Logik Grundvorlesung SS 18
Folien Teil 3: Logikprogrammierung

Gerhard Brewka

Institut für Informatik
Universität Leipzig
brewka@informatik.uni-leipzig.de

Gliederung Teil 3

3.1 Motivation

Grundidee: beschreibe Problem (deklarativ), nicht wie es gelöst wird.
Eingabe: Anfrage an Logikprogramm.
Ausführen des Logikprogramms: Herleitung von $\square$.
Rechenergebnis: Substitution der Variablen in Anfrage.

(1) $\mathit{liebt}(\mathit{Franz}, \mathit{RB})$
(2) $\mathit{liebt}(\mathit{Franz}, \mathit{BVB})$
(3) $\neg \mathit{liebt}(x, \mathit{BVB}) \vee \neg \mathit{liebt}(x, \mathit{RB}) \vee \mathit{liebt}(\mathit{Peter}, x)$

Aufruf: Gibt es jemanden, den Peter liebt? $\exists y\, \mathit{liebt}(\mathit{Peter}, y)$?
Herleitung der Antwort:

$\neg \mathit{liebt}(\mathit{Peter}, y)$	(3)	$\mathit{sub} = [x/y]$
$\neg \mathit{liebt}(y, \mathit{BVB}) \vee \neg \mathit{liebt}(y, \mathit{RB})$	(2)	$\mathit{sub} = [y/\mathit{Franz}]$
$\neg \mathit{liebt}(\mathit{Franz}, \mathit{RB})$	(1)	
$\square$		

Antwort: Ja, Franz (Substitution für y).

Invertierbarkeit der Parameter

Bisher: erstes Argument *liebt* Eingabe-, zweites Ausgabeparameter.
Wir können das einfach umdrehen (invertieren):

(1) $liebt(Franz, RB)$
(2) $liebt(Franz, BVB)$
(3) $\neg liebt(x, BVB) \vee \neg liebt(x, RB) \vee liebt(Peter, x)$

Aufruf: Gibt es jemanden, der Franz liebt? $\exists y\, liebt(y, Franz)$?

Herleitung der Antwort:

$\neg liebt(y, Franz)$		(3) $sub = [y/Peter, x/Franz]$
$\neg liebt(Franz, BVB) \vee \neg liebt(Franz, RB)$	(2)	
$\neg liebt(Franz, RB)$	(1)	
$\square$		

Antwort: Ja, Peter (Substitution für y).

Das Antwortprädikat

Statt relevante Substitution "zu suchen", verwende Antwortprädikat:

Neue Startklausel: $\neg liebt(y, Franz) \vee Ans(y)$?

Herleitung der Antwort:

$$\neg liebt(y, Franz) \vee Ans(y) \quad (3)$$
$$\neg liebt(Franz, BVB) \vee \neg liebt(Franz, RB) \vee Ans(Peter) \quad (2)$$
$$\neg liebt(Franz, RB) \vee Ans(Peter) \quad (1)$$
$$Ans(Peter)$$

Statt $\square$ wird Einer-Klausel mit Antwortprädikat hergeleitet.

Planungsbeispiel

$Has(t, s)$: Peter besitzt t in Situation s.
Aktionen: Funktionen, überführen Situation in Nachfolgesituation.
Anfrage: $\exists s\, Has(Bananas, s)$?

(1) $Has(Bananas, BuyBananas(s)) \vee \neg AtAldi(s) \vee \neg Has(Money, s)$
(2) $Has(Money, GetCash(s)) \vee \neg AtBank(s)$
(3) $AtBank(Go(Bank, s))$
(4) $AtAldi(Go(Aldi, s))$
(5) $Has(Money, Go(x, s)) \vee \neg Has(Money, s)$

