D(F)}) = \overbrace{f(f\cdots f(a)\cdots)}^{\in D(F)}$$

($f(f\cdots f(a)\cdots)$: k-mal; $f(f\cdots f(a)\cdots)$: $k+1$-mal)

Nur die Interpretation der Prädikatsymbole, hier P^B, kann noch frei gewählt werden. Wenn man sich an der Definition der obigen erfüllenden Struktur orientiert, so könnte man in $\mathcal{B}$ festlegen:

$$P^B = \{(\underbrace{f(f(\cdots f(a)\cdots)}_{k}, \underbrace{f(f(\cdots f(a)\cdots)}_{\ell}) \mid k < \ell\}$$

Damit ergibt sich dann ebenfalls eine erfüllende Struktur $\mathcal{B}$.

Dahinter steckt ein allgemeiner Satz.

Satz: Eine Formel F in Skolemform ist erfüllbar genau dann, wenn F durch eine Herbrand-Struktur erfüllt werden kann.

Beweis: ($\Leftarrow$) Das ist klar: Eine Herbrand-Struktur ist auch eine Struktur und wenn diese die Formel erfüllt, so zeigt dies, dass F erfüllbar ist.

($\Rightarrow$) Sei nun $\mathcal{A}$ eine beliebige Struktur ~~die~~ (deren Signatur zu F passt), die die Formel F erfüllt, also $\mathcal{A} \models F$.

Wir müssen nun eine Herbrand-Struktur $\mathcal{B}$ – wie bei dem Beispiel von oben – angeben.

$$\mathcal{B} = (D(F), f_i^{\mathcal{B}}, P_j^{\mathcal{B}})$$

Hierbei stehen $D(F)$ und die f_i^B bereits fest; sie werden durch die Signatur von F (insbesondere die vorkommenden Konstanten, Funktionssymbole in F) festgelegt.

Nur die P_j^B können noch frei gewählt werden. Um diese so zu definieren, dass eine F erfüllende Struktur herauskommt, orientieren wir uns an der Struktur $\mathcal{A}$, von der wir bereits wissen, dass sie erfüllend ist.

Sei P ein n-stelliges Prädikatsymbol, das in F vorkommt. Ferner seien $t_1, \ldots, t_n \in D(F)$ beliebige, variablenfreie Terme, so wie sie in der Definition von $D(F)$ vorgesehen sind.

Die Frage ist: soll $P^B(t_1, \ldots, t_n)$ als wahr oder als falsch definiert werden?

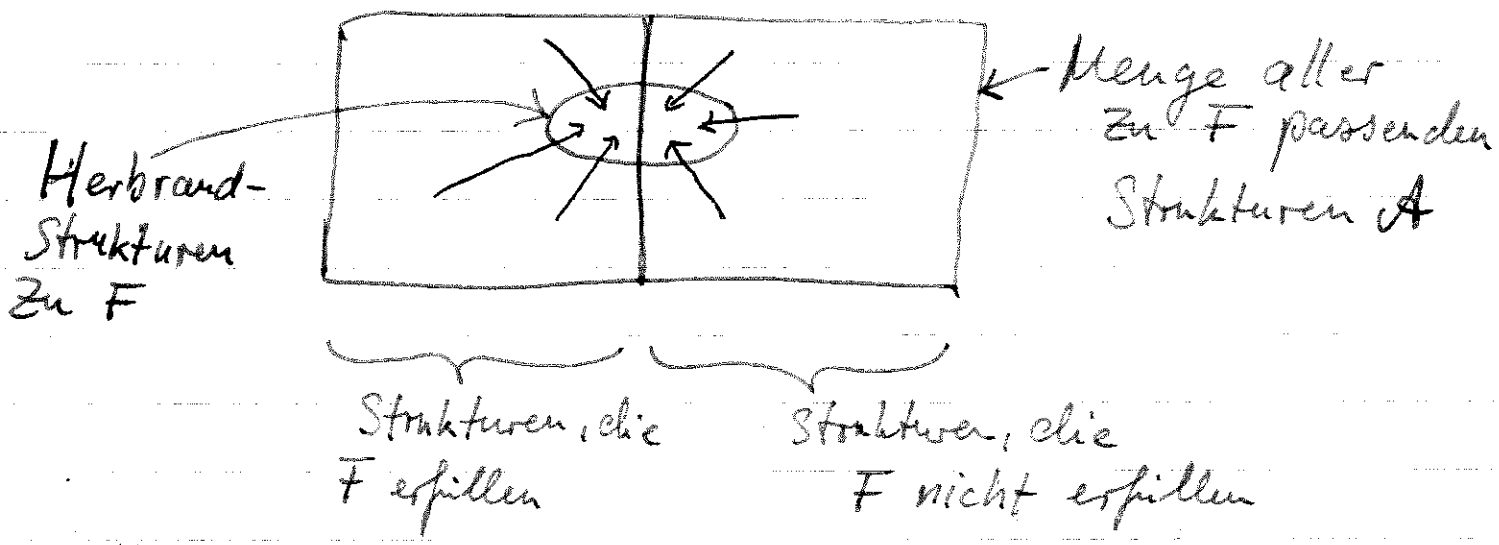
Seien $t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}}$ die Werte aus der Grundmenge M von $\mathcal{A}$, die diese Terme in der Struktur $\mathcal{A}$ annehmen. In der Struktur $\mathcal{A}$ hat $P^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$ einen bestimmten Wahrheitswert. Genauso definieren wir dann auch $P^{\mathcal{B}}$, kurz:

$$P^{\mathcal{B}}(t_1, \ldots, t_n) \quad :\text{gdw.} \quad P^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$$

soll wahr sein — ist wahr

Insofern gilt: Wenn man mittels $\mathcal{B}$ die Formel F auswertet, so ergibt sich in jedem einzelnen Zwischenschritt, für jede Teilformel von F, dass die Auswertung mittels $\mathcal{B}$ denselben Wahrheitswert ergibt wie die Auswertung mittels $\mathcal{A}$. (Im Skript, Seite 32, wird dies ausführlicher durch einen Induktionsbeweis gezeigt.) Daher folgt: $\mathcal{B} \models F$. □

Skizze: Gegeben eine Formel F, die erfüllbar ist (aber keine Tautologie)



Herbrand-Strukturen haben immer eine abzählbare Grundmenge (das Herbrand-Universum). Daher braucht man, um eine prädikatenlogische Formel zu erfüllen, keine überabzählbaren Strukturen, die z.B. $\mathbb{R}$ als Grundmenge haben. Man kommt immer mit $D(F)$ bzw. $\mathbb{N}$ aus. Umgekehrt heißt dies aber auch, dass viele Aussagen, z.B. aus der Analysis, in der Prädikatenlogik der 1. Stufe nicht adäquat ausgedrückt werden können.

Die Herbrand-Expansion $E(F)$

Sei $F = \forall x_1 \dots \forall x_n \, G(x_1, \dots, x_n)$ eine Formel ohne freie Variablen in Skolemform. Wie üblich schreiben wir die Allquantoren gar nicht mehr hin, sondern betrachten nur die Matrix der Formel F, nämlich $G(x_1, \dots, x_n)$.

$$E(F) := \{ G[x_1/t_1] \dots [x_n/t_n] \mid t_1, \dots, t_n \text{ sind beliebige Terme des Herbrand-Universums } D(F) \}$$

Bem: Die traditionelle Schreibweise für $G[x_1/t_1] \dots [x_n/t_n]$ ist $G(t_1, \dots, t_n)$

Beispiel: Sei $F = \forall x \forall y \, (P(x) \wedge \neg P(f(y)))$

Also ist $G(x,y) = (P(x) \wedge \neg P(f(y)))$.

Das Herbrand-Universum ist

$$D(F) = \{ a, f(a), f(f(a)), f(f(f(a))), \dots \}$$

↑ künstlich eingeführt

Wir bilden nun die Formeln in $E(F)$:

$$E(F) = \{ (P(a) \wedge \neg P(f(a))),$$
$$(P(a) \wedge \neg P(f(f(a)))),$$
$$(P(f(a)) \wedge \neg P(f(a))), \ldots \}$$

Die erste Formel entsteht durch: $[x/a][y/a]$,
die zweite Formel durch: $[x/a][y/f(a)]$,
die dritte Formel durch: $[x/f(a)][y/a]$, usw.

Es entsteht im Allgemeinen eine abzählbar unendliche Menge von Formeln, die keine Variablen mehr enthalten. Man kann diese Formeln als aussagenlog. Formeln verstehen.
Setzt man z.B. $A = P(a)$, $B = P(f(a))$, $C = P(f(f(a)))$
so erhält man:

$$E(F) = \{ (A \wedge \neg B),$$
$$(A \wedge \neg C),$$
$$(B \wedge \neg B), \ldots \}$$

$\square$

Wir werden beigen (Satz von Gödel-Herbrand-Skolem)

[F ist (im prädikatenlog. Sinne) erfüllbar] gdw. [die unendliche Formelmenge E(F) ist (im aussagenlog. Sinne) erfüllbar.]

Obige Beispielformel F ist unerfüllbar, denn E(F) enthält die Formel $(B_1 \wedge \neg B)$ die unerfüllbar ist.

(Bemerkung: Aufgrund des Endlichkeitssatzes (im Skript-Anhang) genügt es, um die Unerfüllbarkeit der unendlichen Formelmenge E(F) zu zeigen, eine endliche unerfüllbare Teilmenge von E(F) zu finden.)

Somit Neu-Formulierung des Satzes:

[F ist (im präd. log. Sinne) unerfüllbar] gdw. [Es gibt eine endliche Teilmenge von E(F), die (im aussagen. log. Sinne) unerfüllbar ist]

(Satz von Herbrand)

Beweis des Satzes von Gödel-Herbrand-Skolem:

F habe die Form $F = \forall x_1 \ldots \forall x_n \, G(x_1, \ldots, x_n)$

Wenn F erfüllbar ist, dann hat F oBdA ein Herbrand-Modell $\mathcal{A}$, also:

für alle $t_1, \ldots, t_n \in D(F)$: $\mathcal{A}[x_1/t_1] \ldots [x_n/t_n] \models G$

gdw. für alle $t_1, \ldots, t_n \in D(F)$: $\mathcal{A} \models \underbrace{G(t_1, \ldots, t_n)}_{\in E(F)}$

(Überführungslemma)

gdw. für alle Formeln $H \in E(F)$ gilt: $\mathcal{A} \models H$

gdw. $\mathcal{A}$ erfüllt $E(F)$. □

Algorithmische Umsetzung (nach Gilmore):

Eingabe $F = \forall x_1 \ldots \forall x_k \, G(x_1, \ldots, x_k)$

$n := 0$;

Repeat $n := n+1$;

until die ersten n Formeln von $E(F)$ sind unerfüllbar (feststellbar mit Wahrheitstafeln oder mit Resolution)

Ein Semi-Entsch. Verfahren: $F \longrightarrow$ [Gilmore] $\longrightarrow$ „unerfüllbar" ??

Was ist eine mathematische Theorie?

Gruppentheorie, Graphentheorie, Zahlentheorie, Analysis, Kombinatorik, ...

- Eine solche Theorie hat jeweils ihre individuelle Signatur (Beispiele: „Teilbarkeit", +, * in der Zahlentheorie; „Kantenrelation" in der Graphentheorie; „Integral, Ableitung, etc. in der Analysis, usw.)

- Eine Theorie wird definiert durch die Menge der Formeln, die in der Theorie gelten (die Menge ihrer „Sätze")

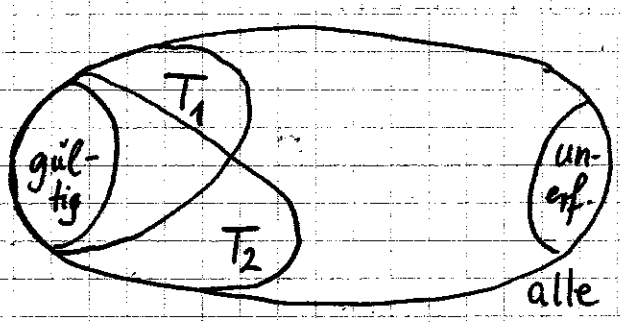
  Beispiele: $x \circ e = x$ in der Gruppentheorie

  $\int x\,dx = \frac{1}{2}x^2$ in der Analysis

Wenn $F_1, \ldots, F_n$ Sätze einer Theorie sind, dann auch alle Formeln G (über der betreffenden Signatur), die aus $F_1, \ldots, F_n$ folgen.

– Dies nimmt man als Definition: Eine Theorie T ist eine Menge von Formeln (über einer bestimmten Signatur), die unter Folgerbarkeit abgeschlossen ist, symbolisch: $Cons(T) = T$.

In jeder Theorie T sind daher alle gültigen Formeln enthalten.



alle präd. log. Formeln

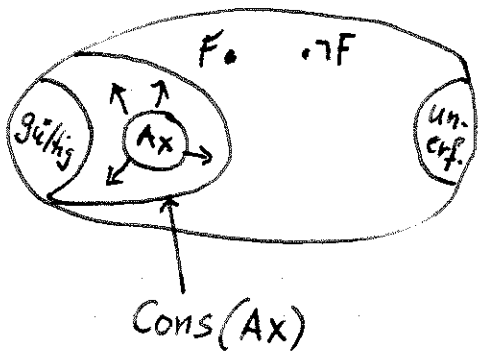
Die Menge der gültigen Formeln ist auch eine Theorie, Gültige Formeln = $Cons(\emptyset)$.

