Artificial Intelligence

Abduction

The point of this lab is to define a predicate

\[
\text{abduce}(\text{ExL}, \text{ObL}, +\text{KB}, +\text{As})
\]

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

- KB lists the knowledge base that is taken for granted

and

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

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

- KB0 encodes the definite clause

\[
h \leftarrow b_1, \ldots, b_n
\]

as the list

\[
[h \mid [b_1, \ldots, b_n]]
\]

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

and

- KB1 lists the integrity constraint

\[
\text{false} \leftarrow b_1, \ldots, b_n
\]

as

\[
[b_1, \ldots, b_n]
\]

so that KB1 allows us to detect conflicts.

**Question.** Why separate definite clauses from integrity constraints in KB?
Step 1: inference

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

_Hint._ Fill in the question marks ??? below.

```prolog
infer(G, KB0) :- member([G|???>], KB0),
               inferAll(???, ???).
```

```prolog
inferAll([], _).
```

```prolog
inferAll([G|More], KB0) :- infer(G, KB0),
                          inferAll(???, ???).
```

Step 2: consistency

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

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

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

Step 3: abduction

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

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

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

```prolog
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

```prolog
| ?- 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