

Linguaggi di Programmazione

Roberta Gori

Systemi Logici 2.2

Regole di Inferenza

Regole di inferenza

premesse (una, nessuna, molte)

$$\frac{x_1 \quad \dots \quad x_n}{y}$$

se le premesse sono valide,
allora anche la conclusione è
valida

conclusioni (una)

x_1, \dots, x_n, y sono formule

ogni variabile è (implicitamente) quantificata universalmente

l'istanza di una regola è ottenuta applicando una sostituzione ρ a x_1, \dots, x_n, y

Assiomi

senza premesse

$$\frac{}{y}$$

la conclusione e' valida

conclusione (sempre valida, e' un fatto)

Istanza di una regola

$$(prod) \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n} \quad n = n_0 \cdot n_1$$

un istanza di prod

$$(prod) \frac{1 \longrightarrow 1 \quad 1 \oplus 2 \longrightarrow 3}{1 \otimes (1 \oplus 2) \longrightarrow 3} \quad 3 = 1 \cdot 3$$

c'è anche un'altra istanza di *(prod)*!

$$(prod) \frac{1 \longrightarrow 3 \quad 1 \oplus 2 \longrightarrow 5}{1 \otimes (1 \oplus 2) \longrightarrow 15} \quad 15 = 3 \cdot 5$$

ma sarà difficile che le premesse siano valide

Altre istanze

$$(prod) \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n} \quad n = n_0 \cdot n_1$$

un istanza di
(prod)

$$(prod) \frac{E \otimes 2 \longrightarrow k \quad E \oplus 1 \longrightarrow 3}{(E \otimes 2) \otimes (E \oplus 1) \longrightarrow 3k}$$

possiamo avere la
stessa variabile

Sistema Logico

$$R = \left\{ \frac{\quad}{z}, \frac{x_1 \cdots x_n}{y}, \dots \right\}$$

Un **sistema logico** e' un insieme assiomi e regole di inferenza

se una regola di inferenza contiene alcune variabili, assumiamo che tutte le sue istanze siano in R

Una derivazione

Dato un sistema logico R , una derivazione in R , si scrive

$$d \Vdash_R y$$

dove

- o $d = \left(\frac{}{y}\right)$ è un assioma di R ;
- oppure $d = \left(\frac{d_1, \dots, d_n}{y}\right)$ ed esistono derivazioni $d_1 \Vdash_R x_1, \dots, d_n \Vdash_R x_n$ e $\left(\frac{x_1, \dots, x_n}{y}\right)$ è una regola di inferenza di R .

una derivazione e' un albero di prova (le cui foglie sono assiomi)

Esempio

$$R = \left\{ \frac{}{N \longrightarrow n}, \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \oplus E_1 \longrightarrow n_0 + n_1}, \frac{E_0 \longrightarrow n_0 \quad E_1 \longrightarrow n_1}{E_0 \otimes E_1 \longrightarrow n_0 \cdot n_1} \right\}$$

$$d = \frac{\frac{1 \longrightarrow 1 \quad 2 \longrightarrow 2}{(1 \oplus 2) \longrightarrow 3} \quad \frac{3 \longrightarrow 3 \quad 4 \longrightarrow 4}{(3 \oplus 4) \longrightarrow 7}}{(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21}$$

$$d \Vdash_R (1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

Teoremi

Dato un sistema logico R , un **teorema di R** e' scritto nel seguente modo:

$$\Vdash_R y$$

$$\exists d. d \Vdash_R y$$

dove y e' una formula tale che possiamo trovare una derivazione per y in R

L'insieme di tutti i teoremi di R è denotato con I_R .