$\neg Has(Bananas, s) \vee Ans(s)$ (1)
$\neg AtAldi(s) \vee \neg Has(Money, s) \vee Ans(BuyBananas(s))$ (4)
$\neg Has(Money, Go(Aldi, s)) \vee Ans(BuyBananas(Go(Aldi, s)))$ (5)
$\neg Has(Money, s) \vee Ans(BuyBananas(Go(Aldi, GetCash(s))))$ (2)
$\neg AtBank(s) \vee Ans(BuyBananas(Go(Aldi, getCash(Go(Bank, s)\ldots)$ (3)
$Ans(BuyBananas(Go(Aldi, getCash(Go(Bank, s)))))$

3.2 Definite Logikprogramme

- Programme, die nur definite Klauseln verwenden, heißen definit.
- Vorteil: Antworterzeugung einfach; SLD-Resolution vollständig.
- Darstellung in Regelform üblich: $P \leftarrow Q_1, ..., Q_k$.
- Fakten: Regeln ohne Vorbedingungen (Körper).
- Logikprogramm: endliche Menge von Regeln.
- Zielklausel: Regel ohne Kopf (negative Klausel).
- Prozedurale Interpretation von $P \leftarrow Q_1, ..., Q_k$:
  Um P abzuleiten, leite $Q_1, ..., Q_k$ ab.
- Werden nach Bedarf Klausel- wie Regelnotation verwenden.

SLD-Resolution

- Selective Linear Definite Resolution.
- Input-Resolution, beginnt mit negativer Klausel (Zielklausel).
- Resolviert mit nicht-negativer (= definiter) Programmklausel.
- Resolventen negative Klauseln, die weiter resolviert werden.
- Eingeführt von Robert Kowalski (1971).

Regeln vs. Disjunktionen vs. Klauseln

$a \leftarrow b, c$	$a \vee \neg b \vee \neg c$	$\{a, \neg b, \neg c\}$
$b \leftarrow c$	$b \vee \neg c$	$\{b, \neg c\}$
c	c	$\{c\}$

Gilt a?

(a) falls	$\neg a$	$\{\neg a\}$
(b, c) falls	$\neg b \vee \neg c$	$\{\neg b, \neg c\}$
(c) falls	$\neg c$	$\{\neg c\}$
$() = \mathit{true}$	false	$\square$

Konfiguration

Definition (Konfiguration, Konfigurationsübergang)

Sei F ein Logikprogramm. Eine Konfiguration ist ein Paar (G, sub) mit G Zielklausel und sub Substitution.

Es gelte $(G_1, sub_1) \vdash (G_2, sub_2)$ gdw.

1. es gibt eine Programmklausel K in F, die mit G_1 unter Verwendung des mgu s resolviert werden kann.
2. G_2 ist die Resolvente von K und G_1.
3. $sub_2 = sub_1 s$.

Berechnung

Definition (Berechnung)

Eine Berechnung von F bei Eingabe $G_1 = \{\neg A_1, ..., \neg A_k\}$ ist eine Konfigurationsfolge (G_i, sub_i), so dass $sub_1 = [\,]$ und für $j \geq 1$:

$$(G_j, sub_j) \vdash (G_{j+1}, sub_{j+1}).$$

Eine endliche, mit $(\square, sub)$ terminierende Berechnung heißt erfolgreich, $(A_1 \wedge ... \wedge A_k)sub$ ihr Ergebnis.

Anmerkungen:

Berechnungen nichtdeterministisch: Konfiguration kann verschiedene Nachfolgekonfigurationen haben.

Mögliche Berechnungen bilden Baum: endlich verzweigt, aber unendliche Pfade möglich.

Satz von Clark

Satz: Sei F Logikprogramm, $G = \{\neg A_1, ..., \neg A_k\}$ Zielklausel.

1. Korrektheit: Falls es eine erfolgreiche Berechnung von F bei Eingabe G gibt, so ist jede Grundinstanz des Ergebnisses der Berechnung Folgerung von F.
2. Vollständigkeit: Falls jede Grundinstanz von $(A_1 \wedge ... \wedge A_k)sub'$ Folgerung von F ist, so gibt es eine erfolgreiche Berechnung von F bei Eingabe G mit Ergebnis $(A_1 \wedge ... \wedge A_k)sub$, so dass für geeignetes s gilt: $(A_1 \wedge ... \wedge A_k)sub' = (A_1 \wedge ... \wedge A_k)sub\, s$.