Es gibt 2 grundsätzlich unterschiedliche Arten eine Theorie zu definieren.

1. Die axiomatische Methode

Man gibt eine Menge von Axiomen Ax an (dies sollte eine entscheidbare Menge sein) und definiert: $T = Cons(Ax)$

Skizze:



Das Musterbeispiel ist die Gruppentheorie.

Die Signatur besteht aus:

e ... 0-stellige Fkt (das neutrale Elem.)

$\circ$ --- 2-stellige Fkt (die Gruppen-operation)

$=$ ... 2-stelliges Prädikat (Gleichheit)

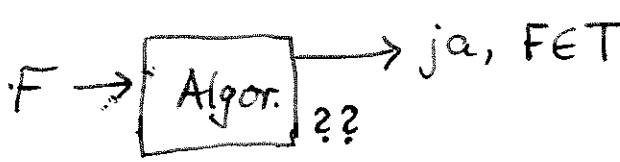
Die Axiome sind (Operationen in Infixnotation):

$$\forall x \ (x \circ e = x)$$

$$\forall x \exists y \ (x \circ y = e)$$

$$\forall x \forall y \forall z \ (x \circ (y \circ z) = (x \circ y) \circ z)$$

Alle Theorien, die axiomatisch definiert sind, sind rekursiv aufzählbar (was dasselbe bedeutet wie semi-entscheidbar):
→ Vorlesung Berechenbarkeit



Es gilt, dass F eine Folgerung der Axiome Ax ist, sofern $Ax \cup \{\neg F\}$ unerfüllbar ist. Da Ax entscheidbar und damit auch rekursiv aufzählbar ist, kann man systematisch die Formeln der Herbrand-Expansion $E = \{F_1, F_2, F_3, \ldots\}$ von $Ax \cup \{\neg F\}$ erzeugen und wie beim Algorithmus von Gilmore daraufhin untersuchen, ob eine unerfüllbare Teilmenge in E enthalten ist. Sofern das der Fall ist, stoppt der Algorithmus in endlicher Zeit, sonst nicht.

Eine Theorie kann (in seltenen Fällen) auch entscheidbar sein. Zum Beispiel, wenn die Signatur höchstens einstellige Prädikate (und keine Funktionssymbole) enthält.

2. Die modelltheoretische Methode

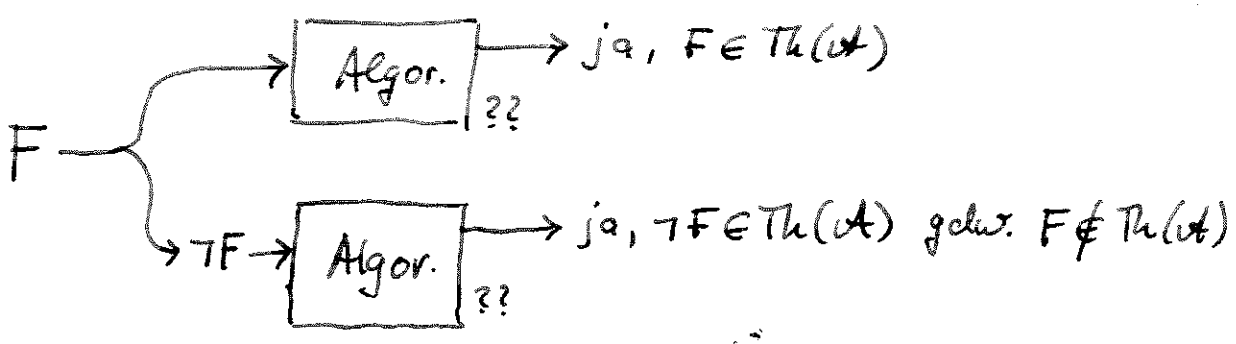
Man erhält eine Theorie, also eine unter Folgerbarkeit abgeschlossene Menge von Formeln, wenn man eine Struktur $\mathcal{A}$ vorgibt (über einer bestimmten Signatur), und dann die Menge aller unter $\mathcal{A}$ wahren Formeln bildet (über der betreffenden Signatur):

$$Th(\mathcal{A}) = \{ F \mid \mathcal{A} \models F \}$$ .... ist immer unter Cons() abgeschlossen

Für jede Formel F gilt genau eines:

entweder $F \in Th(\mathcal{A})$ oder $\neg F \in Th(\mathcal{A})$.

Daher gilt: Wenn eine modelltheoretisch definierte Theorie rekursiv aufzählbar (semi-entscheidbar) ist, dann ist sie sogar entscheidbar:



Das heißt auch: Wenn eine modelltheoretisch definierte Theorie $Th(\mathcal{A})$ auch durch ein Axiomensystem Ax beschrieben werden kann, also

$$Cons(Ax) = Th(\mathcal{A})$$

dann ist $Th(\mathcal{A})$ sogar entscheidbar.

Betrachte die Kontraposition:

Wenn gezeigt werden kann, dass $Th(\mathcal{A})$ nicht entscheidbar ist, dann kann $Th(\mathcal{A})$ auch nicht rekursiv aufzählbar (d.h. nicht axiomatisierbar) sein.

Jeder „Versuch" einer Axiomatisierung Ax von $Th(\mathcal{A})$ bleibt unvollständig, d.h:

$$Cons(Ax) \subsetneq Th(\mathcal{A})$$

Das Musterbeispiel, für das genau diese Aussagen zutreffen, ist die Zahlentheorie. Dies ist die modelltheoretische Theorie $Th(\mathcal{A})$ wobei

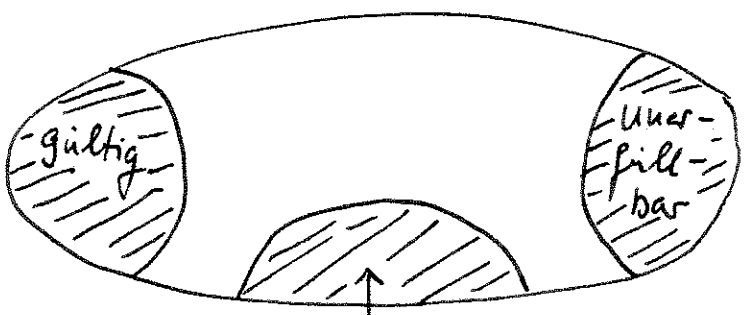
$\mathcal{A} = (\mathbb{N}; \underbrace{+, *, =}_{\text{Addition, Multiplikation, Gleichheit auf } \mathbb{N}})$

Gödel'scher Satz (1931):

Die Zahlentheorie, kurz: $Th(\mathbb{N})$, ist nicht entscheidbar und daher nicht axiomatisierbar.

(Für jeden „Versuch" Ax, die Zahlentheorie zu axiomatisieren, gibt es Formeln $F \in Th(\mathbb{N})$, die aus Ax nicht folgerbar sind: $F \notin Cons(Ax)$.)

Verschiedenartige Semi-Entscheidbarkeit:
in den schraffierten Bereichen kann Algorithmus stoppen und entsprechende Eigenschaft ausgeben.



erfüllbar, aber nicht gültig, mit endlichem Modell.

Herleitung der prädikatenlogischen Resolution

Beispiel: $F = \forall x \forall y \left((\neg P(x) \vee \neg P(f(a)) \vee Q(y)) \wedge P(y) \right.$

$\left. \wedge (\neg P(g(b,x)) \vee \neg Q(b)) \right)$

F^* enthält 3 Klauseln:

$\{\neg P(x), \neg P(f(a)), Q(y)\}$ $\{P(y)\}$ $\{\neg P(g(b,x)), \neg Q(b)\}$

Durch geeignete Substitution Erzeugen von Formeln in $E(F)$

$[x/f(a)]$ $[y/b]$ $[y/f(a)]$ $[y/g(b,a)]$ $[x/a]$

$E(F) \supseteq \{\neg P(f(a)), Q(b)\}$ $\{P(f(a))\}$ $\{P(g(b,a))\}$ $\{\neg P(g(b,a)), \neg Q(b)\}$

aussagenlogische Resolution

$\{Q(b)\}$ $\{\neg Q(b)\}$

$\square$

Problem: Die Substitutionen, die Elemente von $E(F)$ erzeugen, müssen „geeignet" (sozusagen: vorausschauend) gefunden werden, so dass danach ein Resolutionsbeweis ermöglicht wird.

Der Algorithmus von Gilmore sucht nach einer endlichen Teilmenge von $E(F)$, die unerfüllbar ist. Die Formeln (Klauseln) in $E(F)$ kommen dadurch zustande, dass jede Klausel in F^* in jeder denkbaren Weise durch Terme $t \in D(F)$ substituiert wird (sog. Grund-Substitutionen).

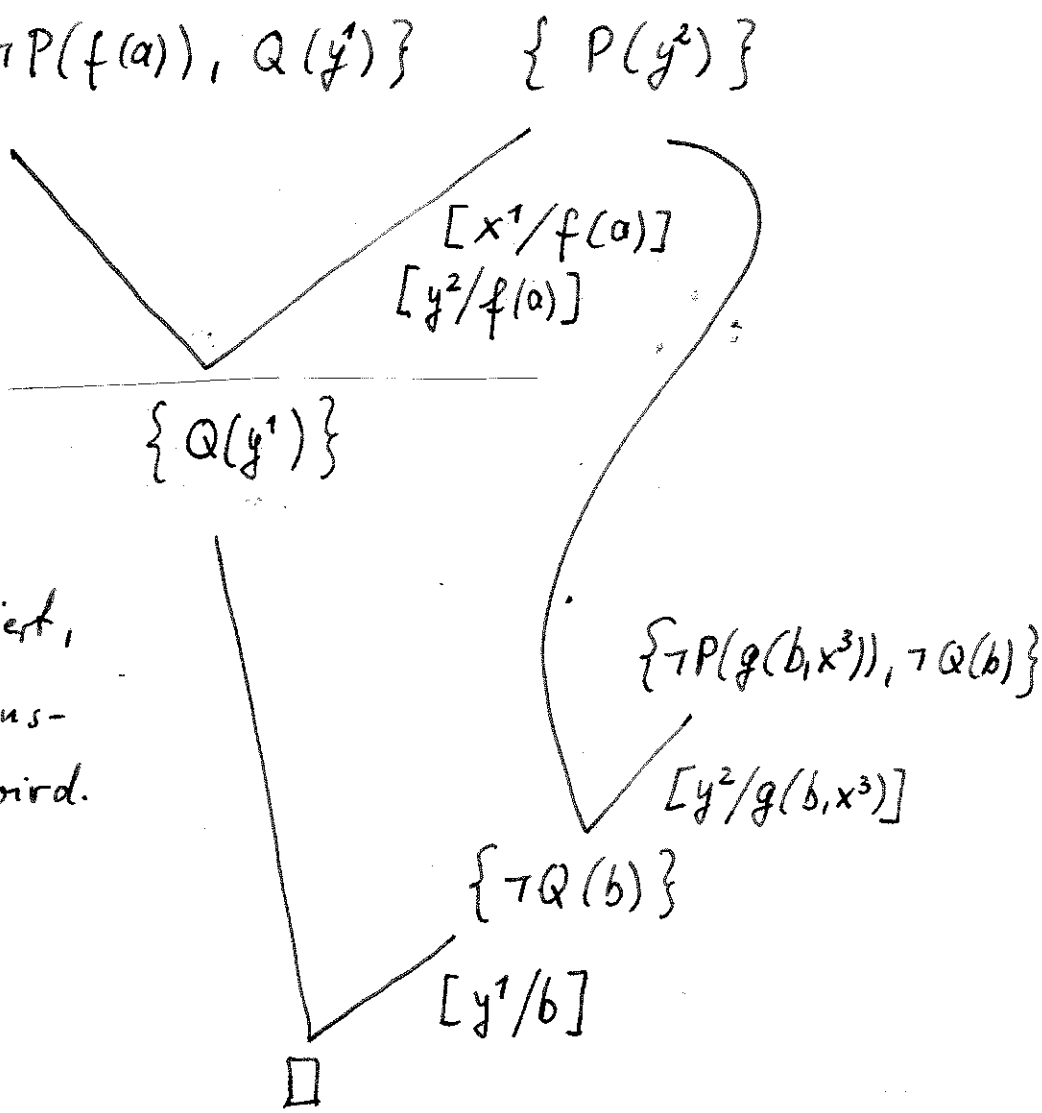
Zusammengefasst:

Grundresolutionssatz: Gegeben $F = \forall x_1 \ldots \forall x_n F^*$ hierbei sei F^* in KNF, besteht also aus Klauseln. Dann ist F genau dann unerfüllbar, wenn eine Folge von Klauseln $K_1, K_2, \ldots, K_t$ existiert, so dass:

- $K_t = \square$
- für $i = 1, \ldots, t$ gilt: entweder ist K_i eine Grundinstanz einer Klausel in F^*, oder K_i ist (aussagenlog.) Resolvent zweier Klauseln K_j, K_l mit $j, l < i$.

