Esercitazione/ approfondimento Prolog e Sistemi Esperti

- Presi da: I. Bratko: Programmare in Prolog per lIntelligenza Artificiale, Masson ed Addison-Weslay, 1988.
- **Base di conosenza per identificare gli animali (problema di classificazione)**

```
:- op(100,xfx,[has,gives,'does not', eats,lays,isa]).
:- op(100,xf,[swims,flies]).
:- op(900,xfx,:).
:- op(800,xfx,was).
:- op(870,fx,if).
:- op(880,xfx,then).
:- op(550,xfy,or).
:- op(540,xfy,and).
:- op(300,fx,'derived by').
:- op(600,xfx,from).
:- op(600,xfx,by).
```

ALTRI ESEMPI

```
rule1:  if
    Animal has hair      or
    Animal gives milk    then
    Animal isa mammal.
rule2:  if
    Animal has feathers or
    Animal flies         and
    Animal lays eggs     then
    Animal isa bird.
rule3:  if
    Animal isa mammal              and
    (Animal eats meat              or
    Animal has pointed teeth       and
    Animal has claws               and
    Animal has 'forward pointing 'eyes') then
    Animal isa carnivore.
```

ALTRI ESEMPI

```
rule4:  if
    Animal isa carnivore          and
    Animal has 'tawny colour'     and
    Animal has 'dark spots'       then
    Animal isa cheetach.
rule5:  if
    Animal isa carnivore          and
    Animal has 'tawny colour'     and
    Animal has 'black stripes'    then
    Animal isa tiger.
rule6:  if
    Animal isa bird               and
    Animal 'does not' fly         and
    Animal swims                  then
    Animal isa pinguin.
```

ALTRI ESEMPI

```
rule7:  if
    Animal isa bird                and
    Animal isa 'good flyer'                then
    Animal isa albatross.
fact: X isa animal:-
member(X,[cheetah,tiger,penguin,albatross]).
askable(_ gives_, 'Animal' gives 'What').
askable(_ flies, 'Animal' flies).
askable(_ lays eggs, 'Animal' lays eggs).
askable(_ eats_, 'Animal' eats 'What').
askable(_ has_, 'Animal' has 'Something').
askable(_ 'does not'_, 'Animal' 'does not' 'Do something').
askable(_ swims, 'Animal' swims).
askable(_ isa 'good flier', 'Animal' isa 'good flier').
```

Nota: sono tutti fatti Prolog dal punto di vista sintattico.

UN SECONDO ESEMPIO

Base di conoscenza per individuare guasti in una rete elettrica (diagnosi).

```
broken_rule: if
    on(Device)                  and
    device(Device)              and
    not working(Device)         and
    connected(Device,Fuse)      and
proved (intact(Fuse))
then
proved(broken(Device)).
fuse_ok_rule: if
    connected(Device,Fuse)      and
    working(Device)
then
proved(intact(Fuse)).
```

UN SECONDO ESEMPIO

```
fused_rule:
if connected(Device1,Fuse)          and
    on(Device1)                     and
    device(Device1)                 and
    not working(Device1)            and
    samefuse(Device2,Device1)       and
    on(Device2)                     and
    not working(Device2)
then
proved(failed(Fuse)).
```

se due differenti dispositivi sono connessi ad un fusibile e sono entrambi on ma non in funzione allora il fusibile è rotto (si assume che non possano essere rotti entrambi i dispositivi).

UN SECONDO ESEMPIO

```
same_fuse_rule: if
    connected(Device1,Fuse) and
    connected(Device2,Fuse) and
    different(Device1,Device2)
then
somefuse(Device1,Device2).
fact: different(X,Y) :- not (X=Y).
fact: device(heater).
fact: device(light1).
fact: device(light2).
fact: device(light3).
fact: device(light4).
fact: connected(light1,fuse1).
fact: connected(light2,fuse1).
fact: connected(heater,fuse1).
```