Auswertungsstrategien

- Nichtdeterminismus: verschiedene Folgekonfigurationen möglich.
- Auswertungsstrategie: Reihenfolge der Ausführung von Schritten.
- Gründe für Nichtdeterminismus:
  1. Auswahl der Programmklausel, die für Resolution verwendet wird.
  2. Auswahl des nächsten bearbeiteten Literals der Zielklausel.

Reihenfolge der Abarbeitung der Teilziele irrelevant in folgendem Sinn:

Definition (kanonische Berechnung)

Eine Berechnung eines Logikprogrammes heißt kanonisch, falls bei jedem Konfigurationsübergang mit dem ersten Literal der Zielklausel resolviert wird (Klauseln als Listen aufgefasst).

Satz: Sei $(G, []) \vdash \ldots \vdash (\square, sub)$ eine erfolgreiche Berechnung von F. Dann gibt es eine erfolgreiche kanonische Berechnung von F bei Eingabe G derselben Länge und mit demselben Rechenergebnis.

Suche im Konfigurationsbaum

- Standardstrategien:
  1. Breitensuche (breadth first): alle Knoten im Baum der Tiefe t abgearbeitet, bevor Knoten der Tiefe $t + 1$ abgearbeitet wird.
  2. Tiefensuche (depth first): von den noch resolvierbaren Knoten der größten Tiefe jeweils der am weitesten links stehende Knoten abgearbeitet.
- Breitensuche vollständig: existierende Lösung wird gefunden; aber extrem ineffizient (exponentieller Speicherplatz).
- Tiefensuche unvollständig: Suche in unendlichen erfolglosen Pfaden; aber effizienter, deshalb in PROLOG verwendet.
- Programmierer muss Programme so schreiben, dass System nicht in Endlosschleife gerät.
- Widerspricht der Idee der deklarativen Programmierung.

Prolog-Algorithmus

Eingabe: $F = (K_1, ..., K_n)$, $K_i = B_i \leftarrow C_{i,1}, ..., C_{i,ni}$; Ziele $G = A_1, ..., A_m$.

```
success:= false;
eval(G,[]);
if success = false then write('nicht ableitbar');

procedure eval(G,sub);
if G = () then begin write((A_1 ∧ ... ∧ A_m)sub); success := true end
else begin      % es sei G = D_1, ..., D_k
      i:= 0;
      while (i < n) and not success do
            begin
            i:= i+1;
            if (D_1, B_i) unifizierbar mit mgu s
            then eval((C_{i,1}, ..., C_{i,ni}, D_2, ..., D_k)s, sub s)
            end;
      end;
```

Anmerkungen

- Breitensuche: im rekursiven Aufruf von eval Körper der resolvierten Klausel hinten statt vorne im 1. Argument einfügen.
- Alle Ergebnisse ausgeben: in while-Bedingung "and not success" weglassen.
- Prolog erlaubt auch negative Anfragen der Form *not A*.
- Keine negativen Literale aus definitem Programm folgerbar: Interpretation, die jedes Atom zu 1 auswertet, ist Modell.
- Aber oft sinnvoll anzunehmen, dass positive Information vollständig ist.
- *not A* wird als herleitbar betrachtet, wenn *A* nicht bewiesen werden kann.