Gesucht: Methode, um prädikatenlogische Klauseln direkt zu resolvieren. Gleichzeitig müssen dabei geeignete Substitutionen gefunden werden.

Beispiel:



Idee:
Es wird immer nur soviel substituiert, dass ein Resolutionsschritt ermöglicht wird.

(„allgemeinster Unifikator"; mgu = most general unifier)

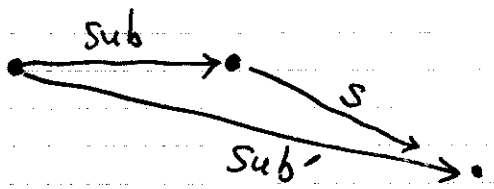
Def: Eine Substitution sub (die eine oder mehrere Variablen durch Terme ersetzt) ist ein Unifikator einer Menge von Literalen $\mathcal{L} = \{L_1, L_2, \ldots, L_k\}$, falls

$L_1 sub = L_2 sub = \ldots = L_k sub$ (kurz: $|\mathcal{L} sub| = 1$)

In diesem Fall heißt $\mathcal{L}$ unifizierbar.

Ein Unifikator sub einer Literalmenge $\mathcal{L}$ heißt dabei allgemeinster Unifikator, falls für jeden Unifikator sub' von $\mathcal{L}$ gibt, dass sich dieser zusammensetzen lässt aus sub und (evtl.) einer weiteren Substitution s:

$$sub' = sub \circ s$$



Bsp: $\mathcal{L} = \{ P(x), P(f(a)), P(y) \}$ wird unifiziert mit sub $= [x/f(a)][y/f(a)]$

Bsp: $\mathcal{L} = \{ P(x), P(f(y)), P(f(g(a,z))) \}$

sub $= [x/f(y)][y/g(a,z)]$ allg. Unif.

sub' $=$ ——— " ——— $[z/b]$ Unif.

$\equiv$ sub $\circ [z/b]$

Unifikationsalg:

$P(x)$

$P(f(y))$

$P(f(g(a,z)))$

↑ ↑

Aufgabe: Nach Herstellen der Pränex- und der Skolemform entsteht am Ende die Formel

$$F^* = (Q(x) \vee Q(g(a,y))) \wedge \neg Q(g(z,z))$$

(a) Wie sind die vorkommenden Variablen x, y, z quantifiziert?

(b) Liegt F^* in KNF vor?

(c) Gib die ersten Elemente des Herbrand-Universums $D(F)$ an.

(d) Ist die zugrunde liegende Formel F unerfüllbar? Wenn ja, zeige dies durch Grundresolution.

(e) Zeige die Unerfüllbarkeit durch prädikatenlogische Resolution (obwohl die formale Def. hiervon noch aussteht).

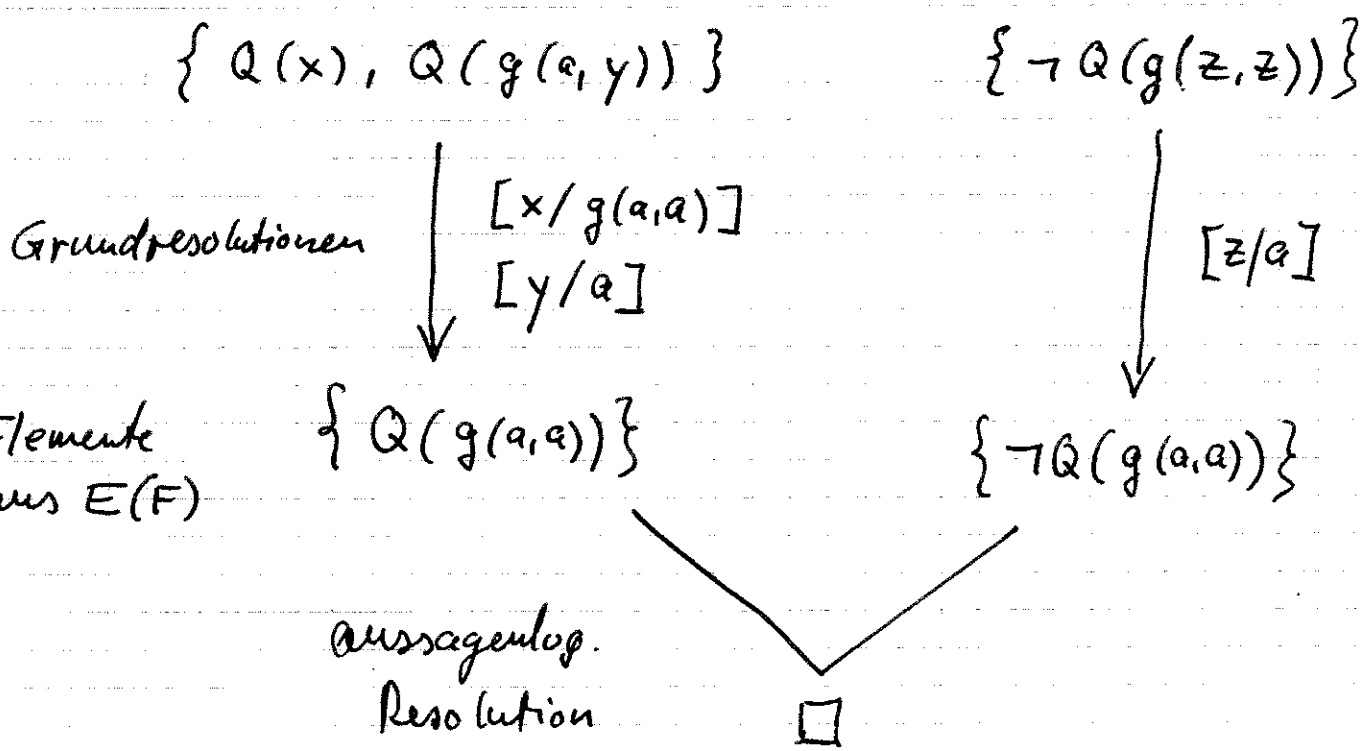
Antwort: (a) x, y, z sind all-quantifiziert. Bei F^* lässt man diese Quantoren weg. (Bei Skolemisieren wurden $\exists$-Quantoren eliminiert.)

(b) Ja, KNF liegt vor. In Klauselform geschrieben lautet F^*:

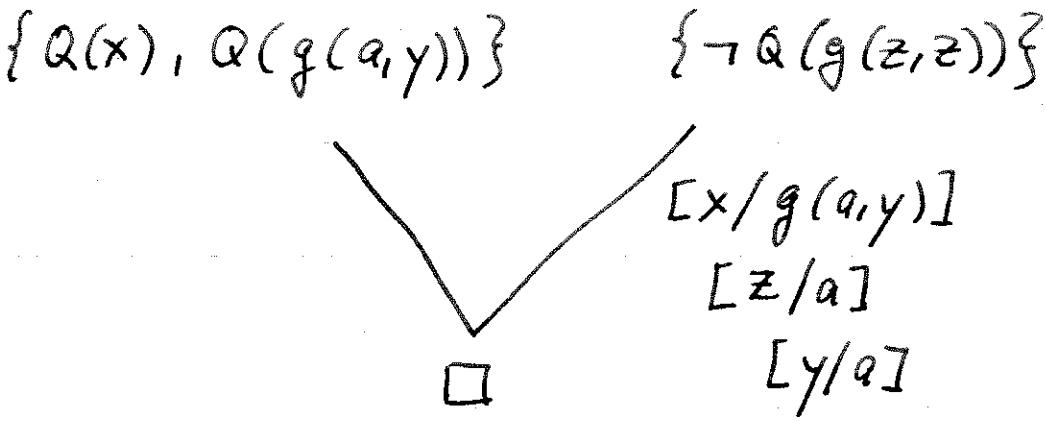
$$\{Q(x), Q(g(a,y))\} \quad \{\neg Q(g(z,z))\}$$

(c) $\mathbb{D}(F) = \{a, g(a,a), g(g(a,a),a), g(a, g(a,a)), \ldots\}$

(d) Ja, F ist unerfüllbar. Mittels Grundresolution (wie im Grundresolutionssatz beschrieben) sieht man das so:



(e) Prädikatenlog. Resolution:



Aufgabe: In der axiomatisch definierten Theorie der Äquivalenzrelationen haben wir als Signatur ein 2-stelliges Prädikat P. Es müssen folgende Axiome gelten:

1. (Reflexivität) $\forall x \; P(x,x)$
2. (Symmetrie) $\forall x \forall y \; (P(x,y) \rightarrow P(y,x))$
3. (Transitivität) $\forall x \forall y \forall z \; (P(x,y) \wedge P(y,z) \rightarrow P(x,z))$

Folgende Formel ist ein Satz dieser Theorie:

4. $\forall x \forall y \forall z \; (P(x,y) \wedge P(z,y) \rightarrow P(z,x))$

Man zeige dies, indem man 1., 2., 3., $\neg$4. in Klauselform bringt und dann per Resolution die Unerfüllbarkeit zeigt.

Lösung: Klauselform von 1: $P(x,x)$ (1)

Klauselform von 2 (wobei wir gleich die Variablen umbenennen): $\neg P(s,t) \vee P(t,s)$ (2)

Klauselform von 3: $\neg P(u,v) \vee \neg P(v,w) \vee P(u,w)$ (3)

Negation der Formel 4, danach Skolemisieren:

$$\neg\ \forall x \forall y \forall z\ (P(x,y) \wedge P(z,y) \rightarrow P(z,x))$$

$$= \exists x \exists y \exists z \neg(\neg(P(x,y) \wedge P(z,y)) \vee P(z,x))$$

$$= \exists x \exists y \exists z \neg(\neg P(x,y) \vee \neg P(z,y) \vee P(z,x))$$

$$= \exists x \exists y \exists z\ (P(x,y) \wedge P(z,y) \wedge \neg P(z,x))$$

$$\rightsquigarrow\ P(a,b) \wedge P(c,b) \wedge \neg P(c,a)$$

(a, b, c neue Skolemkonstanten)

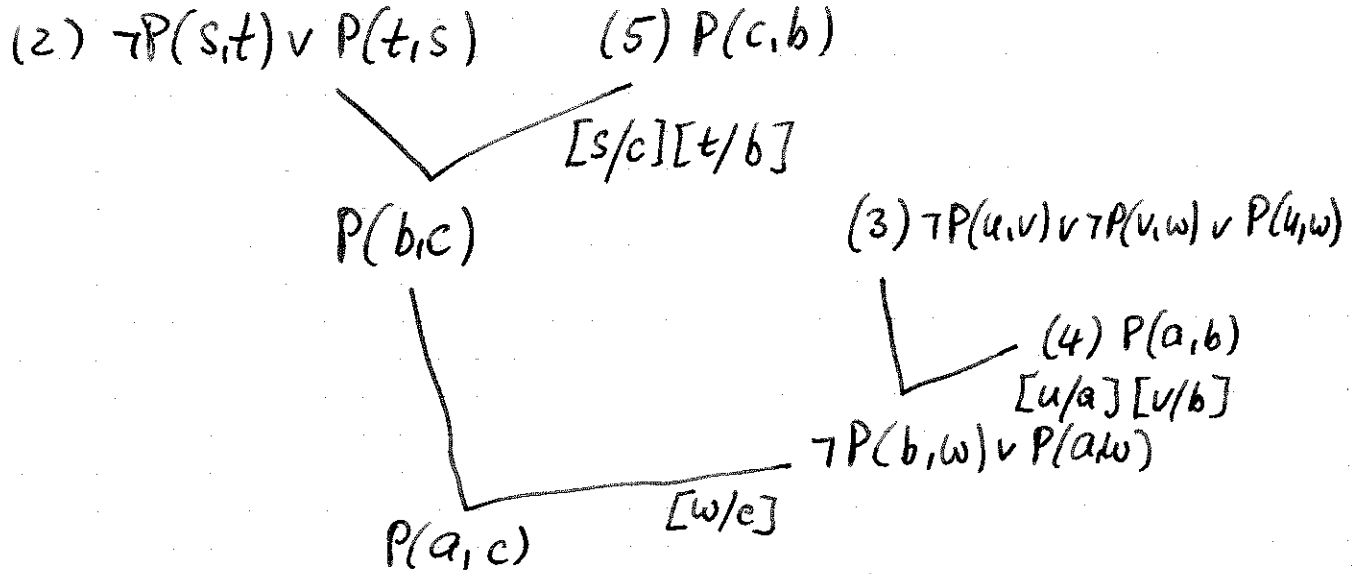
Dies ergibt die einelementigen Klauseln

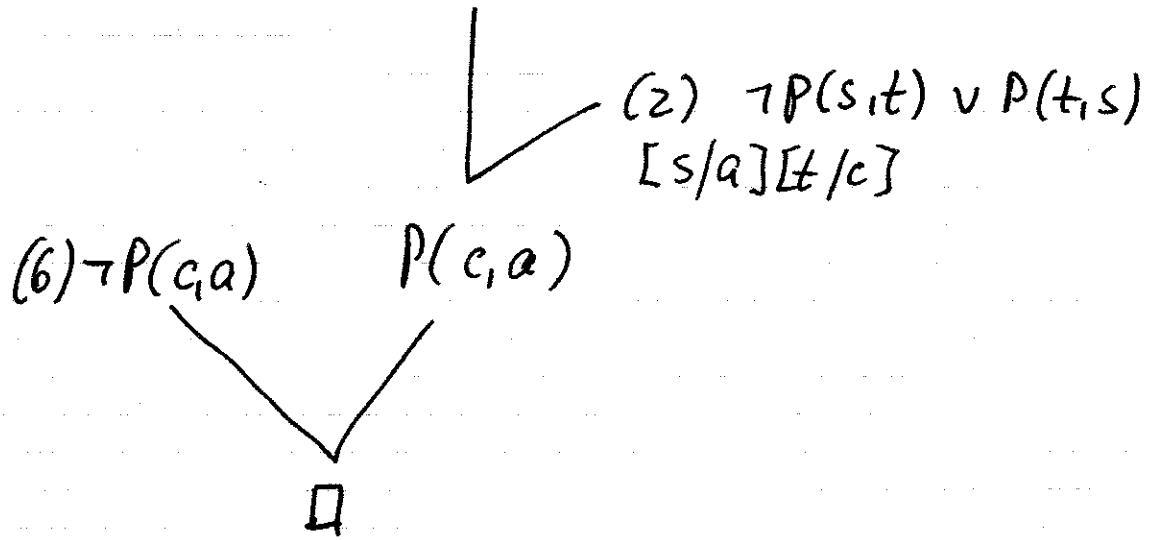
$$P(a,b) \qquad (4)$$

$$P(c,b) \qquad (5)$$

$$\neg P(c,a) \qquad (6)$$

Nun prädl. log. Resolution:




