Il controllo del backtracking

```
sano(X) :- felice(X).
sano(X) :- ha_anticorpi(X).

felice(X) :- canta(X), balla(X).
felice(X) :- stupido(X).

ha_anticorpi(maria).
balla(linda).
canta(linda).
canta(gianni).
stupido(gianni).

?- sano(X).
X = linda ;
X = gianni ;
X = maria.
```

Il controllo del backtracking: il “cut”

```
sano(X) :- felice(X).
sano(X) :- ha_anticorpi(X).

felice(X) :- canta(X), ! , balla(X).
felice(X) :- stupido(X).

ha_anticorpi(maria).
balla(linda).
canta(linda).
canta(gianni).
stupido(gianni).
```

Il controllo del backtracking: il “cut”

```
sano(X) :- felice(X).
sano(X) :- ha_anticorpi(X).

felice(X) :- canta(X), ! , balla(X).
felice(X) :- stupido(X).

ha_anticorpi(maria).
balla(linda).
canta(linda).
canta(gianni).
stupido(gianni).

?- sano(X).
X = linda ;
X = maria.
```

Non viene più fornita la soluzione `X = gianni`.

Il “cut”

- Il cut è un predicato, senza argomenti, che consente di controllare il backtracking
- È un goal che ha sempre successo (dal punto di vista dichiarativo, equivale a **true**)

Il “cut”

- Il cut è un predicato, senza argomenti, che consente di controllare il backtracking
- È un goal che ha sempre successo (dal punto di vista dichiarativo, equivale a **true**)
- Vincola il Prolog alle scelte fatte da quando è stato chiamato il **parent goal** (il goal che ha utilizzato la clausola contenente il cut).

  Taglia le alternative lasciate aperte per il backtracking da quando il parent goal è stato unificato con la testa della regola utilizzata (inclusa la scelta di utilizzare quella clausola).

Il "cut"

- Il cut è un predicato, senza argomenti, che consente di controllare il backtracking
- È un goal che ha sempre successo (dal punto di vista dichiarativo, equivale a **true**)
- Vincola il Prolog alle scelte fatte da quando è stato chiamato il **parent goal** (il goal che ha utilizzato la clausola contenente il cut).

  Taglia le alternative lasciate aperte per il backtracking da quando il parent goal è stato unificato con la testa della regola utilizzata (inclusa la scelta di utilizzare quella clausola).
- Il cut può essere utilizzato per
  - evitare di perdere tempo ad esplorare strade "inutili"
  - evitare di esplorare strade che porterebbero a soluzioni non corrette

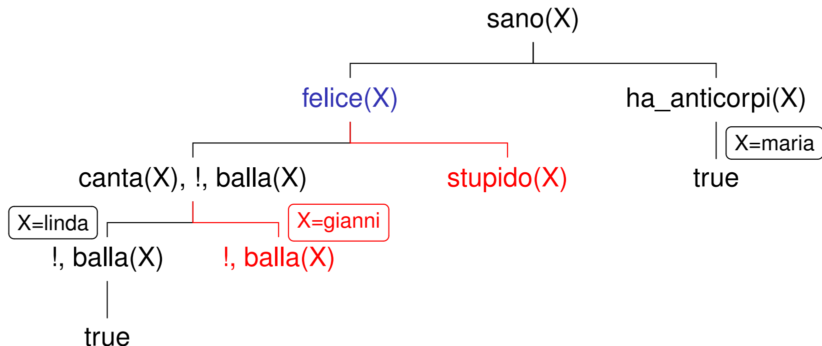
Il "cut"

- Il cut è un predicato, senza argomenti, che consente di controllare il backtracking
- È un goal che ha sempre successo (dal punto di vista dichiarativo, equivale a **true**)
- Vincola il Prolog alle scelte fatte da quando è stato chiamato il **parent goal** (il goal che ha utilizzato la clausola contenente il cut).

  Taglia le alternative lasciate aperte per il backtracking da quando il parent goal è stato unificato con la testa della regola utilizzata (inclusa la scelta di utilizzare quella clausola).
- Il cut può essere utilizzato per
  - evitare di perdere tempo ad esplorare strade "inutili"
  - evitare di esplorare strade che porterebbero a soluzioni non corrette
- Il cut può "rovinare" la reversibilità dei programmi.

