Nichtmonotones Schließen: Motivation

1. Beschreibung typischer Zusammenhänge

Klassische Logik ist monoton: mehr Prämissen haben nie weniger Folgerungen
Sei $Th(T) = \{p \mid T \models p\}$

$T_1 \subseteq T_2$ impliziert $Th(T_1) \subseteq Th(T_2)$

nicht immer gewollt:
Standarbeispiel: kann Vogel Tweety fliegen?
Vögel können fliegen.
Tweety ist ein Vogel.

aber es gibt Ausnahmen: Pinguine, Strauße, Vögel mit gestutzten Flügeln ...
wir wollen ableiten, dass Tweety fliegt, wenn wir nicht wissen, dass er Ausnahme ist.
Falls Zusatzinformation besagt, dass es sich um einen Pinguin oder Strauß oder .... handelt:
Schluss zurücknehmen

$\forall x\ Vogel(x) \supset Fliegt(x)$
$\forall x\ \ Pinguin(x) \supset \neg Fliegt(x)$
$\forall x\ Pinguin(x) \supset Vogel(x)$

ableitbar: es gibt keinen Pinguin (Hinzufügen von Pinguin(Tweety) gibt Inkonsistenz)
Ersetzen wir obige Repräsentation durch:

$\forall x\ Vogel(x) \wedge \neg Ausnahme(x) \supset Fliegt(x)$
$\forall x\ Pinguin(x) \supset Ausnahme(x)$
$\forall x\ Strauß(x) \supset Ausnahme(x)$
...
Vogel(Tweety)

jetzt ist Fliegt(Tweety) nicht ableitbar, da $\neg$Ausnahme(Tweety) nicht gefolgert werden kann.
Wir müssten exakt definieren, was die möglichen Ausnahmen sind, und wissen, dass Tweety zu keiner dieser Klassen gehört.

(relativ) einfache Lösung des Problems in der Logikprogrammierung:
neue Art der Negation not (default negation):
not Ausnahme(Tweety) wahr, wenn Ausnahme(Tweety) nicht hergeleitet werden kann

Notation:

Fliegt(X) $\leftarrow$ Vogel(X), not Ausnahme(X)
Ausnahme(X) $\leftarrow$ Pinguin(X)
Ausnahme(X) $\leftarrow$ Strauß(X)
...
Vogel(Tweety)

Fliegt(Tweety) herleitbar, da Vogel(Tweety) gegeben und Ausnahme(Tweety) nicht herleitbar

2. Situationskalkül.

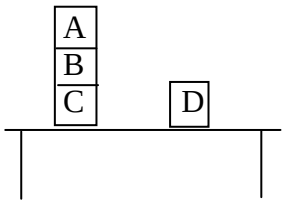
McCarthy, Hayes, 1963, 1969

Situationen: Schnappschüsse von Weltzuständen, entsprechen Folgen von Aktionen
Fluents: zeitabhängige Eigenschaften, können verschiedene Werte annehmen (etwa w,f)
Aktionen: führen von Situation zu Nachfolgesituation

holds(f,s) f ist wahr in s
do(a,s) Situation nach Ausführen von a in s
poss(a,s) Aktion a ist ausführbar in s

Blockwelt:

Σ :=
1) holds(On(C,Table),S_0)
2) holds(On(B,C),S_0)
3) holds(On(A,B), S_0)
4) holds(On(D,Table), S_0)
5) holds(Clear(A), S_0)
6) holds(Clear(D), S_0)
7) holds(Clear(Table), S_0)



Move(x,y): bewege x auf y. Formalisierung (freie Variable allquantifiziert):
Beschreibung, wann Move möglich, sowie der Effekte von Move

8) poss(Move(x,y),s) <-> holds(Clear(x),s) $\wedge$ holds(Clear(y),s) $\wedge$ x$\neq$y $\wedge$ x$\neq$Table

9) poss(Move(x,y),s) $\supset$ holds(On(x,y),do(Move(x,y),s))

10) poss(Move(x,y),s) $\wedge$ holds(On(x,z),s) $\supset$ holds(Clear(z),do(Move(x,y),s))

11) holds(On(y,z),s) $\wedge$ z$\neq$Table $\supset$ $\neg$holds(Clear(z),s)

12) holds(On(x,z),s) $\wedge$ z$\neq$y $\supset$ $\neg$holds(On(x,y),s)

1, ...,12 |= holds(On(A,D),do(Move(A,D), S_0)) unique names assumption!!
1, ...,12 |= holds(Clear(B),do(Move(A,D), S_0))

aber nicht:

1, ...,12 |= holds(On(B,C),do(Move(A,D), S_0)) Persistenz von Eigenschaften

Hinzufügen von Frame-Axiomen:

holds(On(v,w),s) $\wedge$ x$\neq$v $\supset$ holds(On(v,w),do(Move(x,y),s))
(holds(Clear(x),s) $\wedge$ x$\neq$z) $\vee$ x = Table $\supset$ holds(Clear(x), do(Move(y,z),s))

2 neue Axiome. Na und?

