Answer Sets: Beispiel Meeting Scheduling

I. Beschreibung der Probleminstanz:

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
partic(p_1,m_1), …, partic(p_i,m_j),…	% Teilnehmer an Meeting

II. Instanz-unabhängiger Teil:

a) Generate: generiert beliebige Zuweisungen von Zeiten und Räumen an Meetings

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)

b) Test: Ausschluss von Answer Sets, die keine Lösungen sind

timeassigned(M) ← at(M,T)
roomassigned(M) ← in(M,R)

← meeting(M), not timeassigned(M) % *kein M. ohne Zeit*
← meeting(M), not roomassigned(M) % *kein M. ohne Raum*
← meeting(M), time(T), time(T'), at(M,T), at(M,T'), T ≠ T' % *nur 1 Zeit pro M.*
← meeting(M), room(R), room(R'), in(M,R), in(M,R'), R ≠ R' % *nur 1 Raum pro M.*

← in(M,X), in(M',X), at(M,T), at(M',T), M ≠ M'
% *verschiedene Räume für gleichzeitige Meetings*

← partic(P,M), partic(P,M'), at(M,T), at(M',T), M ≠ M'
% *Meetings mit selbem Teilnehmer haben versch. Zeiten*

Answer Sets des Programms beinhalten Lösung des Scheduling Problems!!

Berechnung von Answer Sets (normale Programme)

- Wie findet man answer sets?

Beobachtungen: 1) AS immer Teilmengen der Regelköpfe, Obermengen der Fakten
2) Ober-(Unter-) mengen von AS können nicht AS sein.

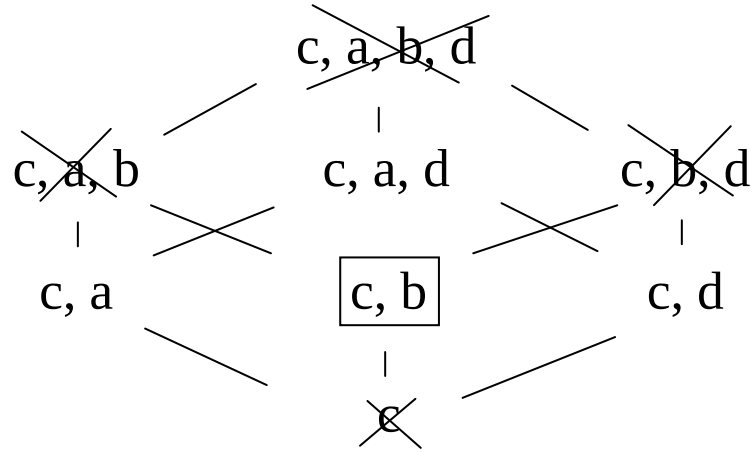
Beweis für 2):
Sei P ein Logikprogramm, seien S, S' answer sets von P. Wir nehmen an $S \subset S'$.
Da S' echte Obermenge von S ist, gilt $P^{S'} \subseteq P^S$ (mindestens so viele Regeln werden durch S' widerlegt wie durch S). Damit ist $Cn(P^{S'}) \subseteq Cn(P^S)$. Da S answer set ist, gilt $Cn(P^S) = S$ und damit $Cn(P^{S'}) \subset S'$. S' ist also kein answer set, im Widerspruch zur Annahme.

- Also kann man systematisch Teilmengen der Regelköpfe überprüfen und, falls AS gefunden, Ober- bzw. Untermengen aus dem Suchraum streichen.

Beispiel:

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 {a, b, d}
{c,b} answer set: Ober- und Teilmengen fallen weg



es bleiben {c,a}, {c,a,d}, {c,d}: keine dieser Mengen ist answer set

Ableitungsbäume

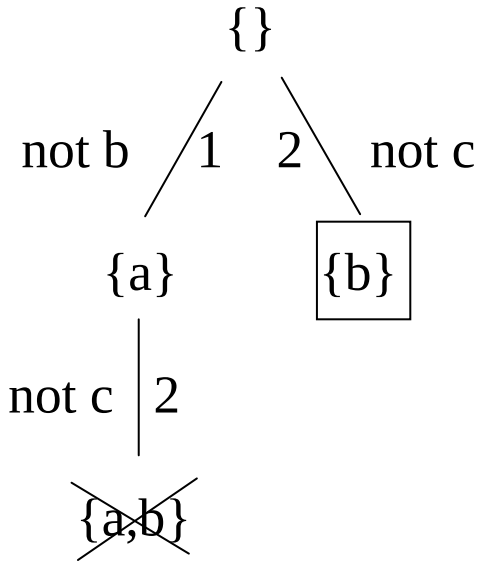
Motivation: betrachte folgendes Programm

