Kalkül | Regeln | Quantoren | Konjunktion | Symmetrie | Vollständigkeit | Semantik | Prädikat | Funktionssymbole

Fügen sie die fehlenden Begriffe an den richtigen Stellen ein!

Die Prädikatenlogik ist mächtiger, da man mit einem [Prädikat] im Regelfall viele gleichartige Zustände zusammenfassen und dieses dann abstrakt durch [Regeln] definieren kann. Die Syntak der Prädikatenlogik sieht neben Variablen und Konstanten auch [Funktionssymbole] vor, die wiederum auf Terme angewendet werden können. Ein wichtiges Element der Prädikatenlogik sind [Quantoren], mit denen Regeln festgelegt werden können.

Die [Semantik] der Prädikatenlogik definiert die Formeln rekursiv, indem Objekte der realen Welt den Konstanten, den Variablen und den Funktionssymbolen zugeordnet werden. Eine Wissensbasis setzt sich aus Regeln und Fakten zusammen, die per [Konjunktion] miteinander verknüpft sind und per [Kalkül] abgefragt werden können. Es gelten diverse Axiome wie die Reflexivität oder die [Symmetrie]. Die [Vollständigkeit] und Korrektheit einer PL-basierten Wissensbasis ist zu gewährleisten.

Input:
Assume that the input is a set of facts specified by the following predicates:

- adjacent(X, Y): country X is adjacent to country Y
- eu(X): country X is a member of the European Union
- candidate(X): country X is a candidate member
- euro(X): country X uses the Euro
- schengen(X): country X belongs to the Schengen area

Note: The name of a state is given by a string constant.

Task:
Write an ASP program that computes all Schengen countries that share a border with a Non-Schengen country. For each such county C, derive an atom outerBorder(C).

Zum Beispiel:

Test	Resultat
#show outerBorder/1.	outerBorder("Bulgaria") outerBorder("Estonia") outerBorder("Finland") outerBorder("France") outerBorder("Greece") outerBorder("Hungary") outerBorder("Italy") outerBorder("Latvia") outerBorde

Antwort: (Abzugssystem: 10, 20, ... %)

Antwort zurücksetzen

```
outerBorder(X) :- schengen(X), adjacent(X,Y), not schengen(Y).
#show outerBorder/1.
```

Welche Definitionen zur Aussagenlogik sind korrekt?

Wählen Sie eine oder mehrere Antworten:

- [ ] a. Die Ausdrücke ***C v A ^ B***, ***((A) v B)*** und ***A B*** sind syntaktisch wohlgeformt.
- [x] b. $A \Rightarrow B$ ist nur falsch, wenn die erste Aussage A (Vordersatz, Implikans) wahr und die zweite Aussage (Hintersatz, Implikat) falsch ist. Die Implikation kann auch über die Negation und die Disjunktion ($(\neg A \vee B)$) ausgedruckt werden.
- [ ] c. Eine aussagenlogische Formel mit n Variablen besitzt 2 * n verschiedene Belegungen.
- [ ] d. ***A ^ B*** stellt eine atomare, aussagenlogische Formel dar.
- [x] e. Die Syntax einer Aussagenlogik bestimmt die Wohlgeformtheit von Ausdrücken, die aus Symbolen und logischen Operatoren bestehen sowie beginnend mit atomaren Formeln rekursiv definiert werden.
- [ ] f. Bei komplexeren Formeln ist es notwendig, eine Reihenfolge der Operatoren festzulegen. Dabei hat die Negation Vorrang vor der Klammerung, dieser wiederrum vor der Konjunktion, der Disjunktion, der Äquivalenz und schließlich der Implikation.
- [x] g. Die Belegung bzw. Interpretation einer Aussagenlogik meint, dass man jeder Aussagenvariable einen Wahrheitswert (wahr oder falsch) zuweist.

Weisen sie dem beschriebenen Szenario die geeignete Technik oder das geeignete Beweisverfahren zu!

Können mehrere aussagenlogische Formeln nicht gleichzeitig erfüllt werden, so nennt man dies ...	Widerspruch
Eine digitale Schaltung ist verifiziert man aufgrund des großen Umfangs in der Regel durch ...	Automatisierte Beweiser
Ein Beweisverfahren, das versucht die leere Klausel abzuleiten um so einen Widerspruch in der ursprünglichen Klauselmenge zu finden, nennt man ...	Resolution
Die Folgerung von Aussagen aus einer Wissensbasis nennt man ...	Deduktion
Die Abbildung von Expertenwissen in Form von aussagenlogischer Formeln nennt man ...	Wissensrepräsentation

Input:
Assume that the input is a set of facts specified by the following predicates:

- childOf(X, Y): X is a child of Y
- female(X): X is female
- male(X): X is male

Task:
Write an ASP program that derives atoms over the following predicates:

- cousin(X, Y): whenever X is a cousin of Y

You can (but do not have to) use helper predicates, but make sure to keep the #show directives in the answer box!