Color(x,c) x hat Farbe c
Paint(x,c) male x mit Farbe c an

holds(Color(x,c), do(Paint(x,c),s) keine Vorbedingungen

Annahme: alle Blöcke und Tisch zunächst rot: holds(Color(x,Red), S_0)

Neue Frame Axiome nötig: Move hat keinen Effekt auf Farbe, Paint färbt nur 1 Objekt

$$holds(Color(x,c),s) \supset holds(Color(x,c),do(Move(y,z),s))$$
$$holds(Color(x,c1),s) \wedge x \neq y \supset holds(Color(x,c1),do(Paint(y,c2),s))$$

Neue Frame Axiome auch für andere Fluents:

$$holds(On(v,w),s) \supset holds(On(v,w),do(Paint(z,c),s))$$
$$holds(Clear(x),s) \supset holds(Clear(x), do(Paint(y,c),s))$$

n Fluents und m Aktionen: $O(n \cdot m)$

Frame Problem: wir kann man Effekte von Aktionen repräsentieren, ohne sämtliche Frame Axiome explizit hinschreiben zu müssen?

Ist FP nur ein Problem des Situationskalküls? bisherige Erfahrung: tritt in verschiedener Form bei jeder Formalisierung von Handlungen wieder auf.

Lösungsidee: verwende eine Default-Regel, die besagt, dass Eigenschaften sich normalerweise nicht ändern:

$$holds(f, do(a,s)) \leftarrow holds(f,s), not \neg holds(f, do(a,s))$$

Answer sets (Ergänzungen zu den Folien)

1. Definites Programm P:

1. r <- s
2. q <- p
3. p

Akzeptable Überzeugungsmenge?

{p}	zu klein, 2 nicht angewendet
{p,q,r}	zu groß, r hat keine Herleitung
{p,q}	ok

answer set = kleinste unter P abgeschlossene Atommenge = kleinstes Modell von P

Minimal abgeschlossen => es gibt Herleitung

2. Normales Programm P:

1. p <- r, not q
2. q <- r, not p
3. r <- not s

Akzeptable Überzeugungsmengen?

{r}	zu klein, 1, 2 nicht angewendet
{p,q,r}	zu groß, p, q haben keine Herleitung
{s}	s hat keine Herleitung (minimal abgeschlossen!)
{p,r}	ok
{q,r}	ok

Minimal abgeschlossen $\not\Rightarrow$ es gibt Herleitung
answer set = unter P abgeschlossen und es gibt gültige Herleitung

Wann hat Element von S gültige Herleitung? Herleitung basiert auf Regeln, die nicht von S widerlegt werden: es gibt kein not b im Körper, so dass b in S.

S answer set? Ein einfacher Test:

Eliminiere aus P alle Regeln, die not L im Körper haben, wobei L in S.
Eliminiere aus allen verbleibenden Regeln die not-Literale.
Teste, ob S kleinstes Modell des (definiten) reduzierten Programms.

{s}	{p,r}	{q,r}
p <- r, ~~not q~~	p <- r, ~~not q~~	~~p <- r, not q~~
q <- r, ~~not p~~	~~q <- r, not p~~	q <- r, ~~not p~~
~~r <- not s~~	r <- ~~not s~~	r <- ~~not s~~

Tweety und answer sets

bird <- penguin
flies <- bird, not exception
exception <- penguin
bird AS: {bird, flies}

dazu: penguin AS: {penguin, bird, exception}

mehrere answer sets:

a <- not b
b <- not a

kein answer set:

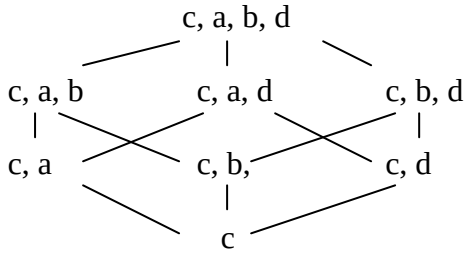
a <- not a

Beweisskizze:
jede Menge, die a nicht enthält, ist nicht closed
jede Menge, die a enthält, ist nicht grounded (keine Herleitung für a)

Wie findet man answer sets?
Beobachtungen: immer Teilmengen der Regelköpfe, Obermengen der Fakten
Ober-(Unter-) mengen von AS können nicht AS sein.
Beweisskizze:
Sei P ein Logikprogramm, 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 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 $Cn(P^{S'}) \subset S'$. S' ist also kein answer set, im Widerspruch zur Annahme.