1) a ← not b
2) b ← not c

Probiere der Reihe nach Regeln anzuwenden, deren positive Vorbedingungen bereits hergeleitet sind und deren negierte noch nicht widerlegt sind:

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

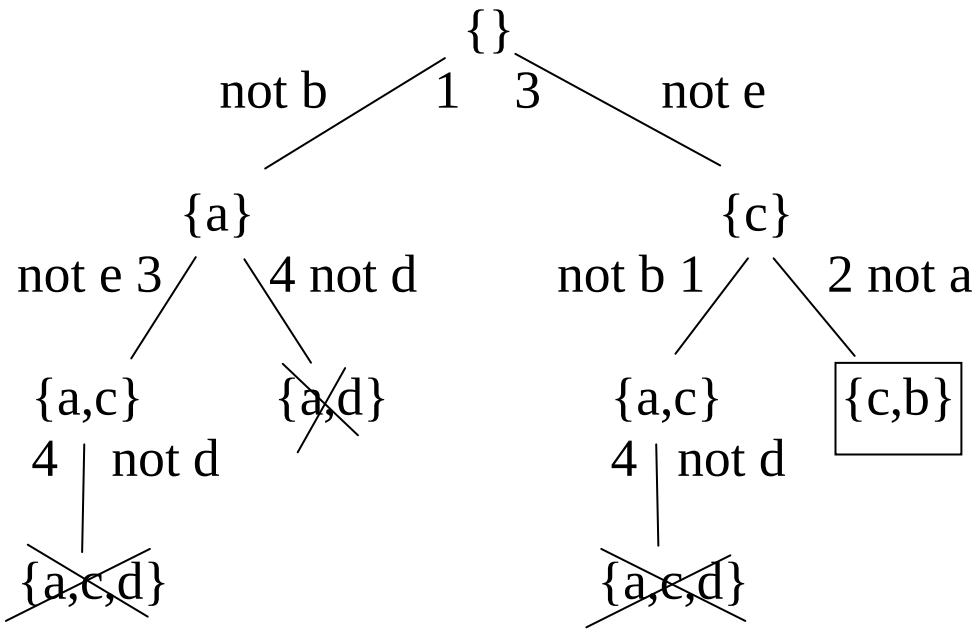
Also: merken, was nicht abgeleitet werden darf!!



- Knoten enthalten bereits abgeleitete Atome; Wurzel: Fakten
- Kanten entsprechen anwendbaren (und im Pfad noch nicht angewendeten) Regeln: positive Vorbedingungen der Regel in Knoten, default-negierte nicht
- Nachfolgerknoten bekommt Kopf der Regel dazu, default-negierte Atome im Körper der Regel an Kanten markiert
- Falls Knoten Atom enthält, das negiert an Kante zu ihm liegt: kein answer set
- Falls keine weitere Regel mehr anwendbar: AS gefunden

Beispiel

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



Smodels Algorithmus (TU Helsinki)