Beispiel

1) $Vater(Peter, Hans)$
2) $Vater(Peter, Maria)$
3) $Vater(Hans, Uli)$
4) $Vater(Karl, Anton)$
5) $Mutter(Maria, Anna)$
6) $Opa(X, Y) \leftarrow Vater(X, Z), Vater(Z, Y)$
7) $Opa(X, Y) \leftarrow Vater(X, Z), Mutter(Z, Y)$

$(Opa(Peter, V))?$
$(Vater(Peter, Z), Vater(Z, V))$
$(Vater(Hans, V))$
$()$

6) $[X/Peter, Y/V]$
1) $[Z/Hans]$
3) $[V/Uli]$
erste Antwort: $V = Uli$

falls weitere Antworten gewünscht, Backtracking:
Rücksprung zu letzter Stelle mit alternativer Regel

$(Vater(Peter, Z), Vater(Z, V))$
$(Vater(Maria, V))$

2) $[Z/Maria]$
nicht erfolgreich

Beispiel Fortsetzung

1) $Vater(Peter, Hans)$
2) $Vater(Peter, Maria)$
3) $Vater(Hans, Uli)$
4) $Vater(Karl, Anton)$
5) $Mutter(Maria, Anna)$
6) $Opa(X, Y) \leftarrow Vater(X, Z), Vater(Z, Y)$
7) $Opa(X, Y) \leftarrow Vater(X, Z), Mutter(Z, Y)$

Backtracking:

$(Opa(Peter, V))$	7) $[X/Peter, Y/V]$
$(Vater(Peter, Z), Mutter(Z, V))$	1) $[Z/Hans]$
$(Mutter(Hans, V))$	nicht erfolgreich

Backtracking:

$(Vater(Peter, Z), Mutter(Z, V))$	2) $[Z/Maria]$
$(Mutter(Maria, V))$	5) $[V/Anna]$
$()$	zweite Antwort: $V = Anna$

Keine weitere Lösung.

3.3 Normale Logikprogramme

- Prolog lässt negative Anfragen zu.
- Warum nicht auch in Regelkörpern?
- Regeln in normalen Logikprogrammen haben die Form

$$A \leftarrow B_1, ..., B_n, \mathit{not}\ C_1, ..., \mathit{not}\ C_m$$

  wobei A, B_i und C_j Atome sind (evtl. mit freien Variablen).
- lies: A herleitbar, falls $B_1, ..., B_n$ herleitbar und $C_1, ..., C_m$ nicht herleitbar.
- Beachte "$\neg C$ herleitbar" vs. "C nicht herleitbar".
- "not" statt "$\neg$", um diese Unterscheidung deutlich zu machen.

Behandlung normaler Logikprogramme in Prolog

- Prozedurale Interpretation: negation as failure (SLDNF)
- Wenn negatives Ziel (*not L*) abgearbeitet wird, starte Beweis für *L*.
- Misslingt dieser Beweis, so ist *not L* wahr, sonst nicht.
- Hauptproblem: Programm läuft möglicher Weise endlos.

Beispiel:

$$a \leftarrow not\ b$$
$$b \leftarrow not\ a$$

Prozedurale Interpretation (SLDNF) liefert Endlosschleife:

Überprüfen von *b* liefert Teilziel *not a*, Beweis für *a* wird gestartet, liefert Teilziel *not b*, Beweis für *b* wird gestartet, ...

Semantik für *not*: Stabile Modelle

- Betrachten zunächst variablenfreie Programme (Variablen in 3.4).
- Ziel: modellbasierte Semantik für normale Logikprogramme.
- Beginnen mit definiten LPs, normale auf diese zurückgeführt.
- Modell repräsentiert als Menge der wahren Atome.

Definition (Stabiles Modell definiter Logikprogramme)

Sei P eine definites Logikprogramm. Das stabile Modell von P ist das kleinste Modell von P, wobei die Regeln in P als Implikationen gelesen werden und für Modelle M_1, M_2 gilt: $M_1 \leq M_2$ gdw. $M_1 \subseteq M_2$.

Beispiel: $a \leftarrow b, c \qquad c \leftarrow d \qquad d$

Modelle: $M_1 = \{d, c\}, M_2 = \{d, c, a\}, M_3 = \{d, c, b, a\}$

M_1 minimal, also stabiles Modell, damit "*not a*" und "*not b*" ableitbar, da a und b in M_1 falsch sind.