(2) ¬P(s,t) ∨ P(t,s)
[s/a][t/c]
(6) ¬P(c,a)
P(c,a)
□

Der Unifikationsalgorithmus

Gegeben eine Menge von Literalen $\mathcal{L}$
(d.h. Prädikate $P(\ldots), P(\ldots), \ldots$ die als Argumente verschiedene Terme enthalten)

sub := [ ] (die leere Substitution)

while $|\mathcal{L}sub| > 1$ **do**

Durchsuche $\mathcal{L}sub$ von links nach rechts, bis eine Position gefunden, bei der sich 2 Literale $L_1, L_2 \in \mathcal{L}sub$ an der Pos. unterscheiden.

IF keines der Unterscheidungssymbole ist eine Variable **then** „nicht unifizierbar"

Sei x Variable und t Term, der im anderen Literal beginnt (t könnte auch eine Variable sein)

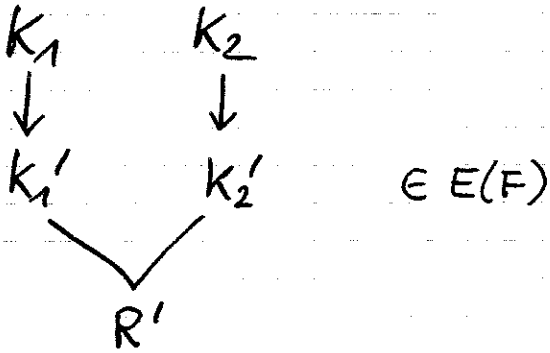
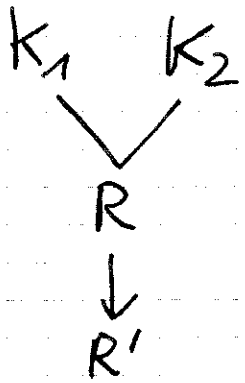
IF x kommt in t vor **then** „nicht unifizierbar" (sog. occur check)

sub := sub [x/t]

end {while}

„unifizierbar" (wobei sub allgemeinster Unifikator ist)

Der Übergang von der Grundresolution zur prädikatenlogischen Resolution wird vermittelt durch das Lifting-Lemma:

> Seien K_1, K_2 zwei präd. logische Klauseln und K_1', K_2' seien Grundinstanzen hiervon (d.h. Substitutionen $[x_i/t_i]$, $t_i \in D(F)$, eingesetzt). Danach erfolgt eine (aussagenlogische) Resolution, die auf den Resolventen R' führt:
>
> 
>
>
> Dann gibt es auch einen prädikatenlog. Resolventen R (es muss nicht unbedingt $R = R'$ gelten), so dass R' Grundinstanz von R ist:
>
> 
>

Dazu muss im Nachgang die prädikatenlog. Form der Resolution definiert werden: Gegeben seien zwei prädikatenlogische Klauseln K_1, K_2. Zunächst müssen diese Klauseln durch Variablen-Umbenennung variablendisjunkt gemacht werden: $K_1 s_1$, $K_2 s_2$ (wobei s_1, s_2 geeignete Variablen-Umbenennungen sind). Erst danach kann der eigentliche Resolutionsvorgang erfolgen. Hierzu wählt man (ein oder mehrere) Literale aus, die in $K_1 s_1$ nur positiv und in $K_2 s_2$ nur negativ auftreten. Nach Abstreifen der Negationen erhält man eine Literalmenge L, bestehend aus mindestens 2 Literalen. Man wendet den Unifikationsalgorithmus auf L an. Sofern dieser mit der Meldung „unifizierbar" endet und einen allgemeinsten Unifikator sub liefert, so entsteht der Resolvent R wie folgt: Aus $K_1 s_1$ und $K_2 s_2$ werden die ausgewählten Literale (die als unifizierbar festgestellt wurden) entfernt und auf die Vereinigung der restlichen verbleibenden Literale die Substitution sub angewandt.

Prädikatenlogischer Resolutionssatz:

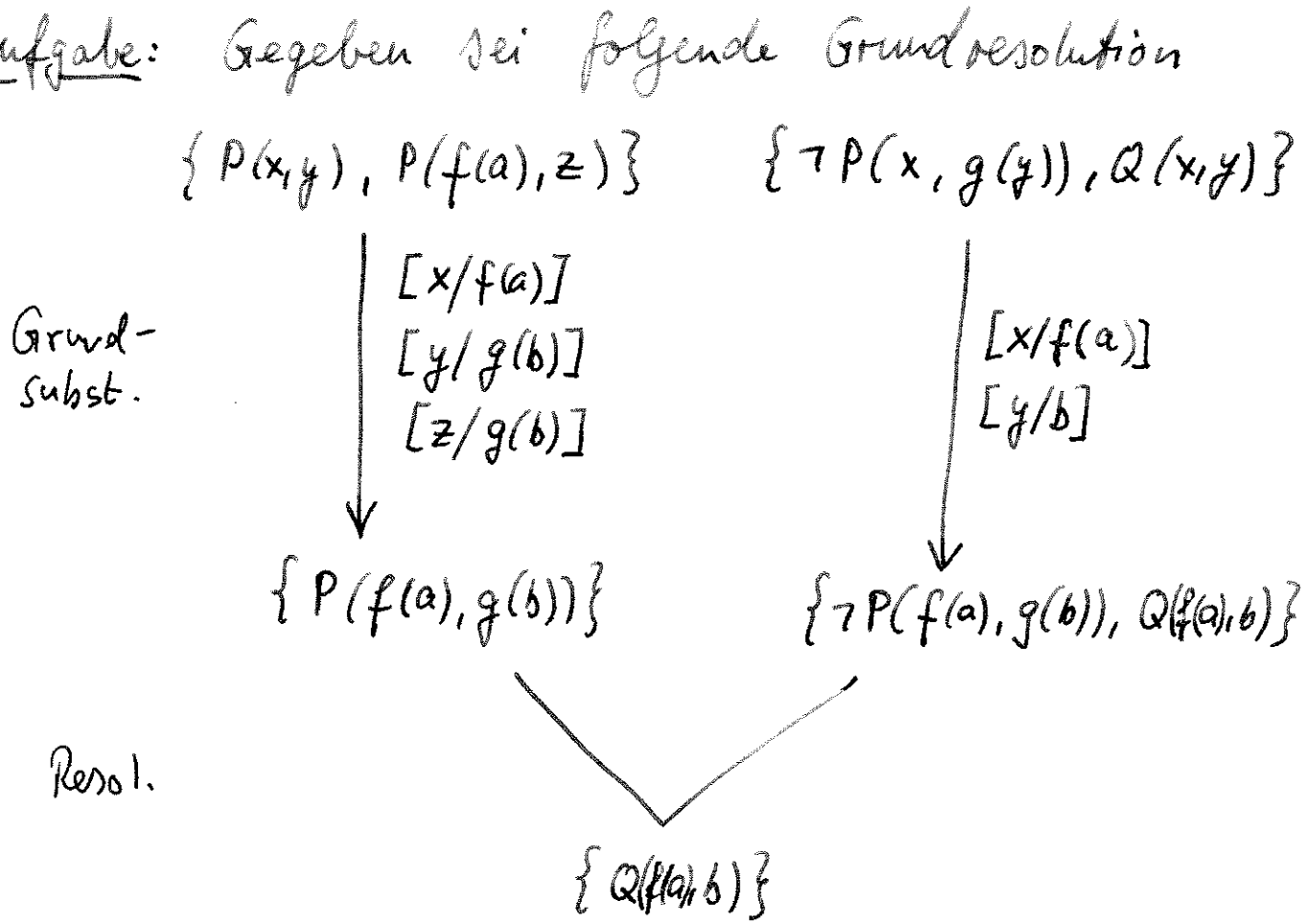
Sei F eine Formel der Prädikatenlogik (ohne freie Variablen), zuvor aufbereitet durch Bereinigen, Pränexform und Skolemform herstellen sowie die Matrix F* in KNF überführen. Dann gilt:

F ist unerfüllbar gdw. es gibt eine prädikatenlogische Resolutionsherleitung der leeren Klausel aus den Klauseln in F*

(Beweis: Grundres.satz + Lifting-Lemma)

Anders als in der Aussagenlogik kann man nicht nach endlich vielen Schritten erwarten, dass man alle möglichen Resolventen erzeugt hat. Bei der prädl. logischen Resolution muss man mit unendlich vielen potenziellen Resolventen rechnen (da die Terme immer komplexer werden können).
Deshalb kann man nie sicher sein, wann man aufhören kann (das ist die Semi-Entscheidbarkeit).

Aufgabe: Gegeben sei folgende Grundresolution



Vollziehen Sie nach wie beim Lifting-Lemma hieraus eine präd. logische Resolution wird.

Antwort: Wir unifizieren $\mathcal{L} = \{ P(x^1, y^1), P(f(a), z^1), P(x^2, g(y^2)) \}$

Hierbei wird durch Hochstellen von Indizes die zunächst herzustellende Variablen-Disjunktheit garantiert, d.h: $K_1 S_1 = \{ P(x^1, y^1), P(f(a), z^1) \}$

$K_2 S_2 = \{ \neg P(x^2, g(y^2)), Q(x^2, y^2) \}$

Der Unifikationsalgorithmus, angewandt auf $\mathcal{L}$, liefert: $[x^1/f(a)][x^2/f(a)][y^1/z^1][z^1/g(y^2)]$

Damit ergibt sich der Resolvent:

$$\{Q(f(a), y^2)\}$$

Der Resolvent $\{Q(f(a), b)\}$ bei der Grundresolution entsteht hieraus durch die Substitution $[y^2/b]$.

Bsp: Jeder Barbier rasiert alle, die sich nicht selbst rasieren:

$$\forall x \left(B(x) \rightarrow \forall y \left(\neg R(y,y) \rightarrow R(x,y) \right) \right)$$

B ... Barbier sein

R ... rasieren

Umformen:

$$\forall x \forall y \left(\neg B(x) \vee \left(R(y,y) \vee R(x,y) \right) \right)$$

als Klausel: $(\neg B(x) \vee R(y,y) \vee R(x,y))$ (1)

Kein Barbier rasiert jemanden, der sich selbst rasiert:

$$\neg \exists z \, (B(z) \wedge \exists u \, (R(u,u) \wedge R(z,u)))$$

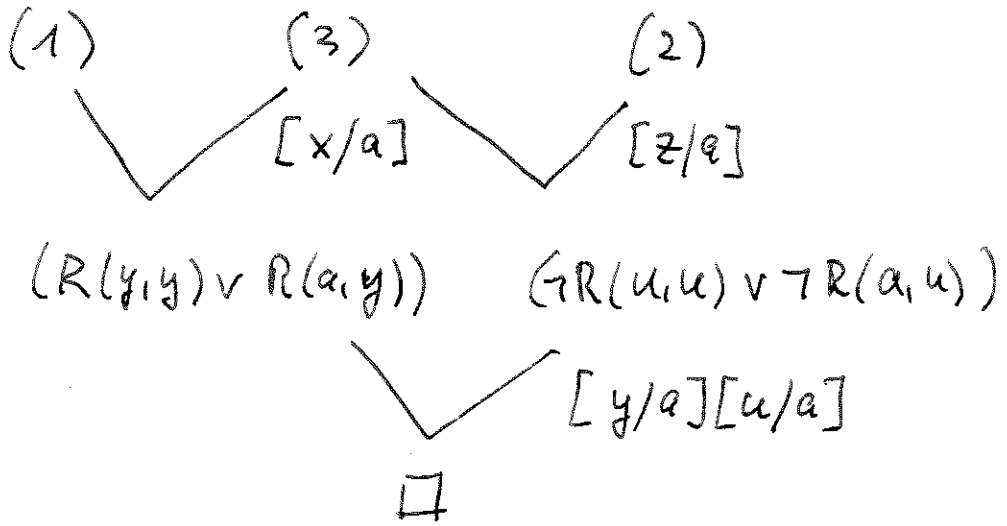
$$= \forall z \, (\neg B(z) \vee \forall u \, (\neg R(u,u) \vee \neg R(z,u)))$$

$$= \forall z \forall u \, (\neg B(z) \vee \neg R(u,u) \vee \neg R(z,u))$$

als Klausel $(\neg B(z) \vee \neg R(u,u) \vee \neg R(z,u))$ (2)

Beh: Aus (1) und (2) folgt: Es gibt keine Barbiere:

Negation: $\exists v \, B(v) \rightsquigarrow B(a)$ (3)



Aufgabe: Stelle fest, ob ~~klassifiziert~~ folgende Menge

$$\mathcal{L} = \{ P(x_1, x_2, \ldots, x_n),\ P(f(x_0, x_0), f(x_1, x_1), \ldots, f(x_{n-1}, x_{n-1})) \}$$

unifizierbar ist und stelle Überlegungen an, wie man den Unifikationsalgorithmus effizient machen kann.