$$I_R \triangleq \{ y \mid \Vdash_R y \}$$

Notazione inline

$$d = \frac{\frac{\overline{1 \longrightarrow 1} \quad \overline{2 \longrightarrow 2}}{(1 \oplus 2) \longrightarrow 3} \quad \frac{\overline{3 \longrightarrow 3} \quad \overline{4 \longrightarrow 4}}{(3 \oplus 4) \longrightarrow 7}}{(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21}$$

$$(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

$$\nearrow (1 \oplus 2) \longrightarrow 3, (3 \oplus 4) \longrightarrow 7$$

$$\nearrow 1 \longrightarrow 1, 2 \longrightarrow 2, (3 \oplus 4) \longrightarrow 7$$

$$\nearrow 2 \longrightarrow 2, (3 \oplus 4) \longrightarrow 7$$

$$\nearrow (3 \oplus 4) \longrightarrow 7$$

$$\nearrow 3 \longrightarrow 3, 4 \longrightarrow 4$$

$$\nearrow 4 \longrightarrow 4$$

niente da provare

\square

derivazione
goal-oriented

Backtracking

$$(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21$$

$$\swarrow (1 \oplus 2) \longrightarrow 7, (3 \oplus 4) \longrightarrow 3$$

$$\swarrow 1 \longrightarrow 1, 2 \longrightarrow 6, (3 \oplus 4) \longrightarrow 3$$

$$\swarrow 2 \longrightarrow 6, (3 \oplus 4) \longrightarrow 3$$

fallimento! dobbiamo tornare indietro (backtrack) fino all'ultima scelta e riprovare

$$\swarrow 1 \longrightarrow 2, 2 \longrightarrow 5, (3 \oplus 4) \longrightarrow 3$$

fallimento! dobbiamo tornare indietro (backtrack) fino all'ultima scelta e riprovare

...

alternativamente possiamo esplorare tutte le alternative in parallelo
(breadth-first vs depth-first)

derivazione
goal-oriented
(depth-first)

Programmazione Logica

PROLOG

Prolog is a simple, yet powerful declarative programming language, based on first-order predicate logic

PROgrammation en LOGique

[’70] (Univ. Marseilles) A. Colmerauer, P. Roussel, R. Kowalski
(aimed at processing natural (French) language)

```
Every psychiatrist is a person.  
Every person he analyzes is sick.  
Jacques is a psychiatrist in Marseille.  
Is Jacques a person?  
Where is Jacques?  
Is Jacques sick?
```

```
Yes. In Marseille.  
I don't know.
```

```
TOUT PSYCHIATRE EST UNE PERSONNE.  
CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.  
JACQUES EST UN PSYCHIATRE A *MARSEILLE.  
EST-CE QUE *JACQUES EST UNE PERSONNE?  
OU EST *JACQUES?  
EST-CE QUE *JACQUES EST MALADE?  
OUI. A MARSEILLE. JE NE SAIS PAS.
```

Algoritmo

algoritmo = logica + controllo

cosa

(descrizione del problema)

clausole di Horn

database
PROLOG

come

(passi per arrivare alla
soluzione)

risoluzione

interprete
PROLOG

Formule

$X = \{x, y, \dots\}$ un insieme di variabili

$\Sigma = \{\Sigma_n\}_n$ una segnatura di simboli di funzioni c, f, g, \dots

$\Pi = \{\Pi_n\}_n$ una segnatura di simboli di predicato p, q, \dots

formula atomica

$$a = p(t_1, \dots, t_n) \quad \begin{array}{l} p \in \Pi_n \\ t_1, \dots, t_n \in T_{\Sigma, X} \end{array}$$

formula

a_1, \dots, a_n una congiunzione (anche vuota) di formule atomiche

Programmi Logici

Clausole di Horn

una formula
atomica
(la TESTA)

$h :- r$

una formula
(il CORPO)

$a :- a_1, \dots, a_n$

simile a

$\frac{a_1 \cdots a_n}{a}$

un insieme (o una lista) di clausole di Horn

Un programma
logico L

$L = \left\{ \begin{array}{c} \dots \\ h :- r. \\ \dots \end{array} \right\}$

Refutazione di un goal

Voglio dimostrare che la formula a_1, \dots, a_n è una conseguenza logica (puo' essere derivata dagli assiomi e dalle regole del sistema)

A questo scopo aggiungiamo alle regole la negazione della formula che vogliamo dimostrare

? $\neg a_1, \dots, a_n$ Goal

e cerchiamo di ottenere una contraddizione

Applicazioni

un programma logico serve a rispondere alla seguente domanda:

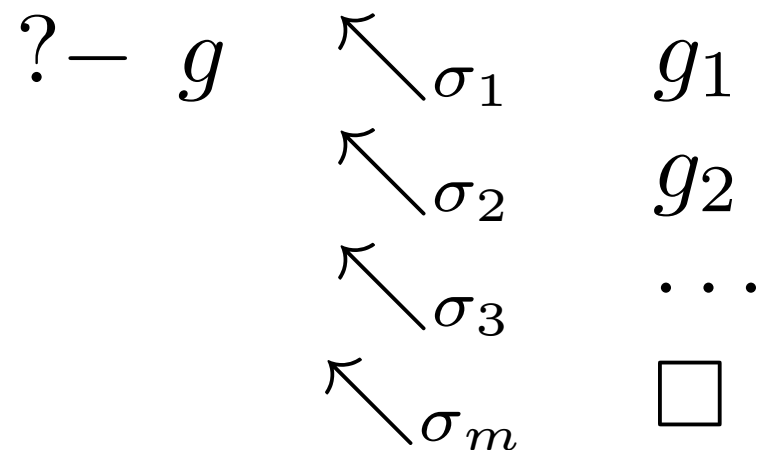
data una formula g che vogliamo dimostrare, quali sono le istanze di g valide?

$$? - (1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow n, \quad n \oplus E \longrightarrow 26$$

Risoluzione SLD

Idea: ridurre iterativamente il goal iniziale g applicando una delle clausole di Horn in L a una delle formule atomiche del goal;

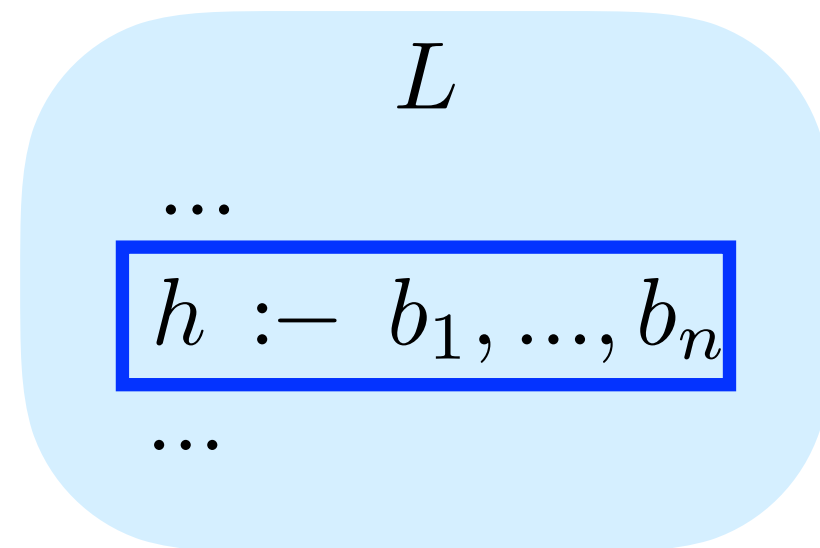
ogni applicazione calcola un unificatore più generale (mgu), sostituisce la formula selezionata con il corpo della clausola selezionata e applica il mgu al nuovo obiettivo



allora $g\sigma_1\sigma_2\dots\sigma_m$ è un teorema

Risoluzione SLD

?- $a_1, \dots, \boxed{a_i}, \dots, a_k$



Ripeti i passi seguenti finché ci sono formule atomiche nel goal

1. seleziona una formula atomica a_i nel goal (per esempio prendi la prima da sinistra);
2. seleziona una clausola Horn $h : -b_1, \dots, b_n \in L$ la cui testa unifica con a_i ;
3. sia σ l'unificatore più generale (mgu) tra h e a_i , $a_i\sigma = h\sigma$;
4. sostituisci a_i nel goal con il corpo della clausola b_1, \dots, b_n ;
5. applica la sostituzione σ a tutto il goal ottenuto: $(a_1, \dots, a_{i-1}, b_1, \dots, b_n, a_{i+1}, \dots, a_k)\sigma$.