Stabile Modelle für normale Logikprogramme

Problem: es gibt nicht immer ein kleinstes Modell, sondern möglicher Weise mehrere minimale:

Beispiel:

$$a \leftarrow c, \mathit{not}\ b$$
$$b \leftarrow c, \mathit{not}\ a$$
$$c \leftarrow \mathit{not}\ d$$

3 minimale Modelle ($\mathit{not} \approx \neg$): $M_1 = \{a, c\}, M_2 = \{b, c\}, M_3 = \{d\}$

M_3 unerwünscht, da d nicht hergeleitet werden kann.

Intuition: nur solche Modelle M, in denen

1. alle anwendbaren Regeln angewendet wurden,
2. jedes wahre Atom eine nicht-zirkuläre Herleitung aus unwiderlegten Regeln hat.

Eine Regel ist widerlegt in M, wenn sie $\mathit{not}\ C$ im Körper hat mit $C \in M$.

Stabile Modelle

Idee: Modell raten und prüfen, ob mit den sich ergebenden Auswertungen der *not*-Literale dasselbe Modell herleitbar ist.

Definition (Stabiles Modell)

Sei P normales Logikprogramm, M Modell von P. Das um M reduzierte Programm, P^M, ergibt sich aus P durch

1. Streichen aller Regeln mit $(not\ C_i)$ im Körper und $C_i \in M$.
2. Streichen aller *not*-Literale aus allen anderen Regeln.

Das reduzierte Programm ist definit, hat also ein kleinstes Modell. M heißt stabil gdw. M kleinstes Modell von P^M ist.

Alle stabilen Modelle minimal, aber nicht jedes minimale Modell stabil.

Stabile Modelle

Beispiel

$$a \leftarrow c, \mathit{not}\ b$$
$$b \leftarrow c, \mathit{not}\ a$$
$$c \leftarrow \mathit{not}\ d$$

Minimale Modelle: $M_1 = \{a, c\}, M_2 = \{b, c\}, M_3 = \{d\}$

P^{M_1}: $a \leftarrow c$, c — kleinstes Modell: $\{a, c\} = M_1$

P^{M_2}: $b \leftarrow c$, c — kleinstes Modell: $\{b, c\} = M_2$

P^{M_3}: $a \leftarrow c$, $b \leftarrow c$ — kleinstes Modell: $\{\} \neq M_3$

Stabile Modelle

Weiteres Beispiel

$$a \leftarrow not\ b$$
$$b \leftarrow not\ a$$
$$c \leftarrow not\ c, a$$

Minimale Modelle: $M_1 = \{a, c\}, M_2 = \{b\}$

P^{M_1}:	a	kleinstes Modell: $\{a\} \neq M_1$
P^{M_2}:	b	kleinstes Modell: $\{b\} = M_2$
	$c \leftarrow a$	

Damit $\{b\}$ das einzige stabile Modell.

Es kann auch keine stabilen Modelle geben: $a \leftarrow not\ a$.

Stabile Modelle heißen auch Antwortmengen (answer sets).

3.4 Antwortmengenprogrammierung

- Englisch: Answer Set Programming (ASP)
- Grundidee: nicht Anfragen werden beantwortet, sondern die Antwortmengen selbst Gegenstand des Interesses.
- Repräsentiere Problem so, dass jede Antwortmenge Lösung des Problems enthält.
- Programme mit Variablen zur kompakten Repräsentation großer Regelmengen.
- *Grounding* ersetzt Regeln mit Variablen durch Grundinstanzen, erst danach *Solving*.
- Programme bestehen typischer Weise aus:
  1. Beschreibung der Probleminstanz.
  2. Generate-Teil, in dem Lösungskandidaten erzeugt werden.
  3. Test-Teil, in dem unerwünschte answer sets eliminiert werden.

