Artificial Intelligence

Abduction

The point of this lab is to define a predicate

\texttt{abduce(?ExL,+ObL,+KB,+As)}

which, given lists \texttt{ObL}, \texttt{KB} and \texttt{As}, returns a list \texttt{ExL} of explanations for the list \texttt{ObL} of observations, relative to the default reasoning framework \texttt{(KB,As)}, where

- \texttt{KB} lists the knowledge base that is taken for granted

and

- \texttt{As} lists the assumables (or hypotheses), instances of which may be included in \texttt{ExL}.

For simplicity, let us restrict \texttt{ObL}, \texttt{ExL} and \texttt{As} to lists of goals. As for \texttt{KB}, let us assume that \texttt{KB} is a pair \texttt{[KB0,KB1]} of lists \texttt{KB0} and \texttt{KB1} of definite clauses and integrity constraints (respectively) such that

- \texttt{KB0} encodes the definite clause

\[
h :- b_1,...,b_n
\]

as the list

\[
\texttt{[h | [b_1,...,b_n]]}
\]

with the case \( n=0 \) encoding the goal \( h \) as \( \texttt{[h]} \)

and

- \texttt{KB1} lists the integrity constraint

\[
\text{false} :- b_1,...,b_n
\]

as

\[
\texttt{[b_1,...,b_n]}
\]

so that \texttt{KB1} allows us to detect conflicts.

\textbf{Question.} Why separate definite clauses from integrity constraints in \texttt{KB}?
Step 1: deduction

Define a predicate `deduce(G,KB0)` that checks if `G` can be deduced from `KB0`.  

*Hint.* Fill in the question marks `???` below.

```
deduce(G,KB0) :- member([G|???],KB0),  
deduceAll(???,???).
```

```
deduceAll([],__).
```

```
deduceAll([G|More],KB0) :- deduce(G,KB0),  
deduceAll(???,???).
```

Step 2: consistency

Define a predicate `consistent(KB)` to check that `KB` is consistent.

```
inconsistent([KB0,KB1]) :- member(???,KB1),  
deduceAll(???,???).
```

```
consistent(KB) :- inconsistent(KB),!,fail ; true.
```

Step 3: abduction

Finally, let us define `abduce(ExL,ObL,KB,As)`.

```
abduce(???,[]_,__).
```

```
abduce([Ob|ExL],[Ob|More],[KB0,KB1],As) :-  
  member(Ob,As),  
  NewKB0 = [[Ob]|KB0],  
  consistent([NewKB0,???]),  
  abduce(ExL,???,[NewKB0,KB1],As).
```

```
abduce(ExL,[Ob|More],[KB0,KB1],As) :-  
  member([Ob|Body],???),  
  append(Body,More,NewOb),  
  abduce(ExL,NewOb,???,As).
```

P.S. Explanation

Define a predicate `explain(G,ExL,KB,As)` to check that `ExL` is an explanation in `(KB,As)` of the goal `G`.

Sample runs

```
| ?- abduce(ExL,[a,b],[[a,d],[b,c],[d,e]],[[e,c]],[d,c]).

ExL = [d,c] ? ;
```

| ?- abduce(ExL, [a, b], [[[a, d], [b, c], [d, e]], [[a, c]], [d, c]]).
| no

| ?- abduce(ExL, [a, b], [[[a, d], [b, c], [d, e]], [[a, c]], [d, b]]).
ExL = [d, b] ;
| no

| ?- abduce(ExL, [fly(sean)], [[[fly(X), bird(X)]], [[bird(Y), man(Y)]], [bird(Z)]].
X = sean,
Z = sean,
ExL = [bird(sean)] ;
| no

| ?- abduce(ExL, [fly(sean)], [[[fly(X), bird(X)]], [[bird(sean)]], [bird(Z)]).
no

| ?- abduce(ExL, [fly(sean)], [[[fly(X), bird(X)]], [[bird(sean)]], [fly(Z)]).
Z = sean,
ExL = [fly(sean)] ;
| no

| ?- explain(a, ExL, [[[a, b, c], [a, d, e], [d, f]], [[d, b]], [b, c, f, e]]).
ExL = [b, c] ;
ExL = [f, e] ;
| no
| ?- explain(a,ExL,[[[a,b,c],[a,d,e],[d,f]],[[d,b],[e,f]]],[b,c,f,e]).

ExL = [b,c] ? ;

no