Attenzione

le formule atomiche in un goal possono condividere variabili: la sostituzione deve essere applicata a tutto il goal per propagare l'informazione



$(a_1, \dots, b_1, \dots, b_n, \dots, a_k)\sigma$



$a_1, \dots, (b_1, \dots, b_n)\sigma, \dots, a_k$

Attenzione

la stessa clausola può essere riutilizzata molte volte:
ogni volta le sue variabili devono essere rinominate (prima dell'unificazione) con nuovi identificatori per evitare clashes

Ripeti i passi seguenti finché' ci sono formule atomiche nel goal

1. seleziona una formula atomica a_i nel goal;
2. seleziona una clausola Horn $h : -b_1, \dots, b_n \in L$;
3. sia ρ un renaming per le variabili in $vars(h : -b_1, \dots, b_n)$;
4. $(h : -b_1, \dots, b_n)\rho$ è chiamata *variante* dell'originale;
5. sia σ l' mgu tra $h\rho$ e a_i , $a_i\sigma = (h\rho)\sigma$;
6. sostituisci a_i con $(b_1, \dots, b_n)\rho$;
7. applica la sostituzione σ : $(a_1, \dots, a_{i-1}, (b_1, \dots, b_n)\rho, a_{i+1}, \dots, a_k)\sigma$.

Attenzione

nelle sostituzioni, solo le variabili che appaiono nel goal sono rilevanti per la risposta

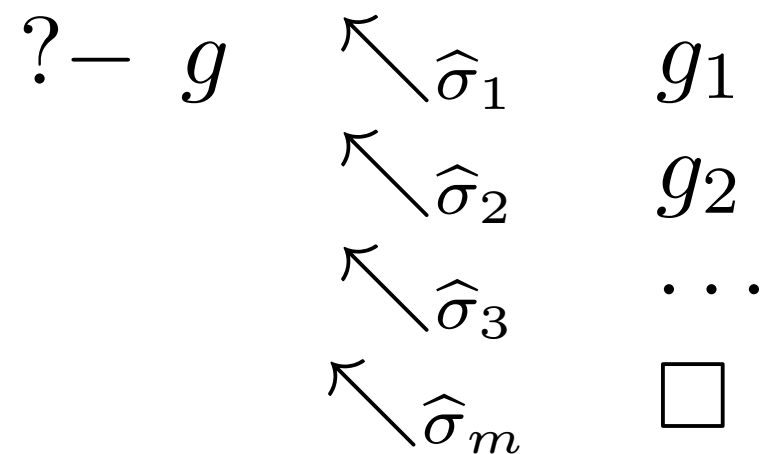
$$\begin{array}{l} \sigma : X \rightarrow T_{\Sigma, X} \\ Y \subseteq X \end{array} \quad \sigma|_Y(x) \triangleq \begin{cases} \sigma(x) & \text{if } x \in Y \\ x & \text{otherwise} \end{cases}$$

riportiamo solo questa informazione parziale

$$a_1, \dots, a_i, \dots, a_k \quad \nwarrow_{\hat{\sigma}} \quad (a_1, \dots, (b_1, \dots, b_n)\rho, \dots, a_k)\sigma$$

$$\hat{\sigma} \triangleq \sigma|_{\text{vars}(a_1, \dots, a_k)}$$

Risposte Calcolate



$$\sigma = \hat{\sigma}_1 \cdot \hat{\sigma}_2 \cdot \hat{\sigma}_3 \cdot \dots \cdot \hat{\sigma}_m$$

la sostituzione e' chiamata risposta calcolata (computed answer substitution) e viene calcolata come composizione delle risposte calcolate ad ogni passo

$$t(\sigma_1 \cdot \sigma_2) \stackrel{\Delta}{=} t\sigma_1\sigma_2 = \sigma_2(\sigma_1(t)) = (\sigma_2 \circ \sigma_1)(t)$$

Esempio

$$\Sigma_0 = \{0, \dots\} \quad \Pi_3 = \{\text{sum}, \dots\}$$

$$\Sigma_1 = \{s, \dots\}$$

somma codificata
come predicato

$\text{sum}(x, y, z)$ significa $x + y = z$

(un fatto)

L

$\text{sum}(0, y, y).$
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z).$

in PROLOG
le clausole
terminano con il
punto

2+2=?

il goal $? - \text{sum}(s(s(0)), s(s(0)), n)$.