Verwendung von Constraints

Sei P Programm, in dem Atom a nicht vorkommt. Hinzufügen einer Regel der Form:

$$a \leftarrow not\ a, b_1, ..., b_k$$

(Constraint) eliminiert Antwortmengen von P, in denen $b_1, ..., b_k$ gilt. Da a irrelevant ist (sofern es nicht in P vorkommt), schreiben wir:

$$\leftarrow b_1, ..., b_k$$

Beispiel: Färbeproblem

$node(1), ..., node(m)$ $edge(i,j), ..., edge(h,k)$	Probleminstanz (Graph)
$col(X, blau) \leftarrow not\ col(X, rot), not\ col(X, gelb)$ $col(X, rot) \leftarrow not\ col(X, blau), not\ col(X, gelb)$ $col(X, gelb) \leftarrow not\ col(X, rot), not\ col(X, blau)$	Generate
$\leftarrow col(X,Z), col(Y,Z), edge(X,Y)$	Test

Alle answer sets enthalten eine Lösung des Färbeproblems.

Punktnotation

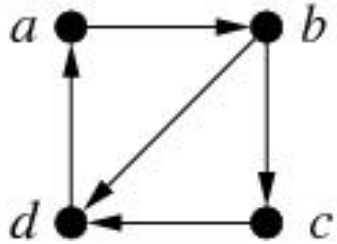
- Bisher Fakten/Regeln durch Zeilenumbruch getrennt.
- Bei wenig Platz/langen Regeln Verwechslung möglich:
  - trennt Komma Vorbedingungen im Körper einer Regel?
  - oder verschiedene Regeln in einer Menge von Regeln?
  - Enthält $\{a \leftarrow b, c\}$ eine Regel mit Körper b, c oder eine Regel und einen Fakt c?
- Deshalb häufig Punkt am Ende eines Fakts, einer Regel:

$$\{a \leftarrow b, c.\} \text{ vs. } \{a \leftarrow b., c.\}.$$

- Auch hier in den weiteren Beispielen.

Hamiltonkreis

Kreis im Graphen, der jeden Knoten genau einmal besucht.



Graph

$vtx(a).vtx(b).vtx(c).vtx(d).$
$edge(a,b).edge(b,c).edge(c,d).edge(d,a).edge(b,d).$

Generate

$in(X,Y) \leftarrow edge(X,Y), not\ out(X,Y).$
$out(X,Y) \leftarrow edge(X,Y), not\ in(X,Y).$

Test

$\leftarrow in(V_2,V_1), in(V_3,V_1), V_2 \neq V_3.$
$\leftarrow in(V_1,V_2), in(V_1,V_3), V_2 \neq V_3.$
$reachble(V,V).$
$reachble(V_1,V_3) \leftarrow in(V_1,V_2), reachble(V_2,V_3).$
$\leftarrow vtx(V_1), vtx(V_2), not\ reachble(V_1,V_2).$

Organisation von Treffen

Beispiel

Probleminstanz:

$meeting(m_1). \ldots meeting(m_n).$
$time(t_1). \ldots time(t_s).$
$room(r_1). \ldots room(r_m).$
$person(p_1). \ldots person(p_k).$
$par(p_1, m_1). \ldots par(p_2, m_3). \ldots$

Generate:

$at(M, T) \leftarrow meeting(M), time(T), not\ notat(M, T).$
$notat(M, T) \leftarrow meeting(M), time(T), not\ at(M, T).$
$in(M, R) \leftarrow meeting(M), room(R), not\ notin(M, R).$
$notin(M, R) \leftarrow meeting(M), room(R), not\ in(M, R).$

Organisation von Treffen

Beispiel, Test

Jedem Treffen ist Raum und Zeit zugeordnet:

$timeassigned(M) \leftarrow at(M, T).$
$roomassigned(M) \leftarrow in(M, R).$
$\leftarrow meeting(M), not\ timeassigned(M).$
$\leftarrow meeting(M), not\ roomassigned(M).$

Kein Treffen hat mehr als 1 Zeit und 1 Raum:

$\leftarrow meeting(M), at(M, T), at(M, T'), T \neq T'.$
$\leftarrow meeting(M), in(M, R), in(M, R'), R \neq R'.$