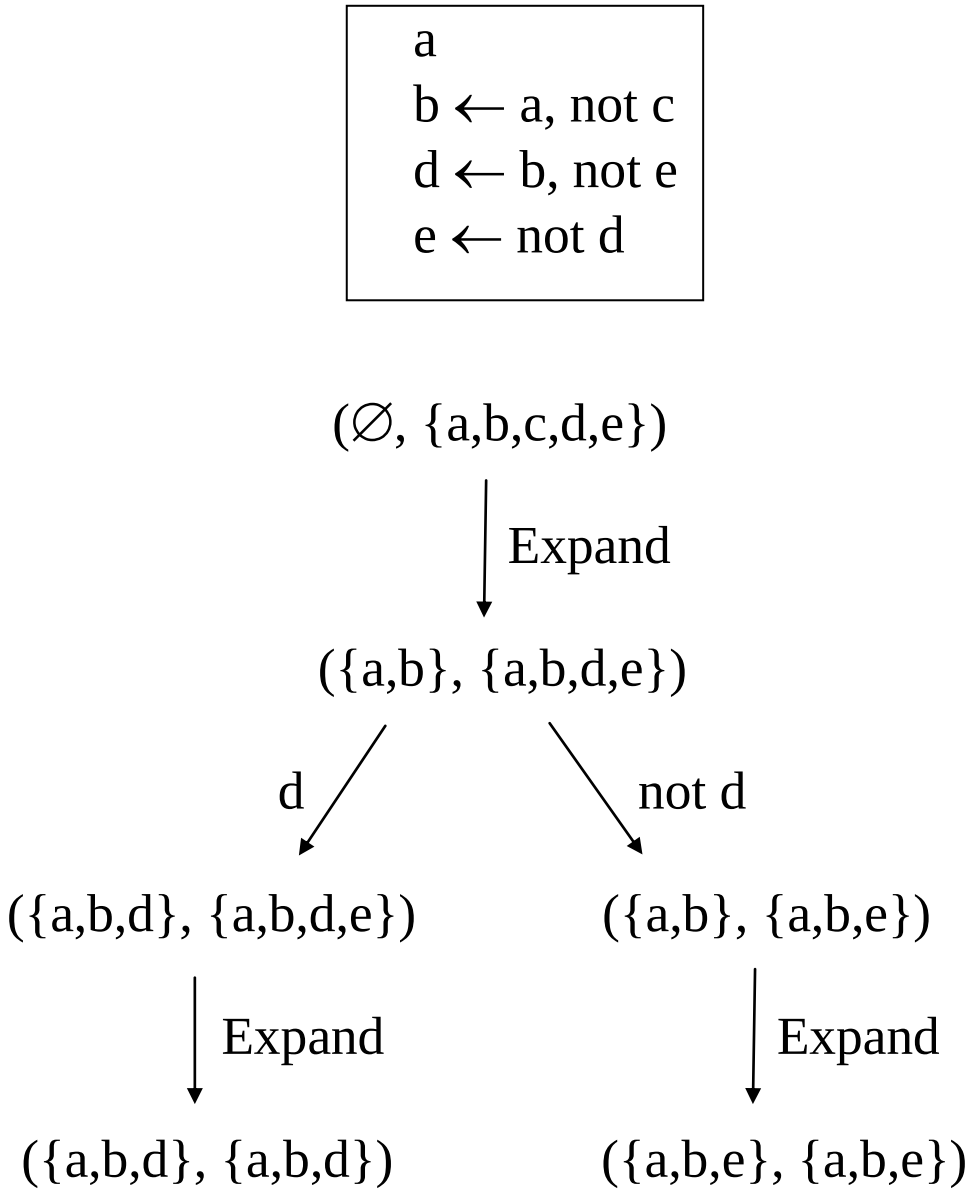
Sei P ein (normales) Logikprogramm. $L \subseteq U$ seien Mengen von Atomen.
S ist eine (L,U)-Antwortmenge von P gdw. S Antwortmenge von P und $L \subseteq S \subseteq U$.

also*: Antwortmenge, die alle Elemente in L enthält, nur Elemente in U.*

Bemerkung: S ist Antwortmenge von P gdw. S ist (∅, Atoms(P)) Antwortmenge von P.
Atoms(P): Menge aller Atome in P.

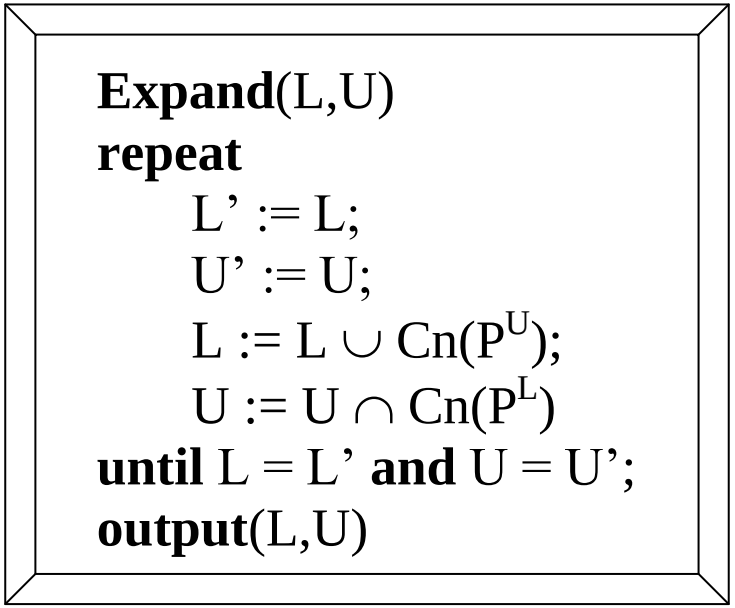
Grundidee von Smodels:

- Erzeuge Binärbaum mit Wurzel (∅, Atoms(P)).
- Erzeuge für jeden Knoten (L,U) einen „expandierten“ Nachfolgerknoten (L’,U’) mit $L \subseteq L'$ und $U' \subseteq U$, so dass die (L,U)-Antwortmengen und die (L’,U’)-Antwortmengen identisch sind
- Wähle für jeden bereits expandierten Knoten (L,U) ein Element a aus U\L. Verzweige und erzeuge Knoten für ($L \cup \{a\}$, U) und (L, U\{a}).
- Wenn nach Expandieren irgendwann L = U => Antwortmenge gefunden
- Wenn $U \subset L$ => Sackgasse.


a
b ← a, not c
d ← b, not e
e ← not d
(∅, {a,b,c,d,e})
Expand
({a,b}, {a,b,d,e})
d
not d
({a,b,d}, {a,b,d,e})
({a,b}, {a,b,e})
Expand
Expand
({a,b,d}, {a,b,d})
({a,b,e}, {a,b,e})

Wie funktioniert Expand?

- Nutze Wissen über Atome, die in AS sein müssen (die in L), und solche, die nicht in AS sein können (diejenigen, die nicht in U sind).
- Gesucht (L,U)-Antwortmengen von P.
- Wenn nur Elemente aus U als Elemente der AS in Frage kommen, dann müssen die Atome, die aus Regeln herleitbar sind, die nicht durch U widerlegt werden, auf jeden Fall auch in AS sein: das sind die in $Cn(P^U)$ => ersetze L durch $L \cup Cn(P^U)$.
- Wenn die Atome in L in AS sein müssen, dann können nur noch solche Atome in AS sein, die aus Regeln herleitbar sind, die nicht von L widerlegt werden: das sind die in $Cn(P^L)$ => ersetze U durch $U \cap Cn(P^L)$.
- Iteriere die Berechnung dieser Mengen so lange, bis sich nichts mehr ändert.

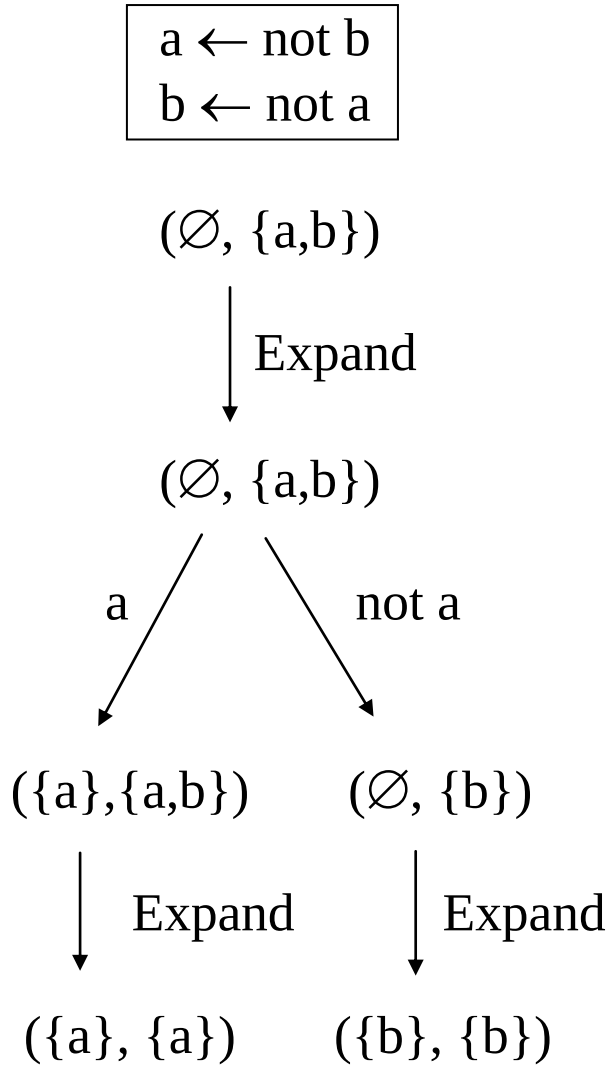


Cn(R) Hülle des (definiten) Programms R
im Beispiel: wir starten mit Expand(∅, {a,b,c,d,e}). Wir erhalten schrittweise:

$$L = Cn(P^{\{a,b,c,d,e\}}) = \{a\}$$
$$U = Cn(P^{\{a\}}) = \{a,b,d,e\}$$
$$L = Cn(P^{\{a,b,d,e\}}) = \{a,b\}$$
$$U = Cn(P^{\{a,b\}}) = \{a,b,d,e\}$$

ab hier passiert nichts mehr.