L

$\text{sum}(0, y, y)$.
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z)$.

$\{\text{sum}(s(s(0)), s(s(0)), n) \stackrel{?}{=} \text{sum}(0, y', y')\}$ **fallisce**

$\{\text{sum}(s(s(0)), s(s(0)), n) \stackrel{?}{=} \text{sum}(s(x_1), y_1, s(z_1))\}$ **ha successo**

$$\sigma_1 = [x_1 = s(0), y_1 = s(s(0)), n = s(z_1)]$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

$$\text{sum}(s(s(0)), s(s(0)), n) \xleftarrow{\hat{\sigma}_1} (\text{sum}(x_1, y_1, z_1))\sigma_1$$

$$= \text{sum}(s(0), s(s(0)), z_1)$$

$\hat{\sigma}_1 = [n = s(z_1)]$

1+2=?

nuovo goal

$$\text{sum}(s(0), s(s(0)), z_1).$$

L

$\text{sum}(0, y, y).$
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z).$

$$\{\text{sum}(s(0), s(s(0)), z_1) \stackrel{?}{=} \text{sum}(0, y', y')\} \quad \text{fallisce}$$

$$\{\text{sum}(s(0), s(s(0)), z_1) \stackrel{?}{=} \text{sum}(s(x_2), y_2, s(z_2))\} \quad \text{ha successo}$$

$$\sigma_2 = [x_2 = 0, y_2 = s(s(0)), z_1 = s(z_2)]$$

$$\hat{\sigma}_2 = [z_1 = s(z_2)]$$

$$\text{sum}(s(s(0)), s(s(0)), n) \begin{array}{l} \nwarrow \hat{\sigma}_1 \\ \nwarrow \hat{\sigma}_2 \end{array} \begin{array}{l} \text{sum}(s(0), s(s(0)), z_1) \\ (\text{sum}(x_2, y_2, z_2))\sigma_2 \\ = \text{sum}(0, s(s(0)), z_2) \end{array}$$

$$\begin{array}{l} \hat{\sigma}_1 = [n = s(z_1)] \\ \hat{\sigma}_2 = [z_1 = s(z_2)] \end{array}$$

0+2=?

nuovo goal

$$\text{sum}(0, s(s(0)), z_2).$$

L

$\text{sum}(0, y, y).$
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z).$

$\{\text{sum}(0, s(s(0)), z_2) \stackrel{?}{=} \text{sum}(0, y_3, y_3)\}$ ha successo

$$\sigma_3 = [y_3 = s(s(0)), z_2 = s(s(0))]$$

$$\hat{\sigma}_3 = [z_2 = s(s(0))]$$

$$\text{sum}(s(s(0)), s(s(0)), n)$$

$\nwarrow \hat{\sigma}_1$

$$\text{sum}(s(0), s(s(0)), z_1)$$

$$\hat{\sigma}_1 = [n = s(z_1)]$$

$\nwarrow \hat{\sigma}_2$

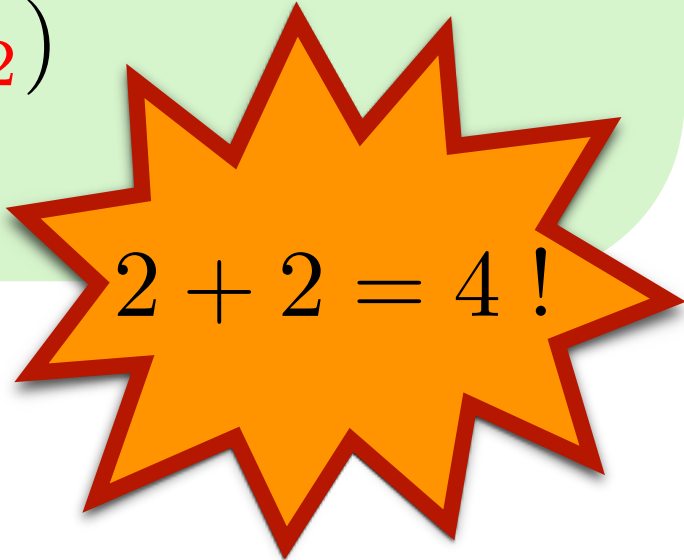
$$\text{sum}(0, s(s(0)), z_2)$$

$$\hat{\sigma}_2 = [z_1 = s(z_2)]$$

$\nwarrow \hat{\sigma}_3$

□

$$\hat{\sigma}_3 = [z_2 = s(s(0))]$$



$$\hat{\sigma}_1 \cdot \hat{\sigma}_2 \cdot \hat{\sigma}_3 = [n = s(s(s(s(0))))]$$

1+n=3?

il goal $? - \text{sum}(s(0), n, s(s(s(0))))$.