Gleichzeitige Treffen brauchen verschiedene Räume:

$\leftarrow in(M, X), in(M', X), at(M, T), at(M', T), M \neq M'.$

Treffen mit den selben Personen brauchen verschiedene Zeiten:

$\leftarrow par(P, M), par(P, M'), M \neq M', at(M, T), at(M', T).$

4. Weitere Logikthemen: Ausblick

Nichtmontones Schließen: Circumscription (John McCarthy)

- Verwendung von *ab*-Prädikaten zur Modellierung von Defaults:

$$\forall x.bird(x) \wedge \neg ab(x) \rightarrow flies(x).$$

- Pro Default eigenes *ab*-Prädikat (unterschieden durch Indizes).
- Reicht noch nicht: Semantik so zu definieren, dass "unnormale" Modelle außen vor bleiben.
- Grundidee: nicht alle Modelle berücksichtigen, sondern nur minimale, d.h. solche, in denen möglichst wenig *ab*-normal ist.
- Aussagenlogik: möglichst wenige ab_i wahr; Prädikatenlogik: Extensionen der ab_i möglichst klein.

Circumscription

Beispiel

$KB = \{bird, bird \wedge \neg ab \rightarrow flies\}$

Modelle:

$M_1 = \{bird, ab, flies\}$, $M_2 = \{bird, ab, \neg flies\}$, $M_3 = \{bird, \neg ab, flies\}$

- M_1 und M_2 enthalten Abnormalität.
- Nur in M_3 ist nichts abnormal.
- Focus auf Modellen, die möglichst normale Situationen repräsentieren.
- Akzeptiere Formel, wenn sie wahr ist in diesen Modellen: hier *flies*.

Circumscription

Prädikatenlogik:

- Gegeben 2 Strukturen I_1 und I_2 mit derselben Domäne. Es sei

  $I_1 \leq I_2$ gdw. $I_1[ab_i] \subseteq I_2[ab_i]$ für jedes ab-Prädikat ab_i,
  $I_1 < I_2$ gdw. $I_1 \leq I_2$ aber nicht $I_2 \leq I_1$.

- Definiere neue Folgerungsrelation:

  $KB \models_{\leq} \alpha$ gdw. für jedes I gilt:
  $I \models \alpha$ falls $I \models KB$ und es gibt kein $I' < I$ mit $I' \models KB$.

- α muss also wahr sein in allen Modellen von KB, die bezüglich Abnormalitäten minimal sind.

Aussagenlogik:

I_1, I_2 Modelle. $I_1 < I_2$ gdw. $\{Ab_i \mid I_1(Ab_i) = 1\} \subsetneq \{Ab_i \mid I_2(Ab_i) = 1\}$.

Rest wie oben.

Circumscription

- Warum ist das nichtmonoton?
- Zusätzliche Information kann Modelle eliminieren.
- Zu prüfen sind die normalsten unter den verbleibenden.

Beispiel

$KB = \{bird, bird \wedge \neg ab \rightarrow flies, ab\}$

Modelle:

$M_1 = \{bird, ab, flies\}$, $M_2 = \{bird, ab, \neg flies\}$, M_3 kein Modell mehr.

- Sowohl M_1 wie M_2 jetzt so normal wie möglich.
- *flies* nicht mehr in allen normalsten Modellen.

Mehrwertige Logiken

- Bisher: Sätze wahr (**1**) oder falsch (**0**), tertium non datur.
- Man kann auch weitere Werte zulassen, z.B. undefiniert (**u**).
- Verwendet, um etwa paradoxen Sätzen Wert zu geben.
- Oder um Widersprüche zu behandeln:

  Auch Interpretationen, die **u** zuweisen, als Modelle betrachten $\Rightarrow$ Modelle auch für klassisch inkonsistente Formelmengen.
- Häufig auch Werte im Intervall $[\mathbf{0}, \mathbf{1}]$ betrachtet.
- Anwendung: Sätze mit vagen Begriffen (groß, schnell, häufig, ...).
- Wann ist jemand groß? Unter 1,70: **0**; über 1,85: **1**; fließender Übergang von **0** bis **1** bei Größen dazwischen.

