Wissensbasierte Systeme I

gehalten im WS 04/05 von G. Brewka

Mitschrift von Stefan Endrullis

17. Februar 2005

Inhaltsverzeichnis

1 Einführung: Was will die KI?

2 Problemlösen und Suche

2.1 Heuristische Verfahren

versuchen optimale (oder annähernd optimale) Lösung in exponentiell großem Lösungsraum durch problemspezifische Information zu finden.

I. Konstruktionsverfahren: Ansätze, die Lösung schrittweise zu konstruieren (A^*, Greedy Algorithmen)

II. Verbesserungsverfahren: starten mit zufällig gewählter Lösung, versuchen diese zu verbessern (Hill Climbing, Genetische Alg.)

2.1.1 Greedy Algorithmen

gegeben: Güteverfahren w, das für (Teil-)Lösungen optimale Lösung konstruiert durch schrittweise Erweitern der vorliegenden Teillösung. Von allen Erweiterungsmöglichkeiten wird die gewählt, die zum großten w-Wert führt.

Beispiel 1: (Auftragsplanung)

gegeben: Menge A von n Aufträgen, jeweils 1 Zeiteinheit, zu jedem Auftrag i gibt es Gewinn p_i und deadline d_i.

gesucht: Menge M von Aufträgen, so dass
1) M kann so sortiert werden, dass jeder Auftrag vor deadline erledigt
2) Gesamtgewinn maximal

Job	Gewinn	Deadline
1	20	2
2	25	2
3	10	1
4	5	3

$A = \{1, 2, 4\}$, Gewinn: $20 + 25 + 5 = 50$, greedy sucht dickste Brocken: 2, 1, 4

Algorithmus 1:

Ordne die Aufträge nach absteigendem Gewinn: $p_1 \geq p_2 \geq \cdots \geq p_m$
$A := \{\}$
for $i := 1$ to n do
  if $A \cup \{i\}$ ist zulässige Lösung then $A := A \cup \{i\}$
Gib A aus

Sei E endliche Menge, U Menge von Teilmengen von E.

(E, U) heißt Teilmengensystem, falls gilt:

1. $\emptyset \in U$,
2. $A \subseteq B, B \in U \Rightarrow A \in U$.

(E, U) heißt Matroid, wenn gilt:

$$X, Y \in U, |X| < |Y| \Rightarrow \exists x \in Y - X : X \cup \{x\} \in U$$

Satz 1:

Sei (E, U) ein Teilmengensystem. Der kanonische Greedy-Alg. liefert für das zugehörige Optimierungsproblem (bzgl. Wertfunktion $w : E \to \mathbb{R}^+$) die optimale Lösung, gdw. (E, U) ein Matroitd ist.

unser Bsp.: E: Menge von Aufträgen A
U: Menge $L(A)$ von Lösungen, d.h. Auftragsmengen, so dass Sortierung entsprechend deadling möglich.

Matroideigenschaft kann einfach gezeigt werden.

2.1.2 Hill-Clinbing

1. starte mit beliebiger vollständiger Lösung l
2. modifiziere l nach bestimten Regeln zu l'
3. teste, ob l' besser als l, wenn ja ersetze l durch l'
   Falls Abbruchkriterium erreicht, gib l aus, sonst gehe zu 2.

Ebenen von Lösungen zur Vermeidung des Steckenbleibens im lokalen Maximum: mehrfacher Neustart mit zufälliger Anfangslösung.

Beispiel 2: (TSP)

Annahme : Entferungsmatrix sysmmetrisch ($M[x, y]$ gibt Entferung zw. x, y an)

Initialisierung: zufällig erzeugte Permutationen der Städte

lokale Verbesserung: wähle 2 Städte v_i, v_j zufällig (o.B.d.A. $i < j$)
falls $M[v_i, v_{i+1}] + M[v_j, v_{j+1}] > M[v_i, v_j] + M[v_{i+1}, v_{j+1}]$
dann ersetze bisherige Rundreise durch:
$v_1, v_2, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+2}, v_{i+1}, v_{j+1}, \ldots, v_n$.

3 Logik und Inferenz

3.1 Repräsentation von Wissen in PL1

1. unvollständiges Wissen

$$\frac{Informatiker(Peter) \vee Mathematiker(Peter) \quad \neg Chemiker(Peter)}{\exists x[Informatiker(x) \wedge Chemiker(x)]}$$

2. terminologoische Zusammenhänge

Definitionen:	$\forall x[RichMan(x) \leftrightarrow Rich(x) \wedge Man(x)]$
Subsumtion (Oberbegriffe):	$\forall x[Man(x) \rightarrow Human(x)]$
Disjunktheit:	$\forall x[Man(x) \rightarrow \neg Woman(x)]$

3. Abstrakte Objekte

   $Kauft(Peter, PC, Aldi, 899, 26Nov, \dots)$ wieviele Argumente braucht Kauft?

   besser: abstraktes Objekt vom Typ $Kauft$ mit Namen (z.B. $K23$)
   $Kauft(K23), Agent(K23) = Peter, Objekt(K23) = PC, Laden(K23) = Aldi$

4. Zeitliche Veränderungen

   Küche erst rot, nach streichen grün. Idee (Situationskalkül): Situation, in der Aussage gilt, als zusätzliches Argument:
   $Farbe(Kueche, S_0) = rot, Result(KuecheStreichen, S_0) = S_1$
   $Farbe(Kueche, S_1) = gruen$

5. Reifikation

   Umwandeln eines Prädikats (od. Funktion) in Objekt:
   $Element(t_1, Birnen)$ statt $Birne(t_1)$
   $Teilmenge(Birnen, Fruechte)$ statt $\forall x[Birne(x) \rightarrow Frucht(x)]$

   !! Setzt geeignete Formalisierungen der Prädikate Element, Teilmenge voraus!!
   etwa $\forall x, y[Element(x, y) \wedge Teilmenge(y, z) \rightarrow Element(x, z)]$

3.2 Schließen mit Horn-Klauseln

Horn-Klausel: Disjunktion mit höchstens 1 pos. Literal

Beispiel 3:

$\neg Kind \vee \neg Maennlich \vee Junge$

Mengentheoretisch: $\{\neg Kind, \neg Maennlich, Junge\}$

Implikation (Regel): $Kind \wedge Maennlich \Rightarrow Junge$

neg. Horn-Klausel: kein pos. Literal
pos. Horn-Klausel: 1 pos. Literal

Beobachtung bei Resolution mit Horn-Klauseln:

1. es entstehen nur Horn-Kauseln
2. es ist immer eine pos. Horn-Klausel beteiligt

Man kann zeigen: Sei S Menge von Horn-Klauseln, c neg. Klausel (z.B. $\square$). Wenn c mit Resolution aus S ableitbar ist, dann gibt es auch eine Abteilung von c, so dass jede Resolution eine negative Klausel ist, die aus der zuletzt abgeleiteten Klausel und einem Element aus S entsteht.

$$\begin{array}{rcl} \text{aus } S & \rightarrow & c_1 \\ & & \downarrow \\ S_1 & \rightarrow & c_2 \\ & & \downarrow \\ S_2 & \rightarrow & c_3 \\ & & \downarrow \\ S_{n-1} & \rightarrow & c_n = c \end{array}$$

Definition 1: (SLD-Ableitung)

Eine SLD-Ableitung einer Klausel c aus einer Menge von Klauseln S ist eine Folge $c_1, \ldots, c_n$, so dass

1. $c_n = c$
2. $c_1 \in S$
3. für alle $i < n$ gilt: c_{i+1} ist Resolvente aus c_i und einem Element von S

Satz 2:

SLD-Ableitungen sind widerlegungsvollständig (und korrekt) für Horn-Klauseln:

S widersprüchlich gdw. $S \vdash_{SLD} \square$,

$S \vdash_{SLD} F$ gdw. es SLD-Ableitung von F aus S gibt.