Zum Beispiel:

Test	Resultat

`childOf(bart,homer). childOf(lisa,homer). childOf(maggie,homer).`
`childOf(homer,abe). childOf(herbert,abe).`
`childOf(marge,jacqueline). childOf(patty,jacqueline). childOf(selma,jacqueline).`
`childOf(bart,marge). childOf(lisa,marge). childOf(maggie,marge).`
`female(lisa). female(maggie). female(marge). female(jacqueline). female(patty). female(selma).`
`male(bart). male(homer). male(abe). male(herbert).`	

Antwort: (Abzugssystem: 10, 20, ... %)

Antwort zurücksetzen

```
1 sibling(A,B) :- childOf(A,C), childOf(B,C), A != B.
2 cousin(X,Y) :- childOf(X,D), childOf(Y,E), sibling(D,E).
3 #show cousin/2.
```

Verbleibende Zeit 0:16:56

Input:

The input defines a graph as follows:

- node(X) defines that X ist a node
- arc(X, Y) defines that there is a (directed) arc from X to Y

Task:

Write an ASP program that computes all nodes X that have both outgoing and incoming arcs. The result is to be encoded using atoms of form output(.).

Zum Beispiel:

Test	Resultat

`node(X) :- arc(X, Y).`
`node(Y) :- arc(X, Y).`
`arc(a, b). arc(b, c). arc(c, d). arc(e, f). arc(f, a). arc(a, g). arc(g,a). arc(h, i). arc(i, j). arc(j, i).`
`start(a).`
	`output(a) outpu`

Antwort: (Abzugssystem: 10, 20, ... %)

Antwort zurücksetzen

```
1  #show output/1.
2
3  output(X) :- arc(X, Y), arc(Z, X).
```

- birds can fly, except for penguins, who cannot fly

Use the following predicates:

- bird(X) specifies that X is a bird
- fly(X) specifies that X can fly
- -fly(X) specifies that X cannot fly

You can (but do not have to) use helper predicates, but make sure to keep the #show directives in the answer box!

Zum Beispiel:

Test	Resultat
`eagle(eddy).` `penguin(tux).`	`-fly(tux) bird(eddy) bird(tux) eagle(eddy) fly(eddy) penguin(tux)`

Antwort: (Abzugssystem: 10, 20, ... %)

Antwort zurücksetzen

```
1  #show bird/1.
2  #show fly/1.
3  #show -fly/1.
4  #show penguin/1.
5  #show eagle/1.
6
7  bird(X) :- eagle(X).
8  bird(X) :- penguin(X).
9  fly(X) :- bird(X), not penguin(X).
10 -fly(X) :- penguin(X).
```

%BSP1

outerBorder(X) :- schengen(X), not schengen(Y), adjacent(X, Y).

```
#show outerBorder/1.


%BSP2
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(X) :- eagle(X).


#show bird/1.
#show fly/1.
#show -fly/1.
#show penguin/1.
#show eagle/1.


%BSP3
euborder(X) :- eu(X), adjacent(X, Y), not schengen(Y).
newschengen(X) :- eu(X).
newschengen(X) :- schengen(X).
newschengen(X) :- candidate(X).
newschengenborder(X) :- newschengen(X), adjacent(X, A), not newschengen(A).

result(X) :- euborder(X), not newschengenborder(X).

#show result/1.


%BSP4
sibling(X, Y) :- childOf(X, A), childOf(Y, A), X != Y.
cousin(A, B) :- childOf(A, X), childOf(B, Y), sibling(X, Y).
```

```
#show cousin/2.
```

```
%BSP5
result(X) :- adjacent(X, A), adjacent(X, B), adjacent(X, C), adjacent(X, D), adjacent(X, E),
A != B, A != C, A != D, A != E, B != C, B != D, B != E, C != D, C != E, D != E.

#show result/1.
```

```
%BSP6
reachable(X) :- conn(Y, X, metro, U), start(Y).
reachable(X) :- conn(Y, A, metro, U), conn(A, X, metro, V), start(Y).
reachable(X) :- conn(Y, A, metro, U), conn(A, B, metro, V), conn(B, X, metro, W), start(Y).

#show reachable/1.
```

```
%BSP7
father(X, Y) :- male(X), childOf(Y, X).
mother(X, Y) :- female(X), childOf(Y, X).

#show father/2.
#show mother/2.
```

%BSP8

result(X) :- state(X), adjacent(missouri, Y), adjacent(Y, X), not adjacent(missouri, X), X != missouri, Y != missouri.

#show result/1.

%BSP9

output(X) :- node(X), arc(X, Y), arc(Z, X).

#show output/1.

%BSP10

reachable(X) :- conn(X, Y, metro, U), reachable(Y).

reachable(X) :- start(X).

#show reachable/1.