Prädikatenlogik 2. Stufe

- Bisher: Quantifikation über Elemente der Domäne.
- Jetzt: Quantifikation auch über Funktionen und Prädikate.
- Dazu Einführung von Funktions- und Prädikatsvariablen.
- Erweiterung der Semantik einfach:

  Funktions- und Prädikatsvariablen zu Funktionen und Prädikaten über der jeweiligen Domäne interpretieren.
- Aber: Semi-Entscheidbarkeit geht verloren.
- Notwendig zur Formalisierung induktiver Definitionen.

Prädikatenlogik 2. Stufe, Beispiel

Natürliche Zahlen, repräsentiert durch Nachfolger-Funktion s:

$$Nat = \{0, s(0), s(s(0)), ...\}$$

Definition (induktiv)

Nat ist die kleinste Menge, für die gilt:

1. $0 \in Nat$.
2. Falls $k \in Nat$, so ist $s(k) \in Nat$.

Definition (Logik 2. Stufe)

Sei x Individuenvariable, p Prädikatsvariable:

$$\forall x\,(Nat(x) \leftrightarrow \forall p\,(p(0) \wedge \forall y(p(y) \rightarrow p(s(y))) \rightarrow p(x))).$$

Grundidee: x in kleinster Menge, die (1),(2) erfüllt, gdw. x in allen Mengen, die (1),(2) erfüllen.

Modallogiken

- Sätze können über andere Sätze sprechen, z.B.:
  - Es ist notwendig/möglich, dass p.
  - John weiß, dass p.
  - John glaubt, dass p.
  - Immer/irgendwann (in der Zukunft) gilt p.
  - p ist verboten/erlaubt.
- Modellierung durch Verwendung von Operatoren: $\Box p$, $\Diamond p$.
- Können geschachtelt auftreten (überall wo $\neg$ stehen könnte).
- Es gilt $\Box p \equiv \neg\Diamond\neg p$.
- Semantik der Operatoren nicht wahrheitsfunktional definierbar.

Modallogiken: Semantik

- Verwendung einer einzigen Interpretation nicht ausreichend.
- Betrachte Menge W von Interpretationen, hier Welten genannt.
- Zusätzlich: Erreichbarkeitsrelation R zwischen Welten
- Grundidee:
  - Formel p ohne Modaloperatoren wahr in Welt w, wenn entsprechende Interpretation sie zu 1 auswertet.
  - $\Box p$ wahr in w, wenn p wahr ist in allen von w erreichbaren Welten.
  - $\Diamond p$ wahr in w, wenn p wahr ist in einer von w erreichbaren Welt.
- Verschiedene Lesarten der Operatoren durch Festlegung von Eigenschaften von R.

Wie entsteht die passende Modallogik?

- Verschiedene Interpretationen der Modaloperatoren erfordern Gültigkeit bestimmter Axiome.
- Kann durch Eigenschaften von R erreicht werden.
- Beispiel: **T** : $\Box\phi \rightarrow \phi$, für alle ϕ.
- Nicht sinnvoll, wenn $\Box$ als "es wird geglaubt, dass" interpretiert wird.
- Aber sinnvoll, wenn $\Box$ "es wird gewusst, dass" bedeutet.
- Gültigkeit von **T** wird erreicht Reflexivität von R.
- Baukasten zur Konstruktion der geeignetsten Modallogik.

Zu guter Letzt

- Kommende Woche: Klausurvorbereitung (Dr. Loebe)
- Bedanke mich für die Aufmerksamkeit.
- Es war mir ein Vergnügen.

Zu guter Letzt

- Kommende Woche: Klausurvorbereitung (Dr. Loebe)
- Bedanke mich für die Aufmerksamkeit.
- Es war mir ein Vergnügen.