Programm 2:



zwei answer sets!

Programm 3:

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

$(\emptyset, \{a,b,c\})$

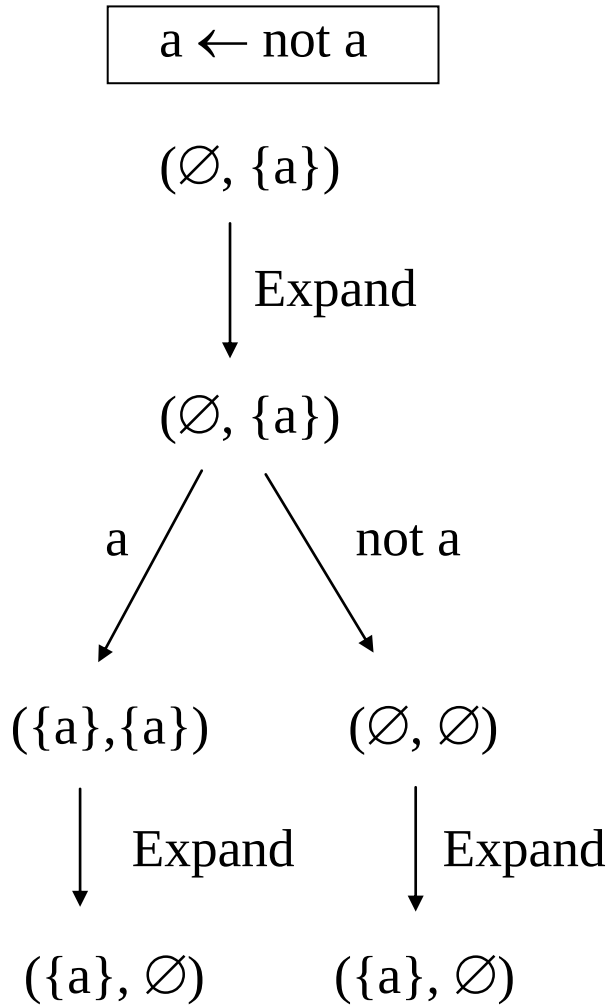
↓ Expand

$(\{b\}, \{b\})$

Hier macht Expand alles alleine! Folge der Belegungen von L und U:

L:	$\emptyset$	$L \cup Cn(P^U) = \emptyset$	$L \cup Cn(P^U) = \{b\}$	keine Änderung
U:	$\{a,b,c\}$	$U \cap Cn(P^L) = \{a,b\}$	$U \cap Cn(P^L) = \{b\}$	keine Änderung

Programm 4:



kein answer set!

Programme mit Kardinalitätsconstraints

Seien u, o natürliche Zahlen, $u \leq o$

$$u\{a_1, ..., a_n\}o$$

bedeutet: mindestens u, höchstens o der a's sind wahr.

Können im Kopf wie im Körper von Regeln vorkommen

$$2\{a_1, ..., a_7\}3 \leftarrow 1\{b_1, ..., b_3\}2$$

wenn 1 oder 2 der b's wahr sind, so sind 2 oder 3 a's wahr.

Falls u fehlt: $u = 0$; falls o fehlt: $o = \infty$

Semantik:
Programmreduktion ähnlich ursprünglicher AS-Semantik (aber viel komplizierter!)