Beispiel 4:

Klauseln	**Regel**
$Toddler$	$Toddler$
$\neg Toddler \vee Child$	$Toddler \Rightarrow Child$
$\neg Child \vee \neg Male \vee Boy$	$Child \wedge Male \Rightarrow Boy$
$\neg Infant \vee Child$	$Infant \Rightarrow Child$
$\neg Child \vee \neg Female \vee Girl$	$Child \wedge Female \Rightarrow Girl$
$Female$	$Female$

Wid.beweis von Girl: Entspricht Finden von Regeln für offene Teilziele

Klausel 1	**Klausel 2**	**Ziel**
$\neg Child \vee \neg Female \vee Girl$	$\neg Girl$	
$\neg Toddler \vee Child$	$\neg Child \vee \neg Female$	$Girl ... Child, Female$
$Toddler$	$\neg Toddler \vee \neg Female$	$Toddler, Female ... Female$
$Female$	$\neg Female$	kein weiteres Teilziel... Erfolg
	$\square$	

Beweis 1: (rek. Beweisprozedur)

input: endliche Liste von Atomen $q_1, \ldots, q_n$

output: JA, falls alle q_i aus gegebener KB [1] folgerbar, NEIN sonst

```
procedure SOLVE[q_1, ..., q_n]
  if n = 0 then return JA;
  for each klausel c ∈ KB do
    if c = p_1 ∧ ... ∧ p_m ⇒ q_1 und SOLVE[p_1, ..., p_m, q_2, ..., q_n]
    then return JA;
  end for;
  return NO;
}
```

backward chaning: von Ziel zu Prämissen (top down)

forward chaning: starte mit Fakten, wende Regel an, deren Vorbedingung bereits abgeleitet, Iteriere bis nichts neues mehr passiert

Formel: Sei KB Menge von Horn-Klauseln. Sei At Menge von Atomen in KB.

Definiere: $T_{KB} : 2^{At} \Rightarrow 2^{At}$ wie folgt:
$T_{KB}(A) = \{\ q \mid p_1 \wedge \ldots \wedge p_m \Rightarrow q \in KB, p_1, \ldots, p_m \in A\ \}$
$T^0_{KB} = \emptyset$
$T^{i+1}_{KB} = T_{KB} \cdot (T^i_{KB})$

Menge von Atomen A heißt Hülle von KB, wenn $A = T^{i+1}_{KB} = T^i_{KB}$
(in anderen Worten, wenn A kleinster Fixypunkt von T_{KB})

Beispiel 5: (Toddler)

$T^0_{KB} = \emptyset$
$T^1_{KB} = T_{KB}(\emptyset) = \{Toddler, Female\}$
$T^2_{KB} = T_{KB}(T^1_{KB}) = \{Toddler, Female, Child\}$
$T^3_{KB} = \cdots = \{Toddler, Female, Child, Girl\} \leftarrow$ Hülle aller folgerbaren Atome
$T^4_{KB} = T^3_{KB}$

backward chaining ist zielgerichtet, aber schon im aussagenlogischen Fall nicht immer Terminierung.
z.B. $p \Rightarrow p \in KB \quad \rightarrow \quad$ Algorithmus ersetzt immer wieder p durch p.

Beispiel 6: (forward wesentlich besser als backward)

$2n$ Atome: $p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1}$

$4n-4$ Regeln:
$p_0 \Rightarrow p_1, \ldots, p_{n-2} \Rightarrow p_{n-1}$
$p_0 \Rightarrow q_1, \ldots, p_{n-2} \Rightarrow q_{n-1}$
$q_0 \Rightarrow p_1, \ldots, q_{n-2} \Rightarrow p_{n-1}$
$q_0 \Rightarrow q_1, \ldots, q_{n-2} \Rightarrow q_{n-1}$

`SOLVE[`q_i`]` und `SOLVE[`p_i`]` liefern jeweils NO, aber erst nach 2^i Schritten (=Anfang von SOLVE). Einfacher Induktionsbeweis für p_i:
$i = 1$: ruft SOLVE[p_0] und SOLVE[q_0], also 2^1 Aufrufe
$i > 1$: ruft SOLVE[p_{i-1}] und SOLVE[q_{i-1}], also 2^{i-1} Aufrufe.

[1] Menge von Horn-Klauseln; B steht für Basis, denke ich; K vielleicht für Konklusion oder Klausel

forward terminitert sofort: $T_{KB}(\emptyset) = \emptyset$

Terminologie (T-Box)	Aussagen über best. Objekte (A-Box)
Def. der wichtigen Konzepte einer Domäne Zusammenhänge zw. Konzepten	Anwendung von Konzepten auf Objekte
Logik: $\forall x[Bachelor(x) \leftrightarrow Male(x) \wedge \neg Married(x)]$ Konzept n-stelliges Prädikat, Formel mit 1 freien Variable	$Bachelor(Peter)$
Rolle: 2-stelliges Prädikat $\exists child, Male$ logisch: $\forall x(\exists child, Male)(x) \leftrightarrow \exists y, child(x, y) \wedge Male(y)$ dabei ist *child* eine Rolle, $Male$ ein Konzept	

Es gibt viele DLS:

- Konstruktoren für komplexe Konzepte
- Konstruktoren für komplexe Rollen
- konkrete DL entsteht durch Auswahl der Konstruktoren

Semantik:

Konzept $\rightarrow$ 1-stelliges Prädikat $\rightarrow$ Teilmenge der Domäne D

auch möglich nicht über Prädikatenlogik zu gehen:
Konzept $\rightarrow$ Teilmenge der Domäne D

Rolle $\rightarrow$ 2-stelliges Prädikat $\rightarrow$ Teilmenge der Domäne D

4 Nichtmonotones Schließen

Klassische Logik monoton: mehr Prämissen haben nie weniger Konklusionen

$$Th(T) = \{\, p \mid T \models p \,\}$$

$$T_1 \subset \underbrace{T_2}_{\text{weniger Modelle}} \Rightarrow Th(T_1) \subset Th(T_2)$$

1. Beschreibung typischer Zusammenhänge

   Wir wissen: Vögel können (normalerweise) fliegen $\neg\forall x Vogel(x) \rightarrow Fliegt(x)$
   Tweety ist Vogel

   $\forall x Pinguin(x) \rightarrow \neg Fliegt(x)$ $[\forall x Fliegt(x) \rightarrow \neg Pinguin(x)]$
   $\forall x Pinguin(x) \rightarrow Vogel(x)$
   $\forall x Vogel(x) \wedge \neg Ausnahme(x) \rightarrow Fliegt(x)$
   $Vogel(Tweety)$

   $\forall x Ausnahme(x) \rightarrow Pinguin(x) \vee Strauss(x) \vee Gestutzt(x) \vee \ldots$

   Problem:

wir können nur dann $Fliegt(Tweety)$ ableiten, wenn wir auch $\neg Pinguin(Tweety), \neg Strauss(Tweety), \neg Gestutzt(Tweety), \ldots$ in die Regeln aufnehmen

D.h. man möchte folgendes ableiten bzw. nicht ableiten können:

$$W \cup \{Vogel(Tweety)\} \mid\!\sim Fliegt(Tweety)$$

$$W \cup \{???)\} \not\mid\!\sim Fliegt(Tweety)$$

(relativ) einfache Lösung des Problems in Logikprogrammierung:
neue Art von Negation (default negation) not (zu unterscheiden von $\neg$)
not $Ausnahme(Tweety)$ wahr, wenn $Ausnahme(Tweety)$ nicht hergeleitet werden kann.

