Prolog III

Predicati dinaminci

- dynamic(p/*intero*)
- Sono predicati la cui definizione è soggetta a cambiamento
  - Contingenti
  - Alcuni prolog richiedono che ogni predicato usato sia definito da almeno una clausola o sia dinamico
  - I predicati dinamici possono essere dinamicamente ridefiniti con assert e retract

Assert e retract

- Vi sono istruzioni per operare su basi dati, ovvero su insiemi di fatti e regole che possono essere modificati dinamicamente
  - Simili a updates in basi dati
- assert(C) aggiunge una clausola alla base dati
- asserta(C) la aggiunge all'inizio
- assertz(C) la aggiunge alla fine
- retract(C) toglie la prima della base dati che unifica con C
- retractall(C) le toglie tutte (se disponibile)

ESEMPIO: generazione di indici nuovi

```
:- dynamic(ind/1).

newInd(I) :- retract(ind(J)), !,
             I is J+1,
             assert(ind(I)).
newInd(0) :- assert(ind(0)).
```

Studiate il libro a pag. 486, 487; è interessante;
per capire alcune parti ricordare le difference lists L-M.

Uso di not

- La negazione è spesso utile nel corpo di una clausola
- I programmi con not nel corpo si dicono normali
- Si hanno problemi dovuti a
  - Invalidità per floundering
  - Assenza di modello minimo

Floundering

- Si ha quando viene selezionato un goal negativo non ground

```
vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
```

vuoto(X)? A mano, si ha: X = cestino

contiene(cestino,X) fail
not contiene(cestino,X)
vuoto(cestino)

Floundering

Ma prolog implementa not come illustrato sotto:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

vuoto(X)

Floundering

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

not(contiene(X,Y))
vuoto(X)

Floundering

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

contiene(X,Y) ! fail
not(contiene(X,Y))
vuoto(X)

Floundering

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

contiene(cesto,cestino) **!** fail
not contiene(cesto,cestino)
vuoto(cesto)

Floundering

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

Cut taglia la ricerca
Tutto fallisce e risposta NO

contiene(cesto,cestino) ! **fail**
not contiene(cesto,cestino)
vuoto(cesto)

Floundering

Però:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

vuoto(cestino)

Floundering

Ma prolog:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

not(contiene(cestino,Y))
vuoto(cestino)

Floundering

Ma prolog:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

contiene(cestino,Y) ! fail
not contiene(cestino,Y)
vuoto(cestino)

Floundering

Ma prolog:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

backtrack

contiene(cestino,Y) fail ! fail
not(contiene(cestino,Y))
vuoto(cestino)

Floundering

Ma prolog:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

not(contiene(cestino,Y))
vuoto(cestino)

Floundering

Ma prolog:

vuoto(X) :- not contiene(X,Y).
contiene(cesto, cestino).
not(X) :- X, !, fail.
not(X).

Risposta SI

not(contiene(cestino,Y))
vuoto(cestino)

Può non esistere modello minimo

ESEMPIO

bello(inpiazza) ← not(bello(infascia)).
bello(infascia) ← not(bello(inpiazza)).

Logicamente equivalente a
bello(inpiazza) ∨ bello(infascia)

bello(inpiazza) ∨ bello(infascia)

Due modelli minimali

{bello(inpiazza)}
{bello(infascia)}

Ma il programma Prolog non termina.
Inoltre:
Siccome non esiste modello minimo, non possiamo più interpretare il fallimento come falsità nel modello minimo.

Ricetta per usare consistentemente la negazione

- Quando not(A) viene selezionata, A deve essere ground
  - Serve per evitare floundering
  - Questa condizione potrebbe essere indebolita
- Per ogni goal ground il programma deve terminare
  - Garantisce che l'assunzione del mondo chiuso è consistente con la negazione come fallimento; cioè, assumendo per falsi gli atomi che falliscono, si ha modello minimo
  - la condizione di terminazione potrebbe essere indebolita

Metaprogrammazione Cenno

- Si hanno primitive per vedere come è costruito un atomo, per costruire atomi e clausole, ecc.

Alcune primitive di metaprogrammazione:

atom(X)
integer(X)
float(X)
ground(X)
call(X)

X =.. Y
? f(T1,..,Tn) =..X
X = [F,T1,..,Tn]

? f(T1,..,Tn) =..[F|A]
F = f
A = [T1,..,Tn]