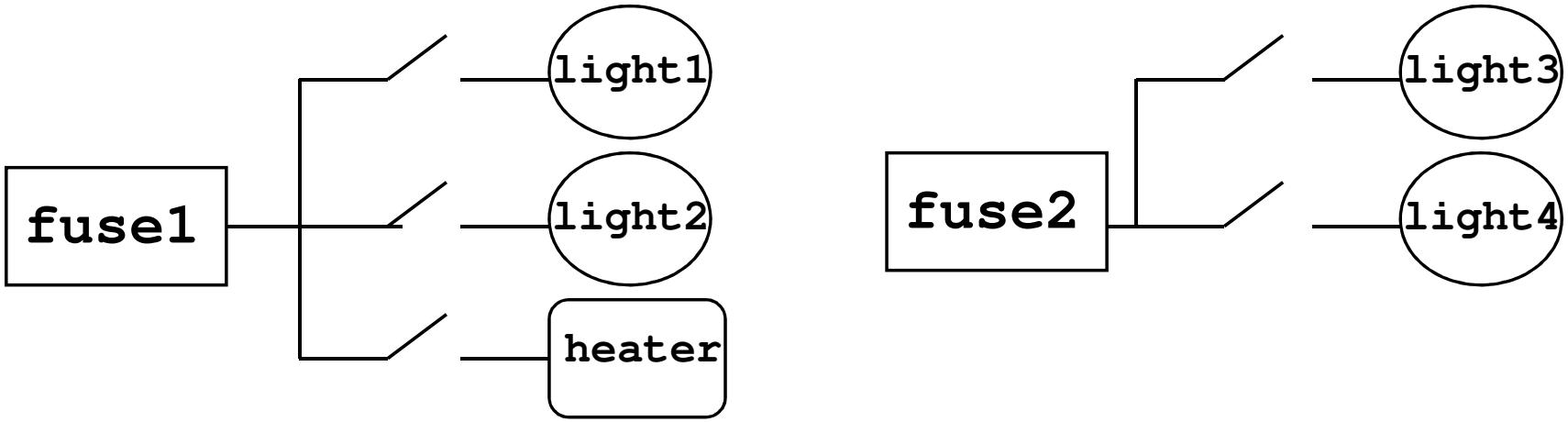
UN SECONDO ESEMPIO

```
fact: connected(light3,fuse2).
fact: connected(light4,fuse2).

askable(on(D), on('Device')).

askable(working(D),working('Device')).
```

- *Nota: sono tutti fatti Prolog.*



USO DEL PROLOG

- Per utilizzare direttamente Prolog andrebbero traslati come regole del tipo:

```
Animal isa mammal:-
          Animal has hair;
          Animal gives milk.
Animal isa carnivore:-
          Animal isa mammal,
          Animal eats meat.
```

- Adesso aggiungiamo dei fatti sul particolare problema:

```
peter has hair.
peter is lazy.
peter is big.
```

USO DEL PROLOG

```
peter has 'tawny colour'.
peter has 'black stripes'.
peter eats meat.
```

- e poi interroghiamo:

```
?- peter isa tiger.
yes
? peter isa cheetah.
no.
```

- Non ci va bene per due motivi:

  1) I fatti devono essere introdotti tutti all'inizio;

  2) Manca una spiegazione (il tracing di Prolog è troppo povero).

  ==> approccio meta-interpretato

MOTORE DI INFERENZA IN PROLOG

- Per dare una risposta Answ a una domanda Q (simile alla ricerca in grafi AND/OR):

  Se **Q** è un fatto allora Answ è: **Q is true**;

  Se c'è una regola del tipo:

  if Condition then Q allora esplora **Condition** per generare la risposta Answ;

  Se **Q** è askable allora chiedi all'utente per avere una risposta per **Q**;

  Se **Q** è della forma **Q1 and Q2** allora esplora **Q1**. Se **Q1** è falso allora Answ è "**Q** e falso", altrimenti esplora **Q2** e poi combina le risposte di **Q1** e **Q2** in Answ;

  Se **Q** è della forma **Q1 or Q2** allora esplora **Q1**. Se **Q1** è vero allora Answ è "**Q** è vero", o alternativamente esplora **Q2** e poi combina le risposte di **Q1** e **Q2** in Answ.

MOTORE DI INFERENZA IN PROLOG

- Le domande del tipo **`not Q`** sono più problematiche e le tratteremo nel seguito.

  Interfaccia con l'utente: why e how.

- La domanda **why** può essere generata dall'utente quando il sistema chiede all'utente qualche informazione e l'utente vuole sapere perchè gli viene chiesta tale informazione.

```
Is a true?

why?
```

- Because:

```
I can use a to investigate b by rule Ra, and
I can use b to investigate c by rule Rb, and ........
I can use y to investigate z by rule Ry, and
z was your original question.
```

- Catena di regole e (sotto)goals che connettono l'informazione richiesta con il goal originale (traccia). Why è ottenuto muovendosi in su nello spazio di ricerca dal corrente (sotto)goal al top goal.