$Fliegt(X) \leftarrow Vogel(X), \text{not } Ausnahme(X)$
$Ausnahme(X) \leftarrow Pinguin(X)$
$Ausnahme(X) \leftarrow Strauss(X)$
...

Vorgehen: not $Ausnahme(X)$ ist dann wahr, wenn nicht bewiesen werden kann, daß X eine Ausnahme ist.

$Vogel(Tweety) \quad \mid\!\sim Fliegt(Tweety)$

$Pinguin(Tweety) \quad \not\mid\!\sim Fliegt(Tweety)$

4.1 Answer Set Programming

4.1.1 Definite Logik-Programme

Definition 2: (Definites Logik-Programm)

Ein definites Logik-Programm ist eine endliche Menge von Regeln der Form

$$A \leftarrow B_1, \ldots, B_n$$

wobei A, B_i Atome sind. A wird als Kopf (head) bezeichnet und die B_i bilden den Körper (body) der Regel. Falls $n = 0$ heißt die Regl Fakt und $\leftarrow$ kann weggelassen werden.

Beispiel 7:

Programm:

1. $r \leftarrow s$
2. $q \leftarrow p$
3. p

acceptable set of beliefes?:

- $\{p\}$ Nein, 2. nicht angewendet
- $\{p, q, \underline{r}\}$ Nein, kein Grund für r

- $\{p, q\}$ Ja, alle anwendbaren Regeln wurden auch benutzt, keine unbegründeten beliefes

mögliche Ableitungen:

- 3, 2 für q
- 3 für p

4.1.2 Answer Sets definiter Logik-Programme

Definition 3: (Answer Sets definiter Logik-Programme)

Sei S eine Menge von Atomen, P ein definites Logik-Programm.

- S ist abgeschlossen (closed) unter P, wenn $A \in S$, so oft $A \leftarrow B_1, \ldots, B_n \in P$ und $B_1, \ldots, B_n \in S$
- S ist gegründet (grounded) in P, wenn $A \in S$ impliziert, daß es eine Ableitung für A aus P gibt

Answer set: eindeutige Menge von Atomen, die in P abgeschlossen und gegründed sind; wird mit $Cn(P)$ bezeichnet

4.1.3 Normale Logik-Programme

Definition 4: (Normales Logik-Programm)

Ein normales Logik-Programm ist eine endliche Menge von Regeln der Form

$$A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m$$

wobei A, B_i, C_j Atome sind.

Beispiel 8:

1. $p \leftarrow r, \text{not } q$
2. $q \leftarrow r, \text{not } p$
3. $r \leftarrow \text{not } s$

acceptable set of beliefes?:

- $\{r\}$ Nein, weil 1, 2 nicht angewendet wurden
- $\{r, p, q\}$ Nein, zwar abgeschlossen, aber nicht gegründet, da 1, 2 wiederlegt
- $\{s\}$ Nein, nicht grounded, aber ist minimal closed, keine Regel kann angewendet werden
- $\{r, p\}$ Ja, da closed(1) und grounded
- $\{r, q\}$ Ja, da closed(2) und grounded

mögliche Ableitungen:

- 3, 2 für q
- 3 für r

Was man am Ende glaubt, ist das, was in allen akzeptierenden beliefes ist (r).

4.1.4 Answer Sets normaler Logik-Programme

Definition 5: (Answer Sets normaler Logik-Programme)

Sei S eine Menge von Atomen, P ein normales Logik-Programm.

- S ist abgeschlossen (closed) unter P, wenn $A \in S$, so oft
  $A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m \in P,\ B_1, \ldots, B_n \in S$ und $C_1, \ldots, C_m \notin S$
- S ist gegründet (grounded) in P, wenn $A \in S$ impliziert, daß es eine Ableitung für A aus P gibt, die in S gültig ist

Answer sets: Mengen von Atomen, die in P abgeschlossen und gegründed sind (auch stabile Modelle genannt)

4.1.5 Erweiterte Logik-Programme

Definition 6: (Erweitertes Logik-Programm)

Regeln haben die Form

$$A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m$$

wobei A, B_i, C_j **Literale** sind.

D.h. es gibt 2 Arten von Negationen:

- klassische Negation: $\neg$
- Default-Negation: not

4.1.6 Answer Sets erweiterter Logik-Programme

Definition 7: (Answer Sets erweiterter Logik-Programme)

Sei S eine Menge von Literalen, P ein erweitertes Logik-Programm.

- S ist abgeschlossen (closed) unter P, wenn $A \in S$, so oft
  $A \leftarrow B_1, \ldots, B_n, \text{not } C_1, \ldots, \text{not } C_m \in P,\ B_1, \ldots, B_n \in S$ und $C_1, \ldots, C_m \notin S$, oder
  $L, \neg L \in S$ für das gleiche L
- S ist gegründet (grounded) in P, wenn $A \in S$ impliziert, daß es eine Ableitung für A aus P gibt, die in S gültig ist

Answer sets: Mengen von Literalen, die in P abgeschlossen und gegründed sind

Bemerkung: (Test eines Answer Sets)

Um zu testen, ob S ein Answer Set von P ist

- generate the S-reduct P^S of P:
  1. lösche Regeln mit not C_i im Körper und $C_i \in S$
  2. lösche alle not -Literale in den übrigen Reglen
- überprüfe, ob $S = Cn(P^S)$

Beispiel 9:

Sei P folgendes Programm:

1. $p \leftarrow r, \text{not } q$
2. $q \leftarrow r, \text{not } p$
3. $r \leftarrow \text{not } s$

Sei $S = \{p, r\}$. Dann ist P^S:

1. $p \leftarrow r$
2. r

$\{p, r\} = \{r, p\} \Rightarrow$ answer set

Sei $S' = \{r, p, q\}$. Dann ist $P^{S'}$:

1. r

$\{r, p, q\} \neq \{r\} \Rightarrow$ kein answer set

4.2 Motivation II: Situationskalkül

Situationen:	Weltzustände, entsprechen Folgen von Aktionen
Fluents:	veränderliche Eigenschaften
Aktionen:	führen von Sit. zu Nachfolgesit.

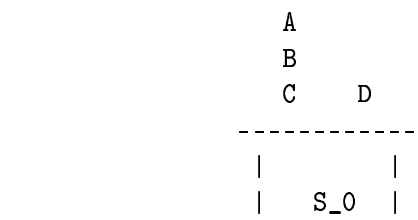
$holds(f, s)$	// Fluent f ist wahr in Sit. s
$do(a, s)$	// liefert Sit. nach Ausführen von a in s
S_0	// Anfangssit.

holds ist ein Prädikat und *do* eine Funktion.

