LKR – SD206
(Logic and Knowledge Representation)

Jean-Louis Dessalles

Evaluation - April 2022

Duration: **1 hour 30 min**. No documents - No turned-on devices. Questions are independent.

Q1. Cities are located on a *one-way* road. Show how to complete the following program so as to check whether one can travel from one city to another.

```
oneWayRoad([lussac, gayac, figeac, trelissac, tourtoirac, dignac,
fronsac, agonac, jumillac]).

travel(City1, City2) :-

      oneWayRoad(R),

      path(R, City1, City2).
```

Q2. Can the following pairs of predicates be unified (provide the resulting substitutions if yes).

a. `p(X, f(X), Z)` and `p(g(Y),f(g(b)), Y)`

b. `p(X, f(X))` and `p(f(Y), Y)`

c. `p(X, f(Z))` and `p(f(Y), Y)`

Q3. Consider the following axioms:

1. Every child loves Santa.
2. Everyone who loves Santa loves any reindeer.
3. Rudolph is a reindeer, and Rudolph has a red nose.
4. Anything which has a red nose is weird or is a clown.
5. No reindeer is a clown.
6. Scrooge does not love anything which is weird.
7. (Conclusion) Scrooge is not a child.

Represent these axioms in predicate calculus; convert each formula to clause form. (Notes: ‘has_a_red_nose’ can be a single predicate. Remember to negate the conclusion.) Prove the unsatisfiability of the set of clauses by resolution.

Q4. Provide a model in which $(\forall x)\ (P(x) \supset (\forall y)\ P(y))$ is true.

Q5. The following DCG recognizes an xml tag (we suppose that input is given as a list of ASCII codes):

```
tag(S) --> [60], str(S), [62].     % 60 is the code for '<' and 62 for '>'
str([X|S]) --> [X], str(S), {X \== 60, X \== 62}.
str([ ]) --> [ ].
```

Write DCG predicate `xml` that checks whether xml tags are well balanced. For instance, `xml` should succeed on the string:

`"<x1>I know that <h1>Prolog </h1>is logical</x1>"`

but is should fail on

`"<x1>I know that <h1>Prolog </x1>is logical</h1>"`

(note: code for `'/'` is 47).

Q6. Provide the best generalization (lgg) for these two examples of the concept `nice_food`:

```
nice_food(X) :- fruit(X), round(X), red(X), juicy(X).

nice_food(X) :- edible(X), yellow(X), sweet(X), has_seeds(X).
```

using the background knowledge :

```
edible(X) :- fruit(X).

juicy(X) :- sweet(X).

edible(X) :- sweet(X).
```

Solutions

Q1. Cities are located on a *one-way* road. Show how to complete the following program so as to check whether one can travel from one city to another.

```
oneWayRoad([lussac, gayac, figeac, trelissac, tourtoirac, dignac,
fronsac, agonac, jumillac]).

travel(City1, City2) :-

      oneWayRoad(R),

      path(R, City1, City2).
```

```
path([City1|R], City1, City2) :-
    !,     % cut for efficiency
    member(City2, R).

path([_|R], City1, City2) :-
    path(R, City1, City2).
```

Q2. Can the following pairs of predicates be unified (provide the resulting substitutions if yes).

a. `p(X, f(X), Z)` and `p(g(Y),f(g(b)), Y)`

b. `p(X, f(X))` and `p(f(Y), Y)`

c. `p(X, f(Z))` and `p(f(Y), Y)`

a. Yes: `X = g(b), Z = Y, Y = b.`

b. No. **We would need `Y = f(f(Y))`, but recursive unification should fail (note: most Prolog implementation do not check for recursion for efficiency purposes).**

c. Yes: `X = f(f(Z)), Y = f(Z).`

Q3. Consider the following axioms:

1. Every child loves Santa.
2. Everyone who loves Santa loves any reindeer.
3. Rudolph is a reindeer, and Rudolph has a red nose.
4. Anything which has a red nose is weird or is a clown.
5. No reindeer is a clown.
6. Scrooge does not love anything which is weird.
7. (Conclusion) Scrooge is not a child.

