Logic, Knowledge Representation and Probabilities
Problem solving and Knowledge representation

Nils Holzenberger

March 4, 2025

TODO

- Logic,
- Knowledge Representation and
- Probabilities

TODO

- Logic, ← *March 11 and 18*
- Knowledge Representation and
- Probabilities ← *April 1 and 8*

Outline

1. Recap last lecture
2. Monkey problem-solving
   - The monkey, the box and the banana
   - The solving
   - Inverting a list
3. Some more Prolog operators
   - Queries in Prolog
   - Cut operator
   - Prolog predicate surgery
4. Knowledge representation
   - Expert systems
   - Ontologies

list_length

`list_length(L,N)` is true when the list `L` contains `N` elements

```
list_length([],0).
list_length([_|T],N) :-
    list_length(T, N1),
    N is N1+1.
```

The following version failed:

```
list_length([],0).
list_length([_|T],N) :-
    N1 is N-1,
    list_length(T, N1).
```

extract

`extract(X,List,Remainder)` takes an element from a list: it succeeds if `Remainder` is obtained by removing `X` from `List`

`extract(a,[a,b,c],[b,c])` succeeds

`extract(b,[a,b,c],[b,c])` fails

Solution:

```
extract(X, [X|T], T).
extract(X, [H|T], [H|Remainder]) :-
    extract(X, T, Remainder).
```

This can be called in different ways:

`extract(b,[a,b,c],L).`

`extract(b,L,[b,c]).`

`extract` can both insert and extract elements from a list. This property is known as *reversibility*.

1. Recap last lecture

2. Monkey problem-solving
   - The monkey, the box and the banana
   - The solving
   - Inverting a list

3. Some more Prolog operators

4. Knowledge representation

The task

- Description: see lab session and `monkey.pl`
- The world can be fully described by its *state*
- It is possible to go from one state to another using *actions*
- This vocabulary comes from *reinforcement learning*
- `state(MonkeyXPos, MonkeyYPos, BoxPos, BananaHolding)`
- `action(StartingState, ActionID, EndState)`

The solving

- Run the monkey program with `trace`
- Individual actions may be taken before they are known
- They remain as variables until unified at the end
- How does Prolog choose the next action to take?
- Prolog uses backtracking: the Prolog solver tries all possibilities in the order they appear in
- Exercise: tower of Hanoi
  `https://ailab.r2.enst.fr/LKR/Hanoi.html`

Inverting a list

Write the predicate `invert(List, Reverse)` where `Reverse` contains the same elements as `List` in the reverse order.

See `invert.pl`

findall and setof

- Very useful predicates
- `findall` and `setof` run an exhaustive query and return the results
- Some results may appear multiple times – see `sister` example

See `simpsons.pl`

!

- Write the predicate `purge(List, ListNoDuplicates)` where `ListNoDuplicates` contains exactly one copy of the elements of `List`
- Try to use it in reverse
- *Green cut* vs *red cut*

See `purge.pl`

`assert` and `retract`

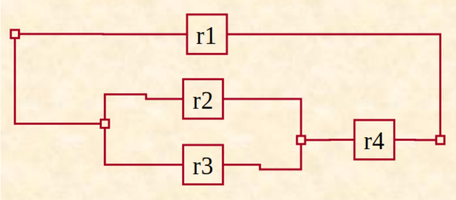
- The use of `assert` and `retract` should generally be avoided
- `assert` adds a clause to Prolog's memory
- `retract` removes a clause from Prolog's memory
- Both work like Prolog predicates: they succeed or fail
- `assert` succeeds if the predicate already exists
- `retract` succeeds if it unifies with something in the memory

See `weather.pl`

Data structures

Some simple examples:

- Dates `date(day(15), month(1), year(2024))`
- Electric circuit `par(r1, seq(par(r2, r3), r4))`



- We will see object-oriented programming in a minute

Automatons

```
final(s3).

trans(s1,a,s1).
trans(s1,a,s2).
trans(s1,b,s1).
trans(s2,b,s3).
trans(s3,b,s4).

silent(s2,s4).
silent(s3,s1).
```

```
% accept(StartingState, InputCharacters)
accept(State, []) :-
        final(State).

accept(State, [X|Rest]) :-
        trans(State, X, State1),
        accept(State1, Rest).

accept(State, L) :-
        silent(State, State1),
        accept(State1, L).

?- L=[_,_,_], accept(s1, L).
L=[a,a,b]; L=[b,a,b]
```

Bird ontology

- Hierarchy of concepts
- Inheritance

See `birds.pl`

Object-oriented programming

Python: `object.attribute = value`

Prolog: `Object(Attribute, Value).`

```
bird(kind_of, animal).
bird(locomotion, flight).
bird(active_period, day).
bird(food, grain).

albatross(kind_of, bird).
albatross(colour,
          black_and_white).
albatross(size, 115).
albatross(food, fish).
```

```
kiwi(kind_of, bird).
kiwi(colour, marron).
kiwi(active_period, night).
kiwi(locomotion, walk).
kiwi(size, 40).

albert(instance_of,
          albatross).
albert(size, 120).
willy(instance_of, kiwi).
```

Object-oriented programming

```
% try to find the attribute within the existing object
value(Object, Attribute, Value) :-
        Request =.. [Object, Attribute, Value],
        Request,
        !.

% otherwise look for its parent ("super" in Python)
value(Object, Attribute, Value) :-
        parent(Object, Parent),
        value(Parent, Attribute, Value).

parent(Object, Parent) :-
        Request =.. [Object, kind_of, Parent],
        Request.
```

Boeing

```
(1.1) "An object is thrown with a horizontal speed of 20 m/s
      from a cliff that is 125 m high."
isa(object01,object_n1),
isa(speed01,velocity_n1),
isa(horizontal01,horizontal_a1),
isa(cliff01,cliff_n1),
isa(height01,height_n1),
isa(throw01,throw_v1),
height(cliff01,height01),
value(speed01,[20,m/s_n1]),
mod(speed01,horizontal01),
value(height01,[125,m_n1]),
object(throw01,object01),
"with"(throw01,speed01),
origin(throw01,cliff01).
```

Boeing

```
(1.2) "The object falls for the height of the cliff."
isa(fall01,fall_v1),
height(cliff01,height01),
agent(fall01,object01),
distance(fall01,height01).
```

```
(1.4) "What is the duration of the fall?"
isa(fall01,fall_v1),
isa(duration01,duration_n1),
duration(fall01,duration01),
query-for(duration01).
```

Expert systems

Advantages

- Queries and dialogs
- Explicit knowledge: auditable and editable
- Explainability
- Interface with ontologies and external knowledge

Drawbacks

- Eliciting knowledge from experts: the *knowledge bottleneck*
- No learning

Some ontologies

- Ontologies and knowledge bases were created as building blocks for AI
- Some examples:
  - WordNet
  - FrameNet
  - BabelNet
  - Mikrokosmos
  - OWL
- Two industries with large ontologies and expert systems: pharmaceutical and food industry

Mikrokosmos