L

$\text{sum}(0, y, y)$.
 $\text{sum}(s(x), y, s(z)) :- \text{sum}(x, y, z)$.

$\{\text{sum}(s(0), n, s(s(s(0)))) \stackrel{?}{=} \text{sum}(0, y', y')\}$ **fallisce**

$\{\text{sum}(s(0), n, s(s(s(0)))) \stackrel{?}{=} \text{sum}(s(x_1), y_1, s(z_1))\}$ **ha successo**

$$\sigma_1 = [x_1 = 0, y_1 = n, z_1 = s(s(0))]$$

$$\hat{\sigma}_1 = []$$

$$\text{sum}(s(0), n, s(s(s(0)))) \xleftarrow{\hat{\sigma}_1} (\text{sum}(x_1, y_1, z_1))\sigma_1$$

$$= \text{sum}(0, n, s(s(0)))$$

$$\hat{\sigma}_1 = []$$

0+n=2?

nuovo goal

$$\text{sum}(0, n, s(s(0)))$$

L

sum(0, *y*, *y*).
 sum(*s(x)*, *y*, *s(z)*) :- sum(*x*, *y*, *z*).

$$\{\text{sum}(0, n, s(s(0))) \stackrel{?}{=} \text{sum}(0, y_2, y_2)\}$$

ha successo

$$\sigma_2 = [y_2 = s(s(0)), n = s(s(0))]$$

$$\hat{\sigma}_2 = [n = s(s(0))]$$

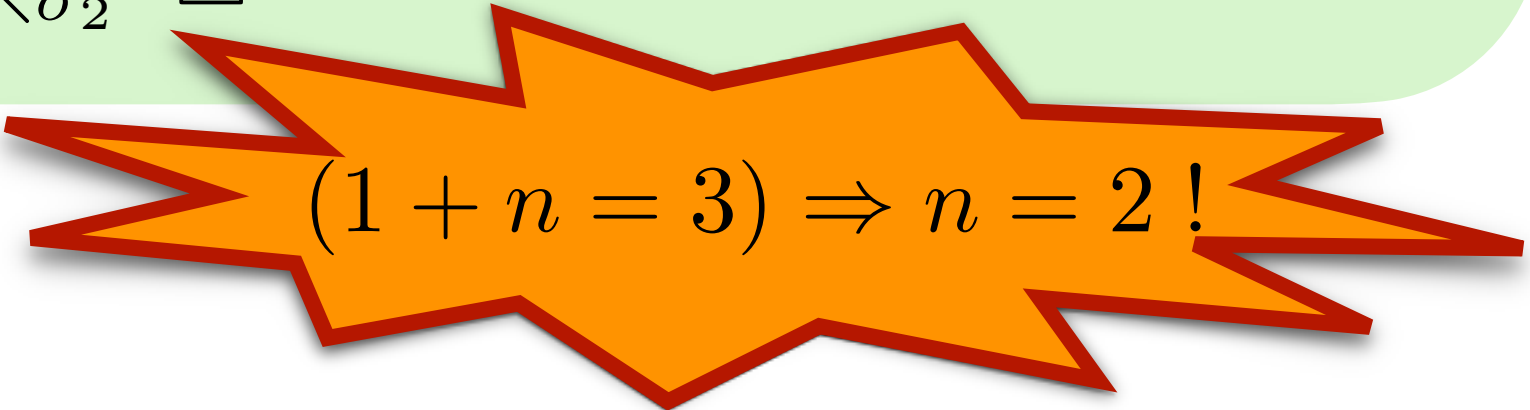
$$\text{sum}(s(0), n, s(s(s(0)))) \xleftarrow{\hat{\sigma}_1} \text{sum}(0, n, s(s(0)))$$