Beispiel 10: (Blockwelt)

Σ: Wir definieren:
$holds(On(C, Table), S_0)$
$holds(On(B, C), S_0)$
$holds(On(A, B), S_0)$

$holds(On(D, Table), S_0)$
$holds(Clear(A), S_0)$
$holds(Clear(D), S_0)$
$holds(Clear(Table), S_0)$



$Move(x, y)$: stelle x auf y

Δ: Effekte von $Move$:

- $holds(On(x, y), do(Move(x, y), s)) \leftarrow holds(Clear(x), s) \wedge holds(Clear(y), s) \wedge x \neq y \wedge x \neq Table$
- $holds(Clear(z), do(Move(x, y), s)) \leftarrow holds(Clear(x), s) \wedge holds(Clear(y), s) \wedge holds(On(x, z), s) \wedge x \neq y$

$\Sigma \cup \Delta \models holds(On(A, D), do(Move(A, D), S_0))$

$\Sigma \cup \Delta \not\models holds(On(B, C), do(Move(A, D), S_0))$ // Persistenz

Frame Axiome:

$holds(On(v, w), do(Move(x, y), s)) \leftarrow holds(On(v, w), s) \wedge x \neq v$

- werden gebraucht, um Persistenz der Eigenschaften zu erhalten
- aber normalerweise ändern Axiome Eigenschaften nicht

Im Allgeimeinen $O(n - m)$ Frame Axiome nötig, wobei n Anzahl, m Fluents???

Frame Problem:

Wie kann man Effekte von Aktionen repräsentieren ohne alle Frame Axiome explizit hinschreiben zu müssen.

Idee: verwende Default-Regel, die besagt, dass sich Eigenschaften normalerweise **nicht** ändern.

$holds(f, do(a, s)) \leftarrow holds(f, s) \wedge \text{not } \neg holds(f, do(a, s))$
(**Persistenz-Default**)

Beispiel 11: (Tweety)

Fall 1:

$bird \leftarrow penguin$
$flies \leftarrow bird, \text{not } exception$
$exception \leftarrow penguin$
$bird$

$S = \{bird, flies\}$ ist grounded und abgeschlossen $\Rightarrow S$ ist Answer Set

Fall 2:

$bird \leftarrow penguin$
$flies \leftarrow bird, \text{not } exception$
$exception \leftarrow penguin$
$bird$
$penguin$

nun muss in S auch noch $penguin$ sein, d.h. $exception$ muss auch noch hinzukommen, damit wird jedoch die Zeile $flies \leftarrow bird, \text{not } exception$ gestrichen und $flies$ muss aus S verschwinden $\Rightarrow S$ ist für Fall 2 kein Answer Set

Beispiel 12: (manchmal sind mehrere Answer Sets möglich)

$a \leftarrow \text{not } b$
$b \leftarrow \text{not } a$

Answer Sets können sein $\{a\}, \{b\}$. $Cn(P^{\{b\}}) = \{b\}$

Beispiel 13: (kein AS)

$a \leftarrow \text{not } a$

Beweis 2:

jede Menge, die a nicht enthält, ist nicht closed.
jede Menge, die a enthält, ist nicht grounded.

Anmerkung: die leere Menge ist immer gegründet.

oder anders: Bsp. für kein AS, das b enthält:
$a \leftarrow \text{not } a, b$

4.3 Wie findet man Answer Sets?

Beobachtungen: immer Teilmenge der Regelköpfe, Obermengen der Fakten.
Echte Obermengen (Untermengen) von AS können nicht AS sein.

Beweisskizze:

Seien S, S' answer sets von P. Wir nehmen an $S \subset S'$.
Da S' echte Obermenge von S ist, gilt: $P^{S'} \subseteq P^S$ (mindestens so viele Regeln in P werden durch S' widerlegt wie durch S). Damit ist $Cn(P^{S'}) \subseteq Cn(P^S)$. Da S answer set ist, gilt $Cn(P^S) = S$ und damit $S' = Cn(P^{S'}) \subseteq Cn(P^S) = S \subset S'$. Also ist S' kein AS. Widerspruch!

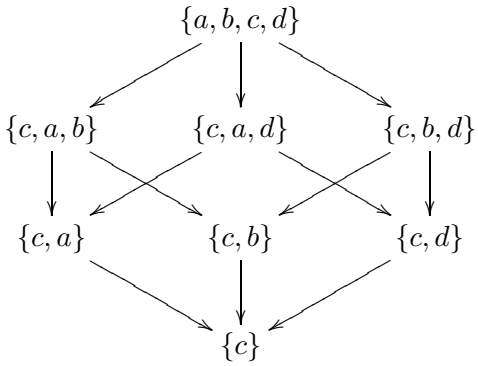
Beispiel 14:

1. $a \leftarrow \text{not } b$
2. $b \leftarrow \text{not } a, c$

3. $c \leftarrow \text{not } e$
4. $d \leftarrow \text{not } d, a$

Kandidaten für AS: Teilmengen von $\{a, b, c, d\}$.
e in keinem Kandidaten, also wegen (3) immer c.

Teilmengenbeziehungen:



Durch überprüfen finden wir AS $\{c, b\}$, d.h. wir können in unserem Graphen alle Teilmengen und Obermengen dieser Menge streichen. Durch Überprüfen der restlichen, stellen wir fest, daß $\{c, b\}$ einziges AS ist.

4.4 Ableitungsbäume

Knoten enhalten abgeleitete und "verbotene" Atome.

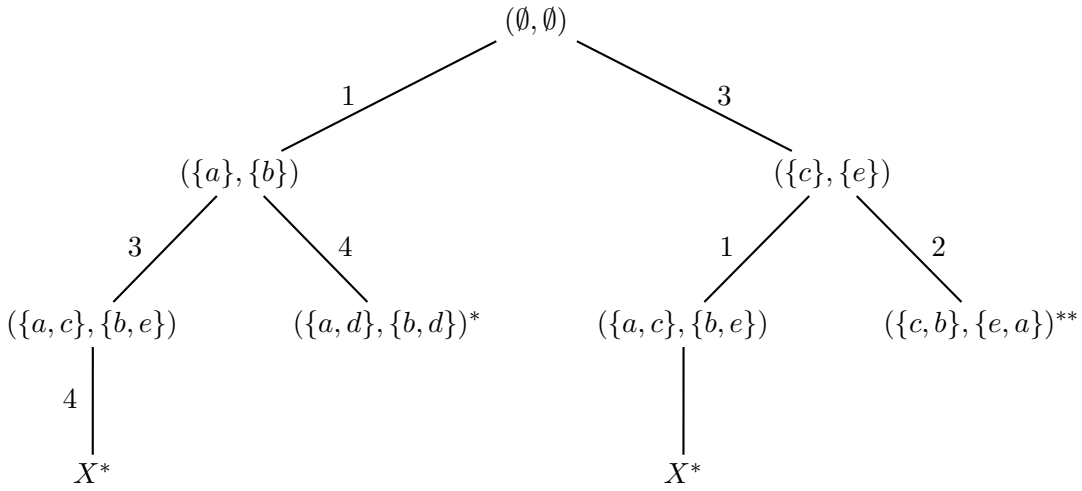
Wurzel: Fakten, keine verbotenen

Kanten: anwendbare Regeln (Vorbedingungen hergeleitet, nicht widerlegt)

Nachfolgeknoten: Regelkopf hinzu, default-negierte Atome in body zu den verbotenen

Falls abgeleitet $\cap$ verboten $\neq \emptyset$: Irrweg, kein AS
Falls keine weitere Regel anwendbar: AS gefunden

Angewendet auf das Beispiel 14 (Seite 15):



* gestrichen
** gefunden

Beispiel 15:

Regeln:

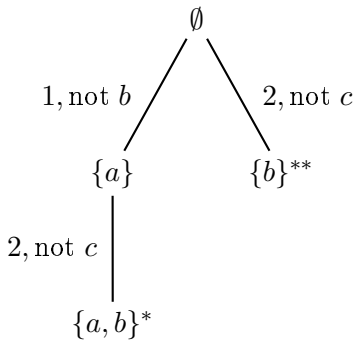
1. $a \leftarrow \text{not } b$
2. $b \leftarrow \text{not } c$

Anwendung von:

1. $\rightsquigarrow a$, wobei ich mir merken muss, daß a nur abgeleitet werden kann, wenn b nicht abgeleitet wird
2. $\rightsquigarrow a, b$, wobei dies gestrichen werden muss

---

2. $\rightsquigarrow b$, funktioniert!

