

# PROLOG

## Sintassi di *Prolog*<sub>0</sub>

Dato un insieme di atomi  $A$  e variabili  $V$ :


1.  $A$  è formata da tutte le parole che iniziano per una lettera minuscola e proseguono (eventualmente) con lettere, numeri e underscore.
2.  $V$  è formata da tutte le parole che iniziano con una lettera maiuscola e proseguono (eventualmente) con lettere, numeri e underscore.

*Nota:*  $A \neq \emptyset$ ,  $V \neq \emptyset$ ,  $A \cap V = \emptyset$  ed entrambi gli insiemi sono infiniti numerabili.

Un programma Prolog è un insieme di clausole, dove ogni clausola può essere un fatto o una regola.

Dato un programma in *Prolog*<sub>0</sub> è possibile ottenere una computazione abbinando al programma un goal. Una computazione così ottenuta, se produce un risultato, una o più sostituzioni che coinvolge un sottoinsieme delle variabili del goal.

Un elemento dell'insieme delle clausole definite unito all'insieme dei goal viene detto **clausola di Horn**. → le clausole di Horn sono tutte quelle clausole che ci permettono di esprimere un goal; tutte le clausole anche quelle che non hanno testa.

*Nota:* un goal è una regola privata dalla testa, quindi se voglio scrivere un goal devo iniziare con 

Con  $Prolog_0^+$  aggiungiamo la possibilità di usare liste ed operatori. Permette di aggiungere un elemento formato da un underscore. Questa variabile viene detta muta (o dummy), essa prevede che venga sostituita con una nuova variabile non presente nel programma o nel goal tutte le volte che viene incontrata.

---

## Il linguaggio $Prolog_!$

$Prolog_0^+$  e  $Prolog_0$  sono sufficienti per risolvere problemi interessanti.  $Prolog_0^+$  però non offre alcun modo per bloccare le scelte non deterministiche.

Di conseguenza non c'è modo di bloccare alcuni percorsi.

Questo problema potrebbe essere risolto con un ipotetico operatore di negazione.

Però la negazione è uno strumento complesso a causa della presenza di **variabili libere**.

Il linguaggio  $Prolog_!$  è come  $Prolog_0$  arricchito da un atomo. Questo operatore viene indicato con `!` (cut).

Quando viene incontrato un cut vengono scartate:

1. Tutte le scelte non deterministiche che precedono il cut. (scelte prima del cut FISSATE)
2. Tutte le scelte non deterministiche per cercare un fatto o una regola con testa. (appena trovi una soluzione non cercare altra roba)

Queste scelte vengono scartate anche se la regola in cui è presente il cut fallisce.

- **Green cut**: non vengono scartate possibili soluzioni (solo soluzioni che ho già).
- **Red cut**: vengono scartate possibili soluzioni (percorsi che non ci interessano).
- `true` → goal che non falliscono.
- `fail` → goal che falliscono sempre.
- `\=` → indica che i due termini sono diversi tra loro solo se non possono essere unificati.

La negazione offerta da *Prolog* viene detta **negazione per fallimento** e viene realizzata soddisfacendo un goal negato solo quando il relativo goal affermato non è soddisfacibile.

Esempio per il predicato `nonmember`:

```
nonmember(_, []).
nonmember(X, [X | _]) :-
    !,
    fail.
nonmember(X, [_ | R]) :-
    nonmember(X, R).

% TODO Ricerca definizione di nonmember/ \=
nonmember(Y, [X | R]).
    Y \= X,
    nonmember(Y, R).
```