1) a <- not b
2) b <- not a, c
3) c <- not e
4) d <- not d, a

nur Teilmengen von {a,b,c,d} kommen in Frage
da e nicht drin, muss c drin sein: c + Teilmenge von {a, b, d}
{c,b} answer set: alle Obermengen und Teilmengen fallen weg



es bleiben c,a; c,a,d; c,d

keiner davon answer set

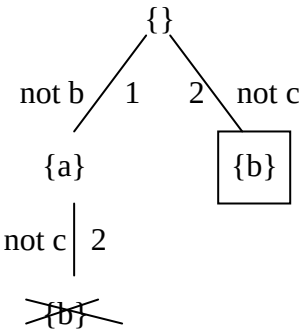
Ableitungsbäume:

Motivation: betrachte folgendes Programm

1) a <- not b
2) b <- not c

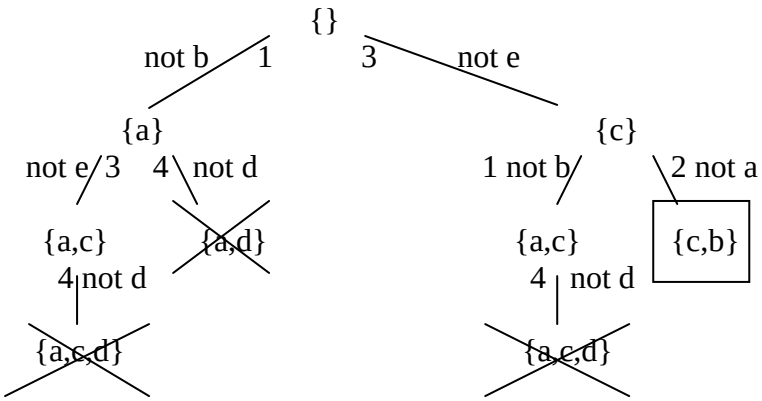
Wir probieren der Reihe nach Regeln anzuwenden, deren nichtnegierte Vorbedingungen bereits hergeleitet sind und deren negierte noch nicht widerlegt sind:

1) liefert a, 2) liefert b, aber ist {a,b} answer set? Nein, da a nur unter der Annahme abgeleitet werden kann, dass b nicht abgeleitet wird.
Also: merken, was nicht abgeleitet werden darf:



Knoten enthalten bereits abgeleitete Atome; Wurzel: Fakten
Kanten entsprechen anwendbaren Regeln: Vorbedingungen hergeleitet, nicht widerlegt
Nachfolgerknoten bekommt Kopf dazu, default-negierte in body an Kanten markiert
Falls Knoten Atom enthält, das negiert an Kante zu ihm liegt: Irrweg, kein answer set
Falls keine weitere Regel mehr anwendbar: AS gefunden

Beispiel:



Meeting scheduling

problem description:

meeting(m_1), …, meeting(m_n)	% zu organisierende Meetings
time(t_1),…, time(t_s)	% verfügbare Zeitpunkte
room(r_1),..., room(r_m)	% verfügbare Räume
person(p_1),..., person(p_k)	% Personen
participant(p_1,m_1)	% Teilnehmer an Meeting
participant(p_2,m_3)	
…	

problem independent part, generate:

at(M,T) <- meeting(M), time(T), not ¬at(M,T)
¬at(M,T) <- meeting(M), time(T), not at(M,T)
in(M,R) <- meeting(M), room(R), not ¬in(M,R)
¬in(M,R) <- meeting(M), room(R), not in(M,R)
% generiert beliebige Zuweisungen von Zeitslots und Räumen

test:
timeassigned(M) <- at(M, T)
roomassigned(M) <- in (M,R)

<- meeting(M), not timeassigned(M)	% kein Meeting ohne Zeit
<- meeting(M), not roomassigned(M)	% kein Meeting ohne Raum
<- meeting(M), at(M,T), at(M,T'), T ≠ T'	% kein Meeting mit 2 Zeiten
<- meeting(M), at(M,T), at(M,T'), T ≠ T'	% kein Meeting mit 2 Räumen

<- in(M,X), in(M',X), at(M,T), at(M',T), M ≠ M'.
% gleichzeitige meetings haben versch. Räume
<- participant(P,M), participant(P, M'), M ≠ M',at(M,T), at(M',T).
% meetings mit demselben Teilnehmer haben versch. Zeiten