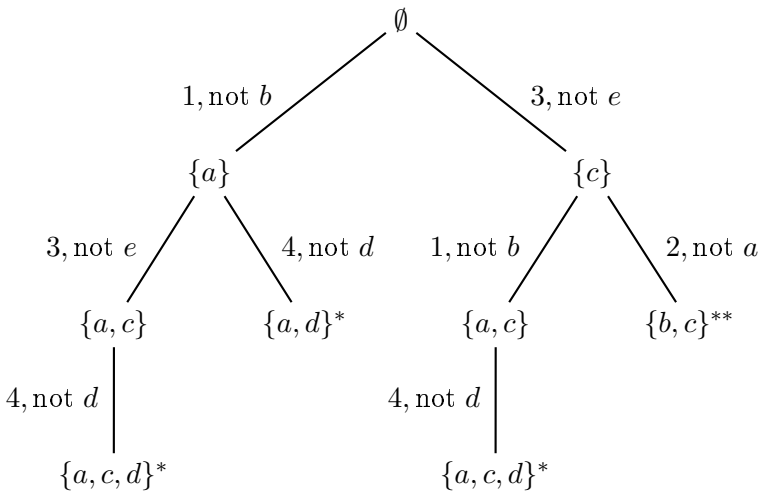


* geht nicht, da not b vorausgesetzt
** funktioniert

Knoten enthalten abgeleitete Atome (Wurzel: Fakten). Kanten entsprechen anwenbaren Regeln. Nachfolgeknoten bekommt Kopf der Regel dazu not -Atome aus Körper werden an

Kante markiert. Falls Knoten Atom enthält, das negiert auf Pfad zu ihm liegt → Widerspruch!

Nochmalige Betrachtung von Beispiel 14 von Seite 15:



* gestrichen
** gefunden, mit Kasten markiert

Probe: $P^{\{b,c\}} : \{c, b\}$ mit Zwischenrechnung:

1. ~~$a \leftarrow$ not b~~
2. $b \leftarrow$ ~~not a~~, c
3. $c \leftarrow$ ~~not c~~
4. $d \leftarrow$ ~~not d~~, a

Bemerkungen zu den Negationen:

- not x ist wahr, wenn x **nicht** hergeleitet werden kann
- $\neg x$ ist wahr, wenn $\neg x$ hergeleitet werden kann

4.5 Meeting scheduling

Problembeschreibung:

$meeting(m_1), \ldots, meeting(m_n)$ // zu organisierende Meetings
$time(t_1), \ldots, time(t_s)$ // verfügbare Zeiten
$room(r_1), \ldots, room(r_m)$ // verfügbare Räume
$person(p_1), \ldots, person(p_k)$ // verfügbare Personen
$participant(p_1, m_1), participant(p_2, m_3), \ldots$ // Teilnehmer an Meeting

---

$at(M, T) \leftarrow meeting(M), time(T), \text{not } \neg at(M, T)$
$\neg at(M, T) \leftarrow meeting(M), time(T), \text{not } at(M, T)$

Es kann immer nur eine Regel der beiden angewendet werden. Wenn keine Informationen über Meeting M zur Zeit T vorliegen, kann ich mir aussuchen, ob es zu dieser Zeit stattfinden soll oder nicht.

$in(M,R) \leftarrow meeting(M), room(R), \text{not } \neg in(M,R)$
$\neg in(M,R) \leftarrow meeting(M), room(R), \text{not } in(M,R)$

damit ist generate abgeschlossen

$timeassigned(M) \leftarrow at(M,T)$
$roomassigned(M) \leftarrow in(M,R)$
$\leftarrow meeting(M), \text{not } timeassigned(M)$ // kein Meeting ohne Zeit
$\leftarrow meeting(M), \text{not } roomassigned(M)$ // kein Meeting ohne Raum
$\leftarrow meeting(M), at(M,T), at(M,T'), T \neq T'$ // kein Meeting zu versch. Zeiten
$\leftarrow meeting(M), in(M,R), in(M,R'), R \neq R'$ // kein Meeting in versch. Räumen
$\leftarrow in(M,R), in(M',R), at(M,T), at(M',T), M \neq M'$ // gleichzeitige Meetings in versch. Räumen
$\leftarrow participant(P,M), participant(P,M'), M \neq M', at(M,T), at(M',T)$ // versch. Meetings mit gleichen Teilnehmern nicht möglich

ASP[2] für Erfüllbarkeit aussagenlogischer Klauselmengen.

generate:

für jedes Atom A:
$A \leftarrow \text{not } \neg A$
$\neg A \leftarrow \text{not } A$

AS: $\{A\}, \{\neg A\}$

erzeugt alle Interpretationen: in jedem AS taucht jedes Atom positiv oder negiert auf, entspricht W bzw. F in Interpretation.

test:

für jede Klausel $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_m$:
$\leftarrow \text{not } A_1, \ldots, \text{not } A_n, B_1, \ldots, B_m$

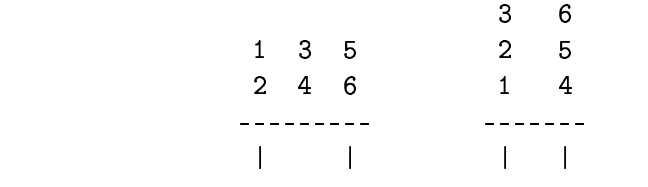
ASs entsprechen genau den Modellen der Klauselmenge.

Beispiel 16:

hier sollte ein kleines Beipiel hin. Egal, zu schnell weg.

4.6 Blockwelt

Anfangszustand **Zielzustand**



[2] Answer Set Programming

gegeben:

Tisch, Blöcke (Integer), Zeitpunkte (Integer)

Aktion: $move(B, L, I)$ bewegt Block B zum Zeitpunkt I auf L.