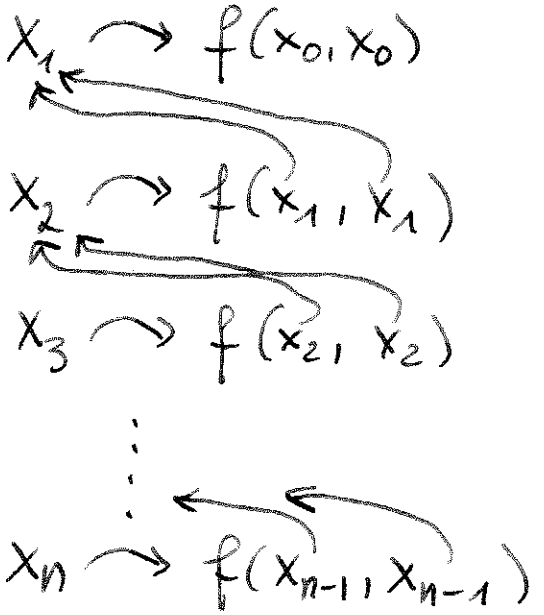
Antwort: Hier entstehen der Reihe nach folgende Substitutionen:

$$[x_1 / f(x_0, x_0)]$$

$$[x_2 / f(f(x_0, x_0), f(x_0, x_0))]$$

usw.

Die Längen der rechten Seiten nehmen exponentiell zu. Effizienter ist es, man speichert nur Links, die sukzessive ineinander eingesetzt werden können.



Aufgabe: Bei manchen Implementierungen von Prolog-Interpreten wird der Unifikationsalgorithmus (aus Effizienzgründen) ohne occur check (kommt x im Term t vor? – bevor man [x/t] hinzufügt) implementiert. Gib eine Literalmenge $\mathcal{L} = \{L_1, L_2\}$ an, so dass L_1, L_2 keine gemeinsamen Variablen enthalten. Trotzdem tritt ein Fehler auf, da der occur check nicht durchgeführt wird.

Lösung: $\mathcal{L} = \{ P(u,u),\ P(f(x), x) \}$

Zunächst wird $sub = [u/f(x)]$ gesetzt.
Danach gilt $\mathcal{L}sub = \{ P(f(x), f(x)),\ P(f(x), x) \}$
Als Nächstes müsste sub um $[x/f(x)]$ erweitert werden. Wegen occur check sollte nun korrekterweise „nicht unifizierbar" ausgegeben werden. Ohne occur check erhält man „unifizierbar" und bei Abfrage nach dem allg. Unifikator erhält man:
$sub = f(f(f(f(f(f(f($ - - - - - - - -

Aufgabe: Gib alle möglichen Resolventen an von

$K_1 = \{\neg P(x,y), \neg P(f(a), g(u,b)), Q(x,u)\}$ und

$K_2 = \{P(f(z), g(a,b)), \neg Q(f(a), b), \neg Q(a,b)\}$

Antwort:

1. mittels $[x/f(z)][y/g(a,b)]$:

$\{\neg P(f(a), g(u,b)), Q(f(z),u), \neg Q(f(a),b), \neg Q(a,b)\}$

2. mittels $[z/f(a)][u/a]$:

$\{\neg P(x,y), Q(x,a), \neg Q(f(a),b), \neg Q(a,b)\}$

3. mittels $[x/f(z)][y/g(a,b)][z/a][u/a]$:

$\{Q(f(a),a), \neg Q(f(a),b), \neg Q(a,b)\}$

4. mittels $[x/f(a)][u/b]$:

$\{\neg P(f(a),y), \neg P(f(a), g(b,b)), \neg Q(a,b)\}$

5. mittels $[x/a][u/b]$:

$\{\neg P(a,y), \neg P(f(a), g(b,b)), \neg Q(f(a),b)\}$

Aufgabe: Man drücke folgende Tatsachen als präd. logische Formeln aus und forme sie in Skolem+ Klauselform:

(a) Jeder Drache ist glücklich, wenn alle seine Kinder fliegen können

(b) Grüne Drachen können fliegen.

(c) Ein Drache ist grün, wenn er Kind mindestens eines grünen Drachens ist.

Man zeige durch präd. log. Resolution, dass aus (a), (b), (c) folgt, dass alle grünen Drachen ~~können~~ glücklich sind.

Lösung: Wir verwenden folgende Prädikate:

Ki(x,y) ... x ist Kind von y

Fl(x) ... x kann fliegen

Gl(x) ... x ist glücklich

Gr(x) ... x ist grün

Die Fakten als präd. log. Formeln:

(a) $\forall x \left(\forall y \left(Ki(y,x) \rightarrow Fl(y) \right) \rightarrow Gl(x) \right)$

in Klauselform:

(a1) $\{ Ki(sk(x), x), Gl(x) \}$ ← Skolemfkt.

(a2) $\{ \neg Fl(sk(x)), Gl(x) \}$

(b) $\forall z \left(Gr(z) \rightarrow Fl(z) \right)$

als Klausel: (b1) $\{ \neg Gr(z), Fl(z) \}$

(c) $\forall u \left(\exists v \left(Ki(u,v) \wedge Gr(v) \right) \rightarrow Gr(u) \right)$

als Klausel: (c1) $\{ \neg Ki(u,v), \neg Gr(v), Gr(u) \}$

Die zu beweisende Behauptung ist

(d) $\forall w \left(Gr(w) \rightarrow Gl(w) \right)$

Bem: Die Formel (a), (b), (c) stellen sozusagen die axiomatisch definierte „Theorie der Drachen" dar. Es soll gezeigt werden, dass (d) ein Satz dieser Theorie ist.

Die zu beweisende Behauptung (d) wird negiert und es wird nun gezeigt, dass

$(a) \wedge (b) \wedge (c) \wedge \neg(d)$ unerfüllbar ist.

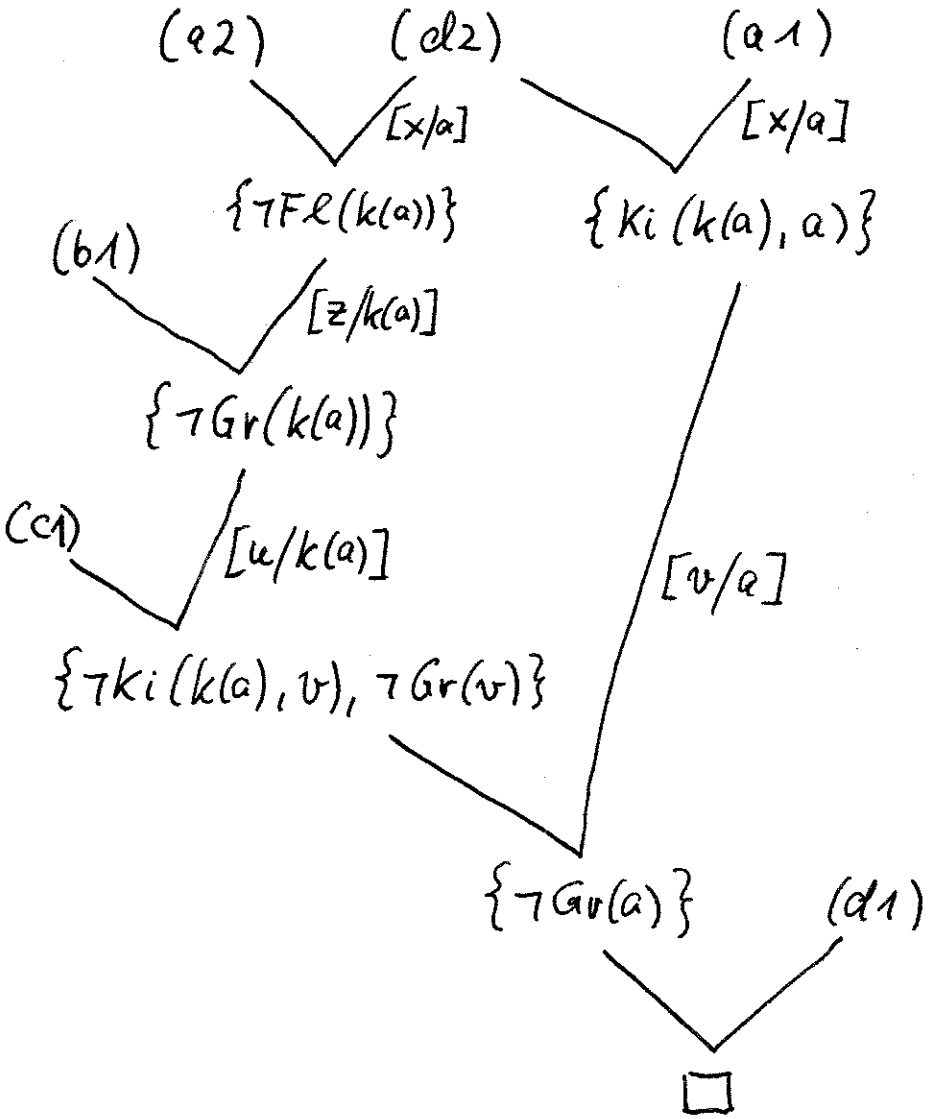
Aus $\neg(d)$ erhält man die Klauseln:

Skolemkonstante

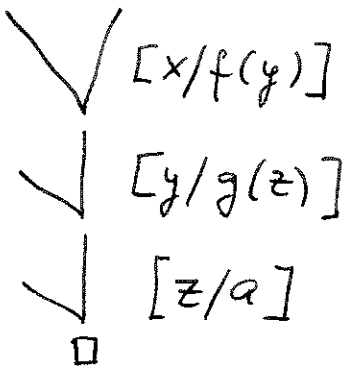
(d1) $\{Gr(a)\}$

(d2) $\{\neg Gl(a)\}$

Hier ist ein entsprechender Resolutionsbeweis:

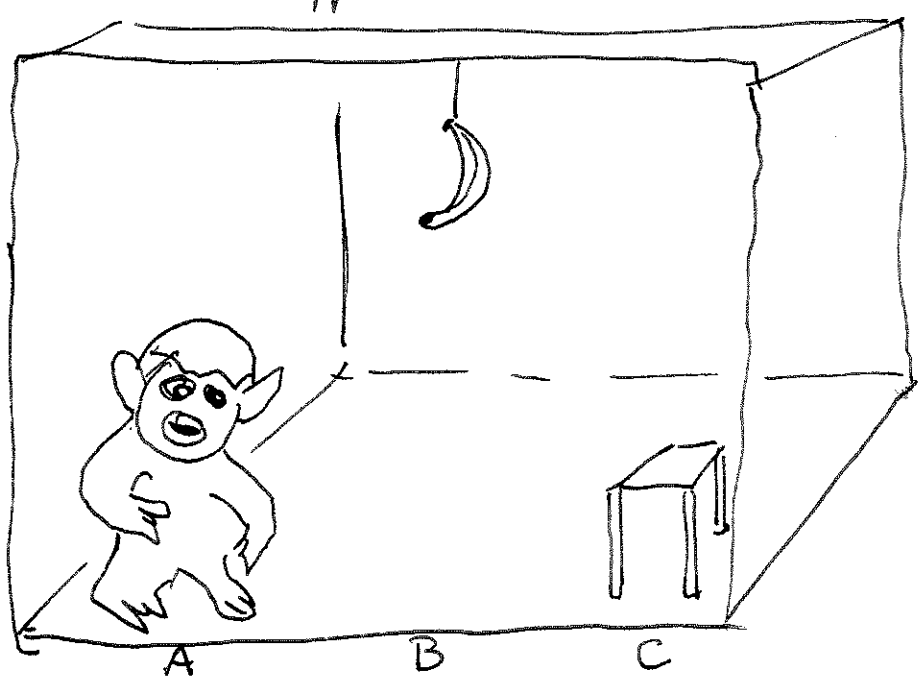


Neuer Aspekt: Die bei den Resolutionsschritten erhaltenen Unifikatoren können auch wie ein „Rechenergebnis" aufgefasst werden:



Lies: Um die „Lösung" zu erhalten führe $[x/f(g(a))]$ durch. Dies können bestimmte „Aktionen" f und g sein. Diese Sichtweise spielt bei sog. Planungssystemen eine große Rolle.