Represent these axioms in predicate calculus; convert each formula to clause form. (Notes: 'has_a_red_nose' can be a single predicate. Remember to negate the conclusion.) Prove the unsatisfiability of the set of clauses by resolution.

1. **[($\forall$x) (child(x) $\supset$ loves(x, santa))]**
2. **[($\forall x$) (loves(x, santa) $\supset$ ($\forall y$) (reindeer(y) $\supset$ loves(x, y)))]**
3. **[reindeer(Rudolph) ^ has_a_red_nose(Rudolph)]**

4. **[(∀*x*) (has_a_red_nose(*x*) ⊃ (weird(*x*) ∨ clown(*x*)))]**
5. **[(∀*x*) ¬(reindeer(*x*) ∧ clown(*x*))]**
6. **[(∀*x*) (weird(*x*) ⊃ ¬loves(Scrooge, *x*))]**
7. **[child(Scrooge)]** (negation of the conclusion)
8. **[¬child(*x*), loves(*x*, santa)]** (rewriting 1.)
9. **[¬loves(*x*, santa), ¬reindeer(*y*), loves(*x*, *y*)]** (rewriting 2.)
10. **[reindeer(Rudolph)]** (from 3)
11. **[has_a_red_nose(Rudolph)]** (from 3)
12. **[¬has_a_red_nose(*x*), weird(*x*), clown(*x*)]** (rewriting 4.)
13. **[¬reindeer(*x*), ¬clown(*x*))]** (rewriting 5.)
14. **[¬weird(*x*), ¬loves(Scrooge, *x*)]** (rewriting 6.)
15. **[loves(Scrooge, santa)]** (resolution of 7. and 8.)
16. **[¬reindeer(*y*), loves(Scrooge, *y*)]** (resolution of 14. and 9.)
17. **[loves(Scrooge, Rudolph)]** (resolution of 10. and 16.)
18. **[weird(Rudolph), clown(Rudolph)]** (resolution of 11. and 12.)
19. **[¬clown(Rudolph)]** (resolution of 10. and 13.)
20. **[weird(Rudolph)]** (resolution of 18. and 19.)
21. **[¬loves(Scrooge, Rudolph)]** (resolution of 14. and 20.)
22. **[ ]** (resolution of 15. and 21.)

Q4. Provide a model in which $(\forall x)\,(P(x) \supset (\forall y)\,P(y))$ is true.

Consider a model with a single element {a}.

Q5. The following DCG recognizes an xml tag (we suppose that input is given as a list of ASCII codes):

```
tag(S) --> [60], str(S), [62].     % 60 is the code for '<' and 62 for '>'
str([X|S]) --> [X], str(S), {X \== 60, X \== 62}.
str([ ]) --> [ ].
```

Write DCG predicate xml that checks whether xml tags are well balanced. For instance, xml should succeed on the string:

"<x1>I know that <h1>Prolog </h1>is logical</x1>"

but is should fail on

"<x1>I know that <h1>Prolog </x1>is logical</h1>"

(note: code for '/' is 47).

```
xml --> tag(T), text, tag([47|T]), {write('recognized: '), writef(T), nl}.

text --> str(_), xml, text.

text --> str(_).

    ?- xml(`<x1>I know that <h1>Prolog </h1>is logical</x1>`, []).
    'recognized: h1
    'recognized: x1
    true .
```

```
?- xml(`<x1>I know that <h1>Prolog </x1>is logical</h1>`, []).
false.
```

Q6. Provide the best generalization (lgg) for these two examples of the concept `nice_food`:

```
nice_food(X) :- fruit(X), round(X), red(X), juicy(X).

nice_food(X) :- edible(X), yellow(X), sweet(X), has_seeds(X).
```

using the background knowledge :

```
edible(X) :- fruit(X).

juicy(X) :- sweet(X).

edible(X) :- sweet(X).
```

the lgg: nice_food(X) :- edible(X), juicy(X).