$time(0), time(1), \ldots, time(k)$ // k - max. Planlänge
$location(B) \leftarrow block(B)$ // jeder Block ist eine Location
$location(table)$ // der Tisch ist eine Location
$block(1), \ldots, block(6)$ // es gibt die Blöcke 1 bis 6

Effekt von $move$: $on(B, L, T+1) \leftarrow move(B, L, T)$
Inertia (Trägheit): $on(B, L, T+1) \leftarrow on(B, L, T), \text{not } \neg on(B, L, T+1), T < k$
Eindeutige Location: $\neg on(B, L_1, T) \leftarrow on(B, L, T), L \neq L_1$

---

generate: $move(B, L, T) \leftarrow \text{not } \neg move(B, L, T), block(B), location(L), time(T), T < k$
$\neg move(B_1, L_1, T) \leftarrow move(B, L, T), block(B_1), B \neq B_1, T < k, location(L_1)$
// in jedem AS zu jedem Zeitpunkt genau eine Aktion (ein $move$)

test: nur 1 Block direkt auf einem anderen:
$\leftarrow on(B_1, B, T), on(B_2, B, T), B_1 \neq B_2$
nur freie Blöcke bewegen:
$\leftarrow move(B, L, T), on(B_1, B, T)$

Nun noch Anfangszustand und Zielzustand beschreiben.

Anfangszustand	**Zielzustand**
$on(1, 2, 0)$	$\leftarrow \text{not } on(3, 2, k)$
$on(2, table, 0)$	$\leftarrow \text{not } on(2, 1, k)$
$on(3, 4, 0)$	$\leftarrow \text{not } on(1, table, k)$
$on(4, table, 0)$	$\leftarrow \text{not } on(6, 5, k)$
$on(5, 6, 0)$	$\leftarrow \text{not } on(5, 4, k)$
$on(6, table, 0)$	$\leftarrow \text{not } on(4, table, k)$

Teil eines AS für $k = 5$:

$move(1, table, 0)$
$move(2, 1, 1)$
$move(3, 2, 2)$
$move(5, 4, 3)$
$move(6, 5, 4)$

Die Fakten sind natürlich auch im AS enthalten, deswegen *Teil eines AS*.

zurück zu den Folien...

Default-Logik ist Verallgemeinerung der Logik-Programmierung, d.h. sie ist aussagekräftiger. Z.B. können keine Regeln der Form "Block A steht auf Block B oder C" in der Logik-Programmierung formuliert werden.

4.7 (3.2) Reiters Default-Logik DL

kurz durchgesprochen

4.8 Default-Theorien

Wir beginnen mit dem sicheren Wissen W, leiten dann neues Wiessen ab, indem wir Default-Regeln anwenden.

Man stelle sich folgendes als Mengen vor:
$((((W)a)...)\neg b)$ alls zusammen ergibt E

$a : b/c$ anwendbar ?

also: E raten; Anwendbarkeit in defaults bzgl. E prüfen

cons auf Folie 80 steht für Konsequenz
Folie 84: Fixpunkte meinen EXTENSIONEN

4.9 Rätsel: Knights and Knaves

zu deutsch: Ritter und Schurken

Knights (Ritter): sagen immer die Wahrheit
Knaves (Schurken): lügen immer

Besucher trifft Einwohner a, b, c

I) a sagt b und c sind Knights.

II) b sagt a ist Knave und c ist Knight.

Frage: Was sind a, b, c?

Formulierung eines Logik-Programmes:

$person(a)$
$person(b)$
$person(c)$
$knight(X) \leftarrow person(X), \text{not } knave(X)$
$knave(X) \leftarrow pseron(X), \text{not } knight(X)$

---

1) $knight(b) \leftarrow knight(a)$ // aus I, falls a Knight
2) $knight(c) \leftarrow knight(a)$ // aus I, falls a Knight
3) $\leftarrow knave(a), knight(b), knight(c)$ // aus I, falls a Knave
4) $knave(a) \leftarrow knight(b)$ // aus II, falls b Knight
5) $knight(c) \leftarrow knight(b)$ // aus II, falls b Knight
6) $\leftarrow knave(a), knave(b), knight(c)$ // aus II, falls b Knave

Kürze nun in der Tabelle Knight mit 1 und Knave mit 0 ab.

a	b	c	eliminiert durch
0	0	0	
0	0	1	6)
0	1	0	5)
0	1	1	3)
1	0	0	1)
1	0	1	1)
1	1	0	2)
1	1	1	4)

einziges answer set enthält: $knave(a), knave(b), knave(c)$

4.10 Default-Logik

beliebige Formeln als Komponenten in D und W.

LP[3]-Regel $p \leftarrow q, \text{not } r$ entspricht $\frac{q : \neg r}{p}$

LP also Spezialfall von Default-Logik Defaults der Form

- $p : q/q$ heißen normale Defaults.
- $true : q/q$ heißen supernormale Defaults.

Für supernormale Default-Theorien (D, W) sind Extensionen genau die MEngen $Th(W \cup C)$, wobei C maximale mit W konsistente Teilmenge der Konklusionen von Defaults in D ist.

4.11 Modellbasierte Diagnose

Diagnose: Identifikation eines vorliegenden Fehlers (Krankheit) auf der Basis von Beobachtungen und Hintergrundwissen über das System.

2 Arten: abduktiv vs. konsistenzbasiert

Typ I: abduktiv:

gegeben: Informationen über: 1) Wirkung von Fehlern H (Implikationen)
2) beobachtete Symptome S (Atome)
3) nicht aufgetretene Symbole N (negierte Literale)

gesucht: Erklärung für S, d.h. minimale Menge von Fehlern F, so dass $F \cup H \cup N$ konsistent und $F \cup H \models S$.

Deduktion:

Ableitung von implizitem Wissen

Alle A sind B
C ist ein A
C ist ein B

[3] LP = Logik-Programm

Abduktion:

Finden von Erklärungen für Beobachtungen

Alle A sind B
C ist ein B
C ist ein A

Induktion:

Ableiten einer allgemeinen Regel aus Beobachtungen

C ist ein A
C ist ein B
Alle A sind B

(alle bisher beobachteten A sind B)

Beispiel 17:

H: $Masern \rightarrow Fieber \wedge Ausschlag$
$Migraene \rightarrow Kopfschmerzen \wedge Uebelkeit$
$Grippe \rightarrow Kopfschmerzen \wedge Gliederschmerzen \wedge Fieber$

N: leere

S: $Fieber \Rightarrow Masern; Grippe$
$Fieber, Ausschlag \Rightarrow Masern;$ ~~{Grippe, Masern}~~
$Kopfschmerzen, Fieber \Rightarrow Grippe; \{Migraene, Masern\}$

1. fällt weg, da Minimalität gefordert war.

Beispiel 18:

H: $Masern \rightarrow Fieber \wedge Ausschlag$
$Migraene \rightarrow Kopfschmerzen \wedge Uebelkeit$
$Grippe \rightarrow Kopfschmerzen \wedge Gliederschmerzen \wedge Fieber$

N: $\neg Uebelkeit$

S: $Fieber \Rightarrow Masern; Grippe$
$Fieber, Ausschlag \Rightarrow Masern;$ ~~{Grippe, Masern}~~
$Kopfschmerzen, Fieber \Rightarrow Grippe; \{$~~Migraene~~$, Masern\}$

2. fällt weg, da bei Migeräne Übelkeit auftritt.

Typ II: konsistenzbasiert:

Beschreibt, was passiert, wenn die Komponente korrekt ist.

$\rightsquigarrow$ *siehe Folie 88*

5 Wissensbasiertes Planen

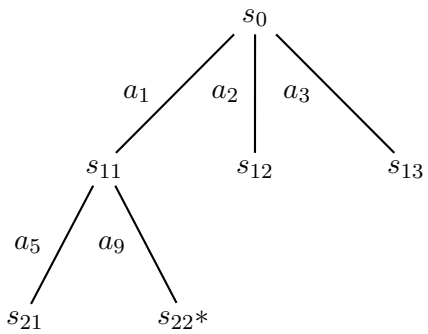
5.1 STRIPS

5.1.1 Naive Vorwärts- und Rückwärtsplaner

Wie findet man die richtige Aktionsfolge für Ziel g?