Beispiel: Das Affe-und-Bananen-Problem:



Affe bei Position A,

Banane unter Position B,

Stuhl bei C.

(1) $\{P(a,b,c,d)\}$

„In der Startsituation d befindet sich Affe bei a, Banane bei b, Stuhl bei c."

(2) $\{\neg P(x,y,z,s),\ P(w,y,z,walk(x,w,s))\}$

„Wenn sich in Situation s der Affe bei x befindet so bewirkt die Anwendung der Funktion walk(x,w,s), dass der Affe nachher bei Position w ist."

(3) $\{\neg P(x,y,x,s),\ P(w,y,w,push(x,w,s))\}$

„Wenn in Situation s Position von Affe und Stuhl übereinstimmen (der Affe steht beim Stuhl), dann kann durch den Operator push(x,w,s) die Situation so geändert werden, dass Affe + Stuhl bei Position w sind."

(4) $\{\neg P(x,y,x,s),\ P(x,y,x,climb(s))\}$

„Wenn in Situation s der Affe beim Stuhl steht, so kann er diesen ersteigen." (Operator climb)

(5) $\{\neg P(x, x, x, climb(s)),\ Reach(grasp(climb(s)))\}$

„Wenn der Affe in Situation s auf den Stuhl gestiegen ist, und wenn die Positionen von Affe, Stuhl und Banane übereinstimmen, so kann durch Greifen (grasp) das Prädikat Reach (der Affe hat die Banane) erfüllt werden."

Dieses sog. Logik-Programm (bzw. ein Axiomensystem, das eine axiomatische Theorie definiert) wird durch die Klauseln (1)-(5) definiert.

Es soll nun gezeigt werden, dass dann folgt, dass der Affe die Banane erreichen kann. Quasi als „Seiteneffekt" wird sich aus dem Resolutionsbeweis (aus den berechneten Unifikatoren) ergeben, welche Aktionen durch zu führen sind, um das Erreichen der Banane zu ermöglichen.

Hierzu muss die zu beweisende Aussage

$\exists z \; \text{Reach}(z)$

negiert werden: (6) $\{\neg \text{Reach}(z)\}$

und per Resolution die Unerfüllbarkeit gezeigt werden:

(SLD-Resolution)

(6)

(5) $[z/\text{grasp}(\text{climb}(s))]$

$\{\neg P(x, x, x, \text{climb}(s))\}$

(4) $[y/x]$

$\{\neg P(x, x, x, s)\}$

(3) $[s/\text{push}(x, y, s^1)]$

$\{\neg P(x, y, z, s^1)\}$

(2) $[s^1/\text{walk}(x, z, s^2)]$

$\{\neg P(x, y, z, s^2)\}$

(1) ~~$[s^2/\text{grasp}(\text{climb}(s$~~

$[x/a][y/b][z/c][s^2/d]$

$\square$

Fügt man die Substitutionen, denen z unterzogen wird, zusammen, erhält man:

$z = grasp(climb(push(c, b, walk(a, c, s))))$

Lies: In der Situation s sollte der Affe von Position a nach Position c gehen (walk(a, c, s)). Dann den Stuhl von c nach b schieben (push(c, b, ...)). Dann auf den Stuhl steigen (climb(...)). Sodann durch Greifen der Banane diese erreichen: Reach(grasp(...)).

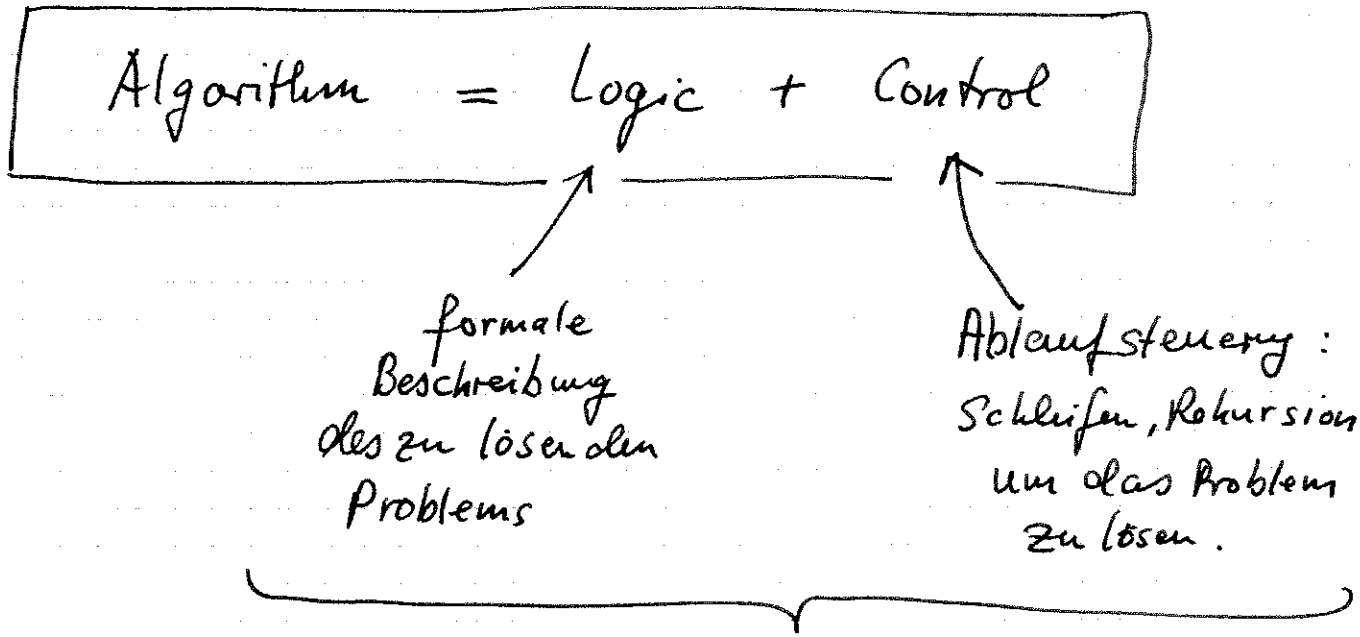
Logik-Programme

sind Horn-Klauseln, genauer: definite Horn-Klauseln, also Fakten und Regeln. Der Aufruf eines Logik-Programms erfolgt durch eine Zielklausel oder Fragеklausel (die Negation der zu beweisenden Behauptung), also eine rein-negative Klausel.

Sofern die Zielklausel eine oder mehrere Variablen enthält, so liefert der Resolutionsbeweis (sofern dieser auf die leere Klausel führt) sozusagen als Seiteneffekt eine entsprechende Substitution für die fragliche(n) Variable(n).

Die Programmiersprache PROLOG (= Programming in Logic) realisiert diese Idee der Logik-Programmierung, enthält jedoch einige weitere nicht-logische Komponenten, z.B. Prädikate wie read, consult, output etc. die feste Interpretationen haben

Grundsätzliche Idee:



bei traditionellen Programmiersprachen muss sich der Programmierer um beides kümmern

Bei der logik-Programmierung beschreibt der Programmierer nur das zu lösende Problem – den Rest, also die entsprechende Ausführung (Control) übernimmt das System.

Konventionen in Prolog:

kleine Buchstaben: Funktionen, Prädikate
große Buchstaben: Variablen

Bsp: `q(a,X,f(g(Y,b))).` (Fakt)

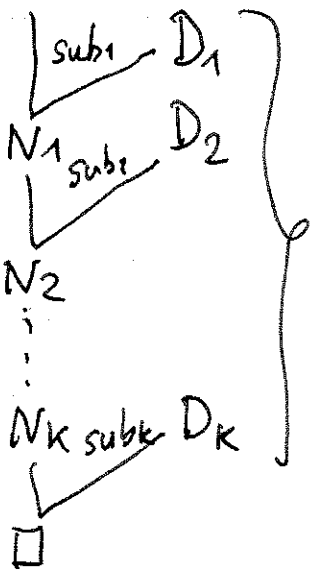
$p(X) :- q(X), r(X,Y).$ (Regel)

$\hat{=}$ $P(x) \leftarrow Q(x) \wedge R(x,y)$

$\hat{=}$ $P(x) \vee \neg Q(x) \vee \neg R(x,y)$

SLD-Resolution:

Zielklausel: N (enthält X)



definite Klauseln (Fakten + Regeln)

Antwort: yes, $X = sub_1\ sub_2 \ldots sub_k$

PROLOG (Programming in Logic)

Im Internet frei verfügbar das Buch

Esther König, Roland Seiffert: Grundkurs Prolog für Linguisten (googeln nach König, Seiffert Prolog)

Ferner die einfache Prolog-Implementierung

„B-PROLOG" (danach googeln)

Man erhält eine zip-Datei, in welcher sich eine Doku, einige Beispielprogramme und die direkt ausführbare Datei bp.exe befindet.

Das Prolog-System meldet sich mit dem Prompt: ?-

Dahinter kann man eine Zielklausel, gefolgt von einem Punkt, eingeben.

Hierfür sollte aber zunächst ein Prolog-Programm zur Verfügung stehen. Ein solches erstellt man im normalen Editor und speichert es ab als

beispiel.pl

Zum Beispiel schreiben wir im Editor das Prolog-Programm:

```
person(hans).
person(maria).
```

und speichert dieses als beispiel.pl ab.

Nach Aufruf des Prolog-Systems (bp.exe) gibt man nach dem Prompt ?- ein: consult(beispiel).

Wenn man nach dem nachfolgend Prompt ?- eingibt: listing.

so erfährt man, dass das Prolog-Programm nun eingelesen ist. Gibt man ein

```
?- person(hans).
```

so erhält man als Antwort: yes.

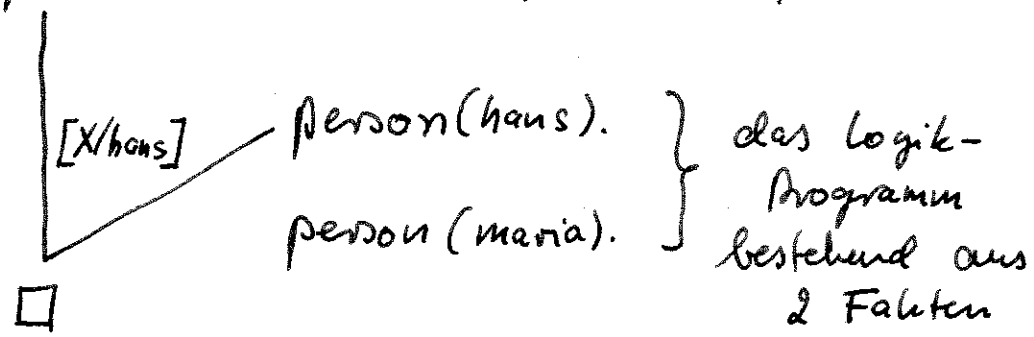
Gibt man ein ?- person(X).

so erhält man als Antwort X = hans. yes

Will man noch weitere Antworten erhalten, so gibt man Semikolon (;) ein: X = maria.

Hinter diesen Antworten stecken bereits prädikatenlogische Resolutionen mit Unifikation:

?- person(X). (entspricht: ¬person(X))



□

Darüber hinaus kann das Logik-Programm auch Regeln enthalten. Diese haben die Form a :- b, c.

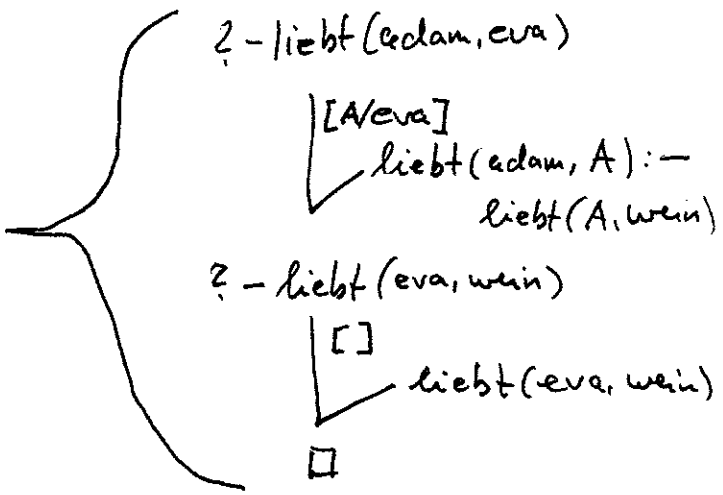
$$\begin{cases} a \leftarrow (b \wedge c) \\ a \vee \neg b \vee \neg c \end{cases}$$

Typischerweise kommen in Regeln auch Variablen vor, die man groß schreibt.