Perché Gianni non è sano

```
sano(X) :- felice(X).
sano(X) :- ha_anticorpi(X).
felice(X) :- canta(X), !, balla(X).
felice(X) :- stupido(X).
ha_anticorpi(maria).   balla(linda).
canta(linda).          canta(gianni).    stupido(gianni).
```



felice(X) è il parent goal.

Cut verdi e cut rossi

- **Cut verde:** influenza l'efficienza ma non cambia il significato del programma (con o senza cut, le soluzioni sono le stesse)

```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y) :- X < Y.
```

- **Cut rosso:** cambia il significato del programma

```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y).
```

Il cut e la logica

Prolog	Logica
p :- a, b. p :- c.	$p \leftrightarrow (a \wedge b) \vee c$
p :- a, !, b. p :- c.	$p \leftrightarrow (a \wedge b) \vee (NOT\ a \wedge c)$
p :- c. p :- a, !, b.	$p \leftrightarrow (a \wedge b) \vee c$

Il cut e la logica

```
p :- a, b.
p :- c.
```

$p \leftrightarrow (a \wedge b) \vee c$

```
p :- a, !, b.
p :- c.
```

$p \leftrightarrow (a \wedge b) \vee (NOT\ a \wedge c)$

```
p :- c.
p :- a, !, b.
```

$p \leftrightarrow (a \wedge b) \vee c$

Ma attenzione: NOT non è la negazione logica.

NOT a = il goal a non è dimostrabile dal programma

La combinazione cut & fail

fail: goal che fallisce immediatamente (inutile? Ma quando il Prolog fallisce, tenta il backtracking...)

Tutti i francesi, eccetto i parigini, sono gentili.

```
gentile(X) :- parigino(X), !, fail.
gentile(X) :- francese(X).
```

```
francese(antoine).
francese(charles).
parigino(antoine).
```

```
?- gentile(antoine).
false.
```

```
?- gentile(charles).
true.
```

La combinazione cut & fail

fail: goal che fallisce immediatamente (inutile? Ma quando il Prolog fallisce, tenta il backtracking...)

Tutti i francesi, eccetto i parigini, sono gentili.

```
gentile(X) :- parigino(X), !, fail.
gentile(X) :- francese(X).
```

```
francese(antoine).
francese(charles).
parigino(antoine).
```

```
?- gentile(antoine).
false.
```

```
?- gentile(charles).
true.
```

Ma attenzione:

```
?- gentile(X).
false.
```

La negazione come fallimento

La combinazione cut & fail sembra fornire qualche forma di negazione: la **negazione come fallimento**, che è definita così:

```
not(Goal) :- Goal, !, fail.
not(Goal).
```

Usando il **not**, possiamo modificare la definizione di gentile:

```
gentile(X) :- francese(X), not(parigino(X)).
```

```
?- gentile(X).
X = charles.
```

La negazione come fallimento si comporta come la negazione logica, a patto che, nel momento in cui il goal **not(G)** viene invocato, tutte le variabili in **G** siano legate.

```
gentile(X) :- not(parigino(X)), francese(X).
```

```
?- gentile(X).
false.
```

Metaprogrammazione

Programmi e dati sono rappresentati in modo uniforme

- Un atomo è un termine: la struttura di un atomo è uguale a quella di un termine (una struttura)
- Una congiunzione di atomi è un termine: la virgola è un funtore
- Una clausola è un termine: `:-` è un funtore
- Un fatto `A` è una clausola della forma `A :- true`.

I programmi possono essere trattati come dati

Meta-programma: programma che tratta altri programmi come dati

Metaprogrammazione

Programmi e dati sono rappresentati in modo uniforme

- Un atomo è un termine: la struttura di un atomo è uguale a quella di un termine (una struttura)
- Una congiunzione di atomi è un termine: la virgola è un funtore
- Una clausola è un termine: `:-` è un funtore
- Un fatto `A` è una clausola della forma `A :- true`.

I programmi possono essere trattati come dati

Meta-programma: programma che tratta altri programmi come dati

In Prolog si possono

- **leggere** le clausole del programma
- **aggiungere** e **eliminare** clausole dal programma: un programma Prolog può automodificarsi