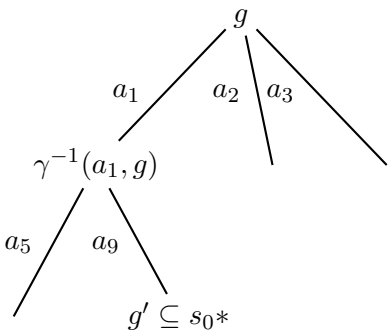
Aktionen generieren Situationsraum

Progressionsplaner:



* Zielknoten

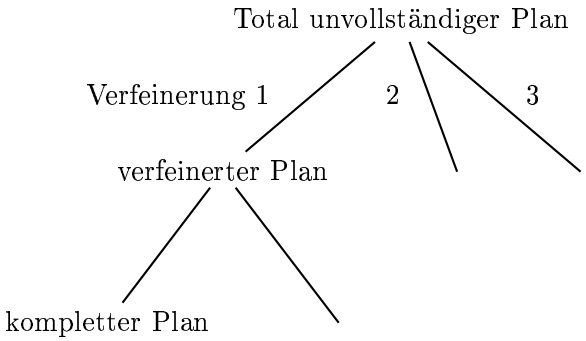
Regressionsplaner:



* fertig

Rückwärtsplaner meist effektiver.

5.1.2 Suche im Raum partieller Pläne



Bemerkung zu Folien 136:

Unvollständig, weil nicht alle Vorbedingungen von Finish erfüllt
→ Einfügen von weiteren Aktionen
(Binden von Variablen bzw. Fenstlegen ...)

5.2 Plangraph

Zeitpunkte $t_1, t_2, \ldots$ sind geordnet, aber zu einen Zeitpunkt können mehrere Aktionen stattfinden.

t_1	$<$	t_2	$<$	t_3
a_1		a_4		a_9
a_2				a_{237}

$\Rightarrow$ totale Ordnung auf Mengen von Aktionen.

Bemerkung zu Folie 21:

q ist Vorbedingung von a_2, a_2 hat s zur Folge, r wird durch a_2 gelöscht

Konflikt zwischen a_1 und a_2 aufgrund von $r \Rightarrow$ können nicht in einem Schritt ausgeführt werden.

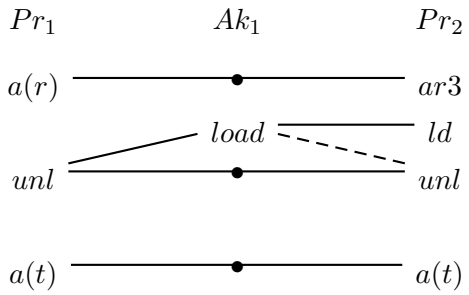
Beispiel 19: (Plangraph)

shoot(x)
pre: *loaded*
add: *dead*(x), *unloaded*
del: *alive*(x), *loaded*

load
pre: *unloaded*
add: *loaded*
del: *unloaded*

init: *alive*(r), *alive*(t), *unloaded*
goal: *dead*(r), *dead*(t)

Graph:



Mutex: $load, noop(unl) \mid ld, unl$

Mutex-Relationen:

Ak_1	Pr_2	Ak_2	Pr_3	Ak_3	Pr_4
$load, noop(unl)$	ld, unl	$load, noop(unl)$	$d(r), d(t)$	$noop(d(r)), noop(d(t))$	$d(r), d(t)$
		$sh(r), sh(t)$	$d(r), ld$	$sh(r), sh(t)$	
		$sh(r), load$	$d(r), ld$	$sh(r), load$	
		$sh(t), load$		$sh(t), load$	
				$noop(d(r)), sh(t)$	
				$noop(d(t)), sh(r)$	

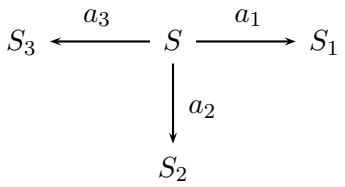
Plan in Level 5 gefunden:

t_1	t_2	t_3	t_4
$load$	$sh(t)$	$load$	$sh(r)$
		$noop(d(t))$	$noop(d(t))$

5.3 Planen mit heuristischer Suche: der FF-Planer

Bemerkung zu Folie 27:

Lokale Suche, die bei einem Startzustand S startet und von diesem aus mit Aktionen a_i zu Nachbarzuständen S_j gelangt. Falls kein besseres Ergebnis gefunden → von Nachbarzuständen ausgehend weitersuchen.



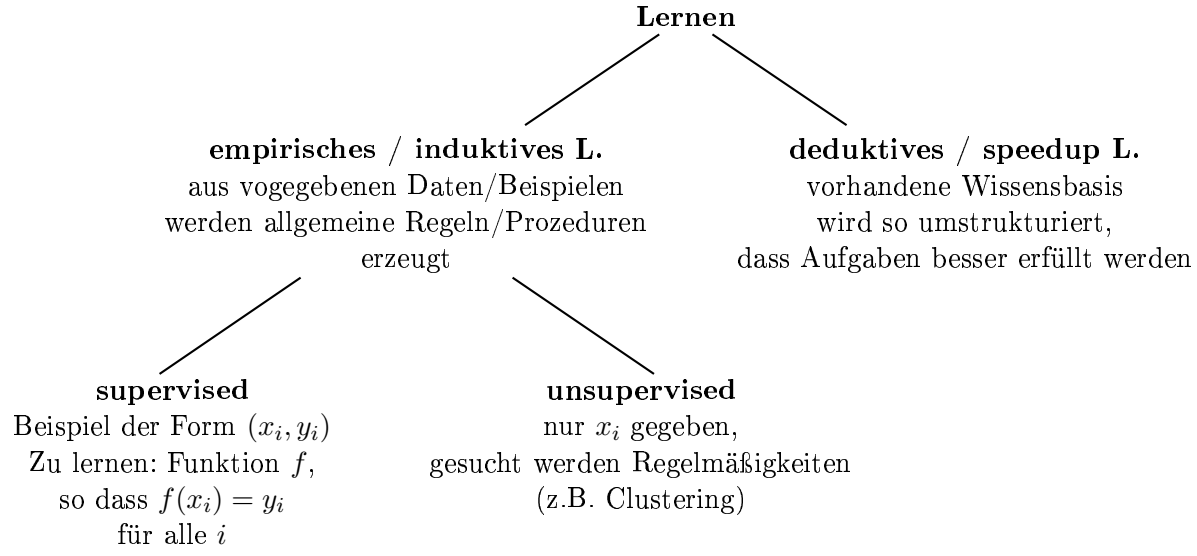
6 Maschinelles Lernen

6.1 Was ist Lernen?

Simon: Jede Veränderung eines Systems, die es ihm erlaubt, eine Aufgabe bei der Wiederholung derselben Aufgabe besser zu lösen.

Michalski: Lernen ist das Konstruieren oder Verändern von Repräsentationen von Erfahrungen.