Wir schreiben folgendes Logik-Programm:

```
liebt(eva, pizza).
liebt(eva, wein).
liebt(adam, X) :- liebt(X, wein).
```

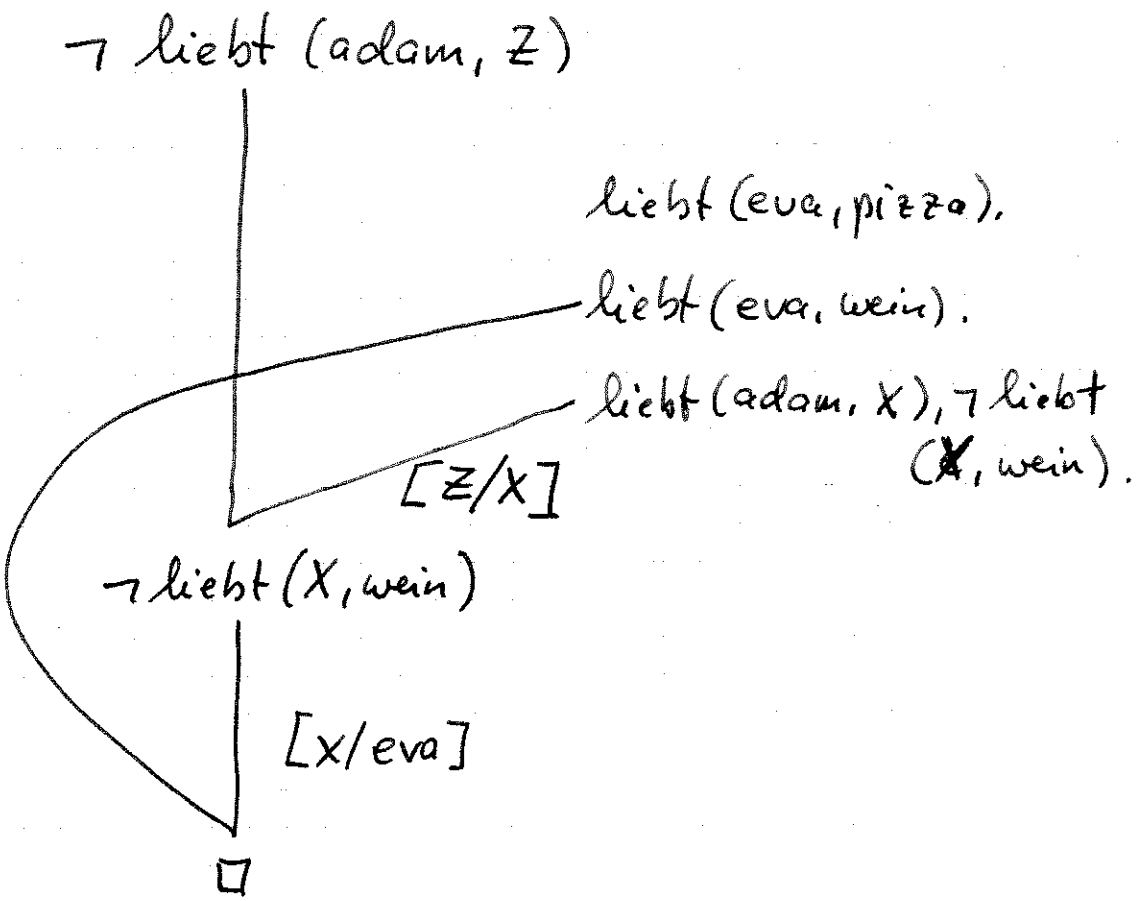
```
B-Prolog Version 8.1, All rights reserved,
| ?- consult(beispiel).
consulting::beispiel.pl

yes
| ?- listing.
liebt(eva, pizza).
liebt(eva, wein).
liebt(adam, A) :-
        liebt(A, wein).

yes
| ?- liebt(eva,wein).

yes
| ?- liebt(eva,bier).
no
| ?- liebt(adam,eva).

yes
| ?- liebt(X,wein).
X = eva ?
yes
| ?- liebt(X,Y).
X = eva
Y = pizza ?;
X = eva
Y = wein ?;
X = adam
Y = eva ?;
no
| ?- liebt(X,X).
no
| ?-
```



Die Anfrage ?- liebt(adam, Z). liefert

Z = eva.

Daher steht folgender Resolutionsbeweis:



Dies ergibt: $Z\,[Z/X][X/eva] = eva$

Prolog-Auswertungsstrategie: Für jeden neuen SLD-Resolutionsschritt wird das Prolog-Programm nach einer passenden Klausel von oben nach unten durchsucht. Hierbei wird immer nur das erste Literal der Zielklausel versucht mit dem ersten (und einzigen positiven) Literal in der Logik-Programm-Klausel

zu unifizieren, um bei Unifizierbarkeit einen Resolventen zu erzeugen.

Backtracking: Kann nicht weiter resolviert werden und die leere Klausel ist noch nicht erreicht, so wird der letzte Resolutionsschritt rückgängig gemacht und nach einem alternativen Resolventen gesucht.

Beispiel:

```
liebt(eva, pizza).
liebt(eva, wein).
liebt(adam, X) :- liebt(X, bier).
liebt(adam, X) :- liebt(X, wein).
```

Zielklausel: `?- liebt(adam, X).`

Der erste Versuch, nach einem Bier-Liebhaber zu suchen, schlägt fehl, dann wird die nächste Klausel herangezogen. Ein Wein-Liebhaber wird gefunden und die Antwort ist:

$$X = \text{eva}.$$

Anhang A: Prolog-Auswertungsstrategie

Eingabe: Logik-Programm $F = (K_1, K_2, \ldots, K_n)$, wobei
$K_i = \mathtt{B}_i :- \mathtt{C}_{i,1}, \ldots, \mathtt{C}_{i,n_i}$ $(n_i \geq 0)$ und Zielklausel $G = ?-\mathtt{A}_1, \ldots, \mathtt{A}_k$.
(Beachte: sofern $n_i = 0$, so ist K_i eine Faktenklausel.)

Das Hauptprogramm besteht aus

```
success := false;
auswerte( G, [ ] );
if success then write('ja') else write('nein');
```

wobei die rekursive Prozedur 'auswerte' folgendermaßen arbeitet.

```
procedure auswerte ( G : Zielklausel; sub : Substitution );
var i : integer;
begin
  if G = □ then
    begin
      H := (A_1 ∧ ··· ∧ A_k) sub;
      write ('Ergebnis:',H);
      success := true
    end
  else {G habe die Form G = ?-D_1, ..., D_m}
    begin
      i := 0;
      while (i < n) and not success do
        begin
          i := i + 1;
          if {D_1, B_i} ist unifizierbar mittels allgemeinstem
             Unifikator s (wobei die Variablen in B_i evtl. zu-
             vor umbenannt wurden) then
               auswerte ( ?-(C_{i,1}, ..., C_{i,n_i}, D_2, ..., D_m) s, sub s )
        end
    end
end
```

Die bisherigen Beispiele beinhalten nur Konstanten (also 0-stellige Funktionssymbole) und Prädikate (liebt(...), person(...)).
Das Herbrand-Universum eines solchen Logik-Programms (das nicht den vollen Umfang der Prädikatenlogik ausnutzt – dies nennt man Datalog) besteht nur aus den endlich vielen, vorkommenden Konstanten. Bei letzten Beispiel wäre dies:

$$D(F) = \{eva, pizza, wein, bier, adam\}$$

Aufgrund der festen Auswertungsstrategie von PROLOG kann es selbst bei einem Logik-Programm in dem einfachen Datalog-Fragment der Präd.-logik zu Unendlich-Schleifen kommen.

Bsp: Das Logik-Programm

```
sohn(anton, berta).
geschwister(claus, berta).
geschwister(X, Y) :- geschwister(Y, X).
neffe(A, B) :- sohn(A, Z), geschwister(Z, B).
```

liefert auf die Anfrage

?- neffe(anton, A).

Also: von wem ist anton der Neffe? die Antwort

A = claus

Denn: ?- neffe(anton, A).

neffe(A', B') :- sohn(A', Z'), geschwister(Z', B')
[A'/anton][A/B']

?- sohn(anton, Z'), geschwister(Z', B')

[Z'/berta]
sohn(anton, berta).

?- geschwister(berta, B')

[X/berta][B'/Y]
geschwister(X, Y) :- geschwister(Y, X).

?- geschwister(Y, berta)

[Y/claus]
geschwister(claus, berta)

□

Antwort: A[A/B'][B'/Y][Y/claus] = claus

Wenn man allerdings die Reihenfolge der beiden geschwister-Klauseln vertauscht, so läuft der PROLOG-Interpreter in eine Unendlichschleife. Faustregel: erst die Fakten, dann die Regeln auflisten.

Im folgenden Logik-Programm kommt eine Konstante a sowie eine 1-stellige Funktion s() vor.

Damit ist $D(F) = \{ a, s(a), s(s(a)), s(s(s(a))), \ldots \}$

Diese Menge könnte man als die Menge der natürlichen Zahlen interpretieren: $\{0, 1, 2, 3, \ldots\}$

Man kann die Addition als 3-stelliges Prädikat add(X, Y, Z) (im Sinne von $X+Y=Z$) definieren:

	bedeutet inhaltlich:
add(X, a, X).	$X+0 = X$
add(X, s(Y), s(Z)) :- add(X, Y, Z).	$X+(Y+1)=(Z+1)$ schon $X+Y=Z$

Beispielabfragen:

?- add(s(s(a)), s(s(s(a))), X). liefert: $X = s(s(s(s(s(a)))))$

?- add(s(s(a)), X, s(s(s(a)))). liefert $X = s(a)$

?- add(X, Y, s(s(s(a)))). liefert der Reihe nach:

X = s(s(s(a)))	X = s(s(a))	X = s(a)	X = a
Y = a	Y = s(a)	Y = s(s(a))	Y = s(s(s(a)))

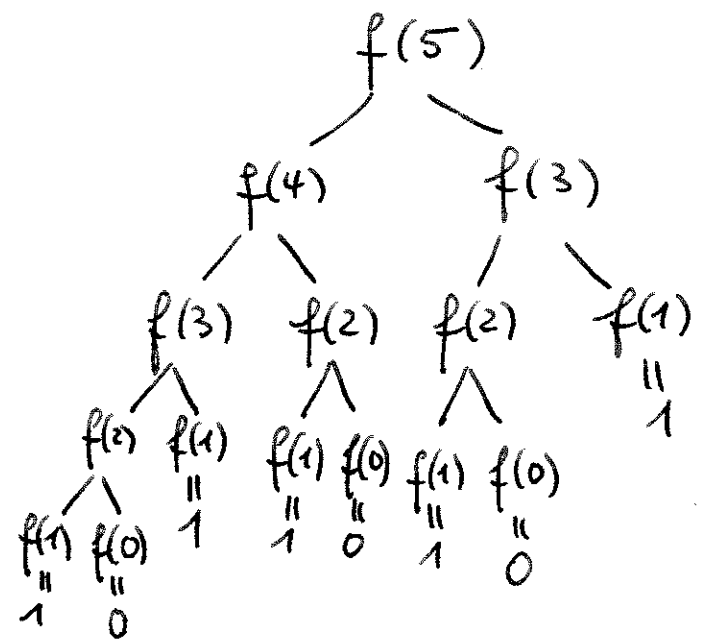
Die <u>Fibonacci-Funktion</u>: $f(0)=0$; $f(1)=1$;

$f(n+2) = f(n+1) + f(n)$ für $n \geq 0$.

Diese kann man durch ein 2-stelliges Prädikat darstellen:

```
fib(a, a).
fib(s(a), s(a)).
fib(s(s(A)), X) :- fib(s(A), Y),
                   fib(A, Z),
                   add(Y, Z, X).
```

Durch diese rekursive Darstellung der Funktion f (und das entsprechende Prädikat fib) – und die Art wie ein PROLOG-Interpreter dies auswertet, ergibt sich eine exponentielle Komplexität. <u>Bsp:</u>



In jeder Verzweigung muss eine Addition ausgeführt werden.

Die <u>Ackermann-Funktion</u> (→ Vorlesung Berechenbarkeit)

$$a(0, y) = y + 1$$

$$a(x+1, 0) = a(x, 1).$$

$$a(x+1, y+1) = a(x, a(x+1, y))$$

Als 3-stelliges Prädikat **ack**(...) :

```
ack(a, Y, s(Y)).

ack(s(X), a, Z) :- ack(X, s(a), Z).

ack(s(X), s(Y), Z) :- ack(s(X), Y, U),
                      ack(X, U, Z).
```

```
| ?- ack(s(s(s(a))),s(s(s(a))),Z).
Z = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(
s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(a)))))))))))))))))))))))))))))))))
)))))))))))))))))))))))))))) ?
yes
| ?-
```

Das bedeutet: $a(3,3) = 61$

Wiederholung:

Hornformeln sind Formeln in KNF, wobei jede Klausel eine Hornklausel ist. Eine Hornklausel hat ≤ 1 positives Literal.

Spezialfälle: ∘ Klausel besteht nur aus 1 pos. Literal $\hat{=}$ Faktenklausel (oder Fakt)

∘ Klausel besteht aus 1 pos. und ≥ 1 negativen Literalen $\hat{=}$ Regelklausel.