MOTORE DI INFERENZA IN PROLOG

- Dunque la traccia (catena dei goals e regole fra il goal corrente e il top goal) deve essere mantenuta esplicitamente durante il processo di ragionamento.
- Inoltre, quando l'utente ottiene una risposta può avere interesse a sapere come questa risposta è stata ottenuta.
- How fa vedere i goal e sottogoal che dimostrano la conclusione, cioè, in pratica, l'albero AND/OR di soluzione.
- Esempio:

```
peter isa carnivore
was derived by rule3 from
peter isa mammal
was derived by rule1 from
peter has hair
was told
and
peter eats meat
was told
```

IMPLEMENTAZIONE

- Procedure principali:

  `explore(Goal, Trace, Answer)`

  che trova una risposta **`Answer`** a un goal **`Goal`** con la traccia **`Trace`**.

  `useranswer(Goal,Trace,Answer)`

  che genera la soluzione per un “askable” **`Goal`** chiedendola all'utente e risponde anche a domande di tipo 'why'.

  `present(Answer)`

  mostra i risultati e risponde a domande di tipo 'how'.

  `explore(Goal,Trace,Answer)`

  “trova una risposta **`Answer`** a un dato goal **`Goal`**. Cerca una soluzione positiva. **`Answer`** falso solo quando sono state tentate con insuccesso tutte le possibilità. Nota: si suppone che ci sia solo una regola applicabile per ogni tipo di goal; **`Goal`** negativi devono sempre essere istanziati”

IMPLEMENTAZIONE

```
:- op(900,xfx,:).

:- op(800,xfx,was).

:- op(870,fx,if).

:- op(880,xfx,then).

:- op(550,xfy,or).

:- op(540,xfy,and).

:- op(300,fx,'derived by').

:- op(600,xfx,from).

:- op(600,xfx,by).

explore(Goal,Trace,Goal is true was 'found

as a fact'):- fact : Goal.
```

IMPLEMENTAZIONE

```
explore(Goal,Trace,Goal is true was 'found  as a fact'):-
        fact : Goal.
explore(Goal,Trace,Goal is TruthValue was 'derived by' Rule from
   Answer):-
        Rule : if Condition then Goal,
        explore(Condition,[Goal by Rule|Trace],Answer),
        truth(Answer, TruthValue).
explore(not Goal, trace, Answer) :- !,
        explore(Goal,Trace,Answer1),
        invert(Answer1,Answer).
explore(Goal1 and Goal2, Trace, Answer):-!,
        explore(Goal1, Trace,Answer1),
        continue(Answer1, Goal1 and Goal2, Trace, Answer).
```

IMPLEMENTAZIONE

```
explore(Goal1 and Goal2, Trace, Answer):-!,
        explore(Goal1, Trace,Answer1),
        continue(Answer1, Goal1 and Goal2, Trace, Answer).
explore(Goal1 or Goal2, Trace, Answer):-
        exploreyes(Goal1, Trace,Answer);
        exploreyes(Goal2, Trace,Answer).
explore(Goal1 or Goal2, Trace, Answer1 and Answer2):-!,
        not exploreyes(Goal1, Trace,_);
        not exploreyes(Goal2, Trace,_),
        explore(Goal1, Trace,Answer1);
        explore(Goal2, Trace,Answer2).
explore(Goal,Trace,Goal is Answer was told):-
        useranswer(Goal,Trace,answer).
```

IMPLEMENTAZIONE

```
exploreyes(Goal,Trace,Answer):-
        explore(Goal,Trace,Answer),
        positive(Answer).
continue(Answer1, Goal1 and Goal2, Trace, Answer):-
        positive(Answer1),
        explore(Goal2,Trace,Answer2),
        (positive(Answer2),
         Answer=Answer1 and Answer2;
         negative(Answer2), Answer=Answer2).
continue(Answer1, Goal1 and Goal2, _, Answer1):-
        negative(Answer1).
truth(Question is TruthValue was Found, TruthValue) :- !.
```

IMPLEMENTAZIONE

```
truth(Answer1 and Answer2, TruthValue):-
        truth(Answer1,true),
        truth(Answer2,true),!,
        TruthValue = true;
        TruthValue = false.
positive(Answer) :-
        truth(Answer,true).
negative(Answer) :-
        truth(Answer,false).
        invert(Goal is true was Found,
                (not Goal) is false was Found).
invert(Goal is false was Found,
                (not Goal) is true was Found).
```

ESERCITAZIONE PROPOSTA: Terminare l'intero shell

- Definire le procedure:

`useranswer(Goal,Trace,Answer)`

- Tenendo conto del fatto che:
  - deve fare la domanda una sola volta per lo stesso goal controllando che sia askable;
  - fare il trace se richiesto dall'utente mediante why;
  - se goal contiene delle variabili farle istanziare dall'utente.

`present(Answer)`

per mostrare la soluzione e eventualmente la spiegazione how

`expert`

Il goal top-level dello shell che chiede il goal e poi mostra la risposta;