6.2 Typen des Lernens



6.3 Induktive Logikprogramme (ILP)

gegeben: Menge positiver Beispiele P für $g(X_1, \ldots, X_n)$
Menge negativer Beispiele N für $g(X_{n+1}, \ldots, X_m)$
beide Mengen bestehen aus Atomen mit g als Prädikat
Hintergrundwissen B (Regeln eines LP)

gesucht: Menge von Regeln H, die Zielprädikat definieren:

H $g(X_1, \ldots, X_n) \leftarrow body_1$
$\ldots$
$g(X_1, \ldots, X_n) \leftarrow body_n$

Was soll in der Klausur dran kommen?

- bei Suche (heuristische/greedy-Verfahren) könnte sein
- Hornklauseln angucken

- keine Beschreibunglogik
- Logikprogrammierung, Answer set
- Diagnose
- Planen!

Algorithmus 2:

find_a_rule liefert Regel, die einige pos. abgedec...

Beispiel 20:

Hintergrundwissen B:

$parent(ann, mary), parent(ann, tan), parent(tan, eve),$
$parent(eve, ian), female(ann), female(mary), female(eve)$

pos Beispiele P: $daughter(mary, ann), daughter(eve, tan)$
neg Beispiele N: $daughter(tan, ann), daughter(eve, ann)$

Annahmen: heuristischer Wert eines Literals: Anzahl pos. abgedeckter minus Anzahl neg. abgedeckter Literale.

zulässige Literale nur solche mit Variablen als Argument.

abgedeckt wird ein Bsp., wenn die Regel mir erlaubt, daß Bsp. richtig zu identifizieren.

Aufruf von find_a_rule(P, N) konstruiert folgende bodies:

$body_1$: initialisiert mit true $daughter(X_1, X_2) \leftarrow true$
beide neg. Beispiele abgeleitet

$body_2$:	Alternativen:			
	$female(X_1)$,	$female(X_2)$,	$parent(X_1, X_2)$,	$parent(X_2, X_1)$
heurist. Wert:	2-1	1-2	0-0	2-1

$daughter(X_1, X_2) \leftarrow female(X_1)$

$body_3$: Auswertung ergibt maximalen herist. Wert für $parent(X_2, X_1)$
$\rightsquigarrow H = \{daughter(X_1, X_2) \leftarrow female(X_1), parent(X_2, X_1)\}$

6.4 Instance Base Learning

Bisher: Erzeugen von Regeln aus Beispielen

kann man Wert nicht gleich aus Beispielen ableiten?

Ja, wenn es geeignetes Ähnlichkeitsmaß zw. Objekten gibt.

Idee: $F(x_i) = y_j$, falls das Paar (x_j, y_j) in den Beispielen enthalten ist, und x_j das Objekt in den Beispielen ist, daß x_i am ähnlichsten ist.

Zur Erhöhung der Zuverlässigkeit: betrachte die k ähnlichsten Objekte.

Beispiel 21:

$k = 4$, die x ähnlichsten 4 Beiepiele sind die folgenden:
(x_1, a) $sim(x, x_1) = 1$
(x_2, a) $sim(x, x_2) = 1$
(x_3, a) $sim(x, x_3) = 1$
(x_4, b) $sim(x, x_4) = 2$

sim gibt Ähnlichkleit an

für x prognostizierter Wert: der y-Wert, der folgenden Ausdruck maximiert:

$$\sum_{(x_i,y_i)\in P} sim(x, x_i)$$

P ist die Menge der k ähnlichsten Beispiele

$$\sum_{(x_i,a)\in P} sim(x, x_i) = 3$$

$$\sum_{(x_i,b)\in P} sim(x, x_i) = 2$$

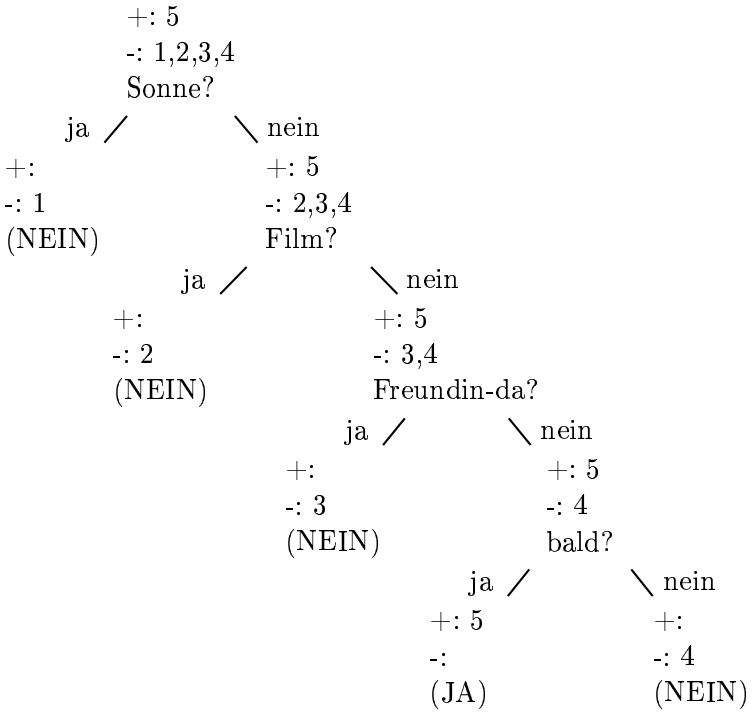
das wars...

7 Klausur

7.1 Lernen von Entscheidungsbäumen

Soll ich für Klausur lernen?

	Sonne?	Film	Freundin-da?	bald?	Entscheidung
1	ja	nein	nein	ja	nein
2	nein	ja	nein	ja	nein
3	nein	nein	ja	ja	nein
4	nein	nein	nein	nein	nein
5	nein	nein	nein	ja	ja



7.2 Planungsproblem

Gegeben sei ein Planungsproblem in STRIPS-Notation durch den folgenden Startzustand S, den Zielzustand Z, sowie die angegebenen Operationen:

$$S = \{ontable(A), ontable(B), freeC), free(B), empty\}$$
$$Z = \{ontable(C), on(B, C), on(A, B)\}$$

dann wurden die Operationen mit Vorbedingungen, add- und delete-List in einer Tabelle angegeben.

7.3 Modellbasierte Diagnose

gegeben:

$A = 2, B = 3, C = 4, D = 1$

A_1, A_2 seien Addierer
M_1, M_2 seien Multiplizierer

A und B seien Eingänge von A_1
C und D seien Eingänge von A_2
Ausgänge von A_1 und A_2 seien Eingänge von M_1
Ausgänge von A_1 und A_2 seien Eingänge von M_2

Ausgang von $M_1 = 20$
Ausgang von $M_2 = 25$

Diagnosen:

Falls A_1, A_2 (und M_2) korrekt, dann ist M_1 kaputt

Falls M_1 korrekt, dann sind (A_2 und M_2) oder (A_1 und M_2) kaputt

7.4 Logikprogrammierung

$a \leftarrow \text{not } b$
$b \leftarrow \text{not } a$
$d \leftarrow \text{not } d, a$

AS: $\{b\}$

Tests:

$P^{\{b\}}$:
b
$d \leftarrow a$

$\Rightarrow Cn(P^{\{b\}}) = \{b\}$

$P^{\{a,d\}}$:
a

$\Rightarrow Cn(P^{\{a,d\}}) = \{a\}$

8 Vokabeln

englisch	deutsch
defeats	wiederlegen
derivation	Ableitung