{Faktenklauseln} ∪ {Regelklauseln} = {definite Hornklauseln} (das Logik-Programm)

∘ Klausel besteht nur aus negativen Literalen, aber kein positives $\hat{=}$ Zielklausel (oder Frageklausel). oder Aufrufklausel

Prolog-Notationen:

```
mensch(maier).                    ... Fakt

neffe(X,Y) :- sohn(X,Z), geschwister(Z,Y).    ... Regel

?- mensch(X), neffe(X, anton).    ... Zielklausel
```

Einige Resolutionsrestriktionen sind im allgemeinen Fall nicht widerlegungsvollständig, sind es aber, wenn sie auf Hornformeln angewandt werden.

- Unit Resolution: Mindestens eine der Elternklausel muss eine Unit (eine einelementige Klausel) sein

- Input Resolution: Mindestens eine der Elternklauseln muss immer aus dem Input (der ursprünglichen Klauselmenge) stammen.

- SLD-Resolution: Der Resolutionsbeweis muss am Anfang eine Zielklausel / Frageklausel verwenden. Die Resolutionsschritte bilden eine lineare Kette:

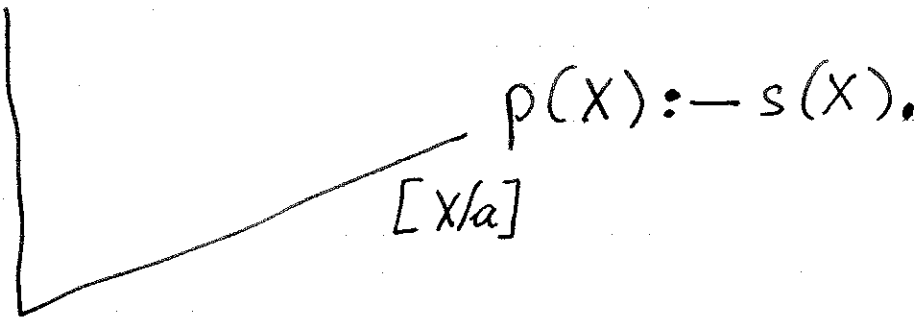
N ... rein negative Klauseln, Zielklauseln

D ... definite Hornklauseln (aus dem „Logik-Programm")

Sofern die Zielklausel in der SLD-Resolution aus mehreren negativen Literalen besteht, so wird das am weitesten links stehende für den nächsten Resolutionschritt herangezogen.

Beispiel:

?- p(a), q(Z, Y), r(Z, b).



?- s(a), q(Z, Y), r(Z, b)

Die Prolog-Auswertungsstrategie versucht einen SLD-Resolutionsbeweis aufzubauen. Dabei werden die definiten Hornklauseln (also die Klauseln des Logik-Programms) immer in der Reihenfolge von oben nach unten für einen möglichen Resolutionsschritt inspiziert.

Beachte: Dies birgt die Gefahr einer Unendlich-Schleife in sich, obwohl grundsätzlich ein

SLD-
Resolutionsbeweis existieren würde.

Beispiel:

```
a :- b.
b :- a.
b.
```

} Merke: erst die Fakten, dann die Regeln.

führt bei der Anfrage ?- a auf eine Unendlich-Schleife

```
?- a
 |_ a :- b.
?- b
 |_ b :- a
?- a
 |_ a :- b
?- b
 :
```

obwohl ein SLD-Beweis existiert:

```
? - a
 |_ a :- b.
?- b
 |_ b.
 □
```

Aufgabe: Gegeben ist das Logik-Programm

```
p(a).
r(f(f(X))) :- p(X).
r(f(Y)) :- r(Y).
```

Vollziehe nach, was der Prolog-Interpreter bei der Frageklausel ?- r(f(f(f(Z)))). macht.

Antwort: $?- r(f(f(f(Z)))).$

$r(f(f(X))) :- p(X)$
$[X/f(Z)]$

$?- p(f(Z)).$

Kein Resolvent möglich → Backtracking:

$?- r(f(f(f(Z)))).$

$r(f(Y)) :- r(Y).$
$[Y/f(f(Z))]$

$?- r(f(f(Z))).$

$r(f(f(X)) :- p(X).$
$[Z/X]$

$?- p(X).$

$p(a).$
$[X/a]$

□

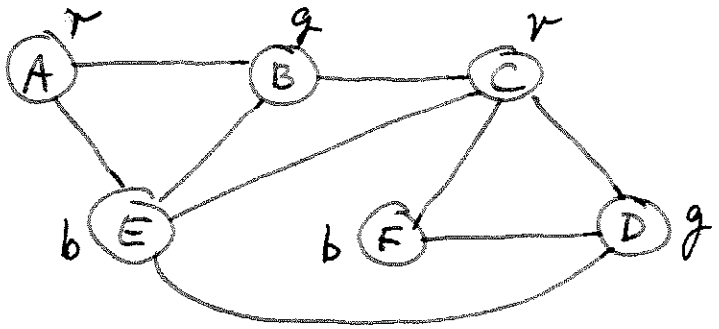
Antwort: yes, $Z = a$.

Bei Eingabe von Semikolon erhält man die weiteren Antworten $Z = f(a)$, $Z = f(f(a))$, $Z = f(f(f(a)))$, usw.

```
| ?- listing
p(a).
r(f(f(A))) :-
        p(A).
r(f(A)) :-
        r(A).

yes
| ?- r(f(f(f(Z)))).
Z = a ?;
Z = f(a) ?;
Z = f(f(a)) ?;
Z = f(f(f(a))) ?;
Z = f(f(f(f(a)))) ?;
Z = f(f(f(f(f(a))))) ?;
Z = f(f(f(f(f(f(a)))))) ?
```

Prolog-Programme, die das Backtracking ausnutzen:

Gegeben ein Graph



Ist dieser Graph 3-färbbar (ein NP-vollständiges Problem), mit den Farben rot, grün, blau?

Beschreibung des Graphen:

```
graph(A,B,C,D,E,F):-
        kante(A,B), kante(A,E),
        kante(B,E), kante(B,C),
        kante(C,D), kante(C,E),
        kante(C,F), kante(D,E),
        kante(D,F).
```

Zulässige Farben an einer Kante:

```
kante(r,g).
kante(r,b).
kante(g,b).
kante(g,r).
kante(b,r).
kante(b,g).
```

Die Anfrage `?-graph(A,B,C,D,E,F).` führt

auf die Antwort:

yes: $A=r$, $B=g$, $C=r$, $D=g$, $E=b$, $F=b$

Mittels Semikolon erhält man weitere mögliche Färbungen.

Prolog-Programm zum Lösen eines 3-SAT-Problems:

$$F = (A \vee B \vee \bar{C}) \wedge (\bar{B} \vee C \vee \bar{D}) \wedge (\bar{A} \vee \bar{B} \vee \bar{C}) \wedge (B \vee C \vee D) \wedge (\bar{A} \vee B \vee C)$$

Das Prolog-Programm:

```
inv(0,1).
inv(1,0).
```

} 0 und 1 sind invers zueinander

```
claus(0,0,1).
claus(0,1,0).
claus(1,0,0).
claus(0,1,1).
claus(1,0,1).
claus(1,1,0).
claus(1,1,1).
```

} zulässige Klauselbelegungen

```
formel(A,B,C,D) :- claus(A,B,NC), claus(NB,C,ND),
                   claus(NA,NB,NC), claus(B,C,D),
                   claus(NA,B,C), inv(A,NA),
                   inv(B,NB), inv(C,NC),
                   inv(D,ND).
```

Darstellung der Formel

Anfrage: `?- formel(A,B,C,D).`

yes: A=0, B=0, C=0, D=1 ;

A=0, B=1, C=1, D=1 usw.

Listen in Prolog

Prolog erlaubt den Umgang mit Listen, z.B.

`liste([a,b,c,d]).`

Hierbei wird der Ausdruck in eckigen Klammern System-intern als ein Term dargestellt:

`f(a, f(b, f(c, f(d, nil))))`

Hierbei ist f eine 2-stellige Funktion (der Listen-Konstruktor) und nil ist eine spezielle Konstante,

die die leere Liste, also [], bezeichnet.

Angenommen, L bezeichnet eine Liste, also einen Funktionsterm der Form f(...). Will man an L vorne ein Element a einfügen so schreibt man in Prolog dafür [a|L]

das erste Element der neuen Liste

die „Restliste"

Intern ist dies:

$$f(a, \underbrace{f(\ldots)}_{\text{entspricht } L})$$

append(L1, L2, L3):
Die Liste L_1 und L_2 sollen hintereinander gehängt werden, so dass die Ergebnisliste L_3 ist:

```
append( [], L, L ).
append( [A|B], L, [A|X] ) :- append(B, L, X).
```

```
?- append( [a, [b,c], d], [[d,e], f], Y).
```

yes: Y = [a, [b,c], d, [d,e], f]

```
?- append( [a,b,c], Z, [a,b,c,d,e] ).
```

yes: Z = [d,e]

Ein Programm zum Umdrehen einer Liste:

```
reverse(A,B) :- rev(A,[],B).

rev([],Z,Z).

rev([X|Y],U,V) :- rev(Y,[X|U],V).
```

Beispielanfrage: `?- reverse([a,b,c],Z).`

`yes: Z = [c,b,a].`

Eine Liste in jeder möglichen Weise permutieren:

```
permut([],[]).

permut([B|C],A) :- permut(C,D),
                   insert(B,D,A).

insert(A,B,[A|B]).

insert(A,[B|C],[B|D]) :- insert(A,C,D).
```

```
?- permut([a,b,c,d],X).
X = [a,b,c,d] ;
X = [b,a,c,d] ;
X = [b,c,a,d] ;
X = [b,c,d,a] ;
```

usw. alle $4! = 24$ Permutationen.

Andere Möglichkeit für reverse (mittels append):

```
reverse([], []).
reverse([X|Y], Z) :- reverse(Y, U),
                     append(U, [X], Z).
```

Die Länge einer Liste bestimmen in Form von s(...) - Iterationen (s... Nachfolgerfunktion).

```
länge([], a).
länge([X|Y], s(Z)) :- länge(Y, Z).
```

```
? - länge([a,b,c,d], Z).
yes, Z = s(s(s(s(a))))
```

Durchgang durch Themen der Logik-Vorlesung

(„Was kann in der Klausur drankommen?")

- Aussagenlogik: aus Wahrheitstafel DNF bzw KNF herleiten, Äquivalenzumformungen (Negationen nach „innen" mittels de Morgan, KNF/DNF herstellen mittels Distributivgesetz)
  Sprachliche Aussagen mit log. Formeln modellieren
  Begriffe: erfüllbar, Tautologie, erfüllbarkeitsäquiv. Umformung; Rechenaufwand bzw. Anzahl verschiedener Boole'scher Funktionen.
- Hornformeln (Fakten, Regeln, Zielklauseln), effizienter Erfüllbarkeitstest, 2KNF,
- Resolution: Definition, Widerlegungsvollständigkeit, Korrektheit, Folgerung, mögliche exponentielle Länge von Resolutionsbeweisen, Resolutions-Restriktionen und -Strategien, SLD-Resolution bei Hornformeln, Aussage des Endlichkeitssatzes.

- Prädikatenlogik: Quantoren, Strukturen, Terme, Signatur, Prädikat, Funktion(ssymbol), Konstante, freie/gebundene Variablen, bereinigen einer Formel, gebundene Umbenennung, Substitution, Pränexform herstellen durch geeignete Umformungen, Aussage des Überführungslemmas, Herstellen der Skolemform.
- Herbrand: H-Universum, H-Struktur, H-Expansion Satz von Herbrand; Algor. von Gilmore
- Mathematische Theorie: modelltheoretisch definiert versus axiomatisch definiert; Semi-Entscheidbarkeit; Aussage des Gödel'schen Satzes
- Prädikatenlog. Resolution, zunächst herleiten aus Grundresolution, Unifikationsalgorithmus (occur check), allgemeinster Unifikator, Aussage des Lifting-Lemmas,
- Prolog: andere Notation, SLD-Resolution, Prolog-Auswertungsstrategie, Substitution als „Antwort" des Prolog-Systems

Prolog-Programme erstellen:

- einfache Datalog-Programme (z.B. Verwandtschaftsbeziehungen)

- Prolog-Programme, die rudimentär „rechnen", wobei $D(F) = \{a, s(a), s(s(a)), \ldots\}$

$$\hat{=} \{0, 1, 2, \ldots\}$$

Beachte: n-stellige Funktionen $f: \mathbb{N}^n \to \mathbb{N}$ müssen in Prolog als $(n+1)$-stellige Prädikate $P \subseteq \{a, s(a), s(s(a)), \ldots\}^{n+1}$ definiert werden, wobei

$$f(x_1, \ldots, x_n) = y \iff P(\text{"}x_1\text{"}, \text{"}x_2\text{"}, \ldots, \text{"}x_n\text{"}, \text{"}y\text{"})$$
ist wahr

- Prolog-Programme, die Backtracking ausnutzen (das Prolog-Programm beschreibt nur die „Logik" der Aufgabenstellung)
- Listen $[a_1, \ldots, a_n]$ in Prolog – als alternative Schreibweise des Terms $f(a_1, f(a_2, \ldots, f(a_n, nil)\ldots))$; $[a|L]$ als Schreibweise für $f(a, L)$