Notation: $0\{on(A,B,T): block(A)\}1 \leftarrow block(B), time(T)$

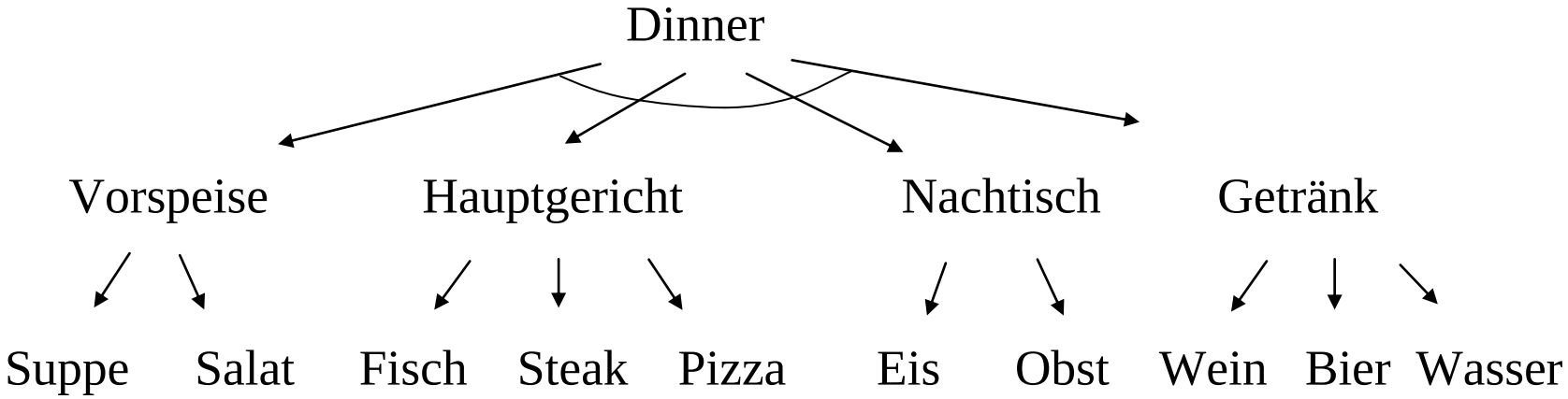
erst Grundinstanzen der Regel gebildet (für jeden Block b und Zeitpunkt t):

$$0\{on(A,b,t): block(A)\}1 \leftarrow block(b), time(t)$$

dann Ausdruck {...} ersetzt durch (a, b, c seien alle Instanzen von block!):

$$0\{on(a,b,t), on(b,b,t), on(c,b,t)\}1 \leftarrow block(b), time(t)$$

Anwendungsbeispiel Konfiguration: Häufig in Form von AND/OR Bäumen:



Repräsentation mir Kardinalitätsconstraints:

$$\begin{aligned} \text{dinner} &\leftarrow \\ 4\{\text{vorspeise, hauptgericht, nachtisch, getränk}\}4 &\leftarrow \text{dinner} \\ 1\{\text{suppe, salat}\}1 &\leftarrow \text{vorspeise} \\ 1\{\text{fisch, steak, pizza}\}1 &\leftarrow \text{hauptgericht} \\ 1\{\text{eis, obst}\}1 &\leftarrow \text{nachtisch} \\ 1\{\text{wein, bier, wasser}\}1 &\leftarrow \text{getränk} \end{aligned}$$

Hinweis: Smodels kennt auch minimize und optimize statements:

minimize[a_1, …, a_n] besagt: nur die AS sollen geliefert werden, in denen eine minimale Anzahl der a's enthalten ist.

Reviewer Assignment

Probleminstanz:

```
reviewer(r1), …          % die verfügbaren Reviewer
paper(p1), …             % die eingereichten Papers
classA(r1, p1), …        % die bevorzugten papers der Reviewer
classB(r1, p2), …        % die akzeptablen papers der Reviewer
coi(r1,p3), …            % Interessenskonflikte
```

Problem:

Jedes Paper hat genau 3 Reviewer:

$$3\ \{assigned(P,R): reviewer(R)\}\ 3 \leftarrow paper(P)$$

Ein Paper darf nicht einen Reviewer mit Interessenskonflikt zugewiesen werden:

$$\leftarrow assigned(P,R), coi(R,P)$$

Kein Reviewer hat ein ungewolltes Paper:

$\leftarrow$ paper(P), reviewer(R), assigned(P,R), not classA(R,P), not classB(R,P)

Kein Reviewer hat mehr als 8 Papers:

$\leftarrow$ 9 {assigned(P,R): paper(P)}, reviewer(R)

Jeder Reviewer hat mindestens 7 Papers:

$\leftarrow$ {assigned(P,R): paper(P)} 6, reviewer(R)

Jeder Reviewer hat höchstens 2 classB Papers:

assignedB(P,R) $\leftarrow$ classB(R,P), assigned(P,R)
$\leftarrow$ 3 {assignedB(P,R): paper(P)} , reviewer(R)

Minimiere Zahl der classB Papers:

minimize [assignedB(P,R): paper(P): reviewer(R)]