$$\xleftarrow{\hat{\sigma}_2} \square$$

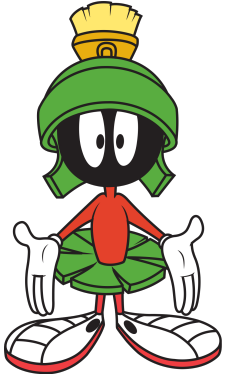
$$\hat{\sigma}_1 = []$$

$$\hat{\sigma}_2 = [n = s(s(0))]$$

$$\hat{\sigma}_1 \cdot \hat{\sigma}_2 = [n = s(s(0))]$$



Jumping creatures



Assuming that:

1. All jumping creatures are green
2. All small jumping creatures are Martians
3. All green Martians are intelligent
4. Ngtrks is small and green
5. Pgvdrk is a jumping Martian

Who is intelligent?

Antenati

Supponiamo di avere un predicato $\text{parent}(x,y)$ che ci dice che x è genitore di y .

- Definiamo un predicato $\text{sibling}(x,y)$ che è vero quando x e y hanno un genitore in comune
- Definiamo un predicato $\text{cousin}(x,y)$ che è vero quando x e y sono cugini
- Definiamo un predicato $\text{ancestor}(x,y)$ che è vero quando x è un antenato diretto (padre, nonno, bisnonno...) di y

Supponendo i seguenti fatti per parent

```
parent (alice, bob) .  
parent (alice, carl) .  
parent (bob, ella) .  
parent (carl, francisco) .
```

Posso derivare i seguenti goal?

```
?- sibling(ella, francisco) .  
?- sibling(ella, diana) .  
?- cousin(ella, francisco) .  
?- cousin(ella, diana) .  
?- ancestor(alice, ella) .  
?- ancestor(carl, ella) .
```

Esercizio

Un albero binario T è vuoto (nullo) oppure è composto da un valore radice e due successori, che sono essi stessi alberi binari.

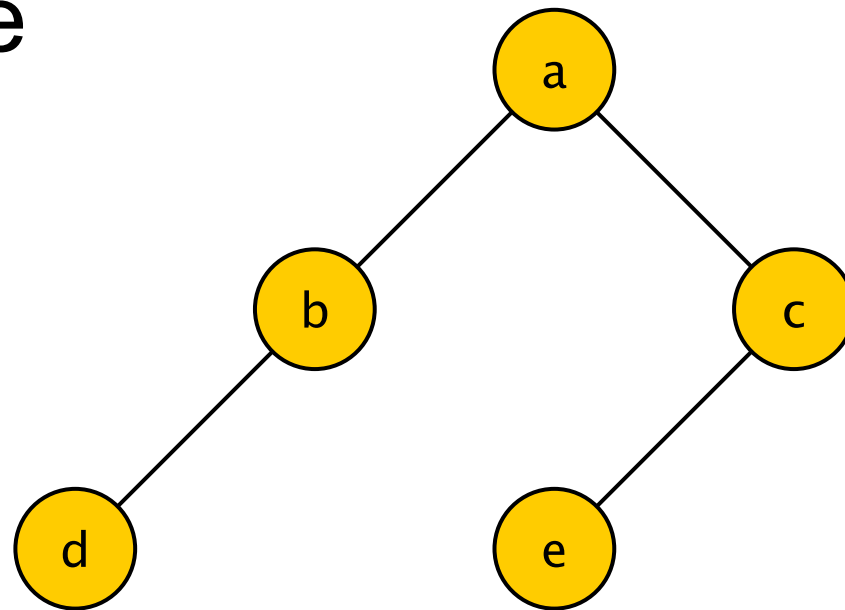
T è simmetrico se è possibile tracciare una linea verticale attraverso il nodo radice e quindi il sottoalbero destro è l'immagine speculare del sottoalbero sinistro (siamo interessati solo alla struttura, i valori non sono rilevanti).

1. Data la firma qui sotto, scrivi le clausole di Horn per verificare se un albero è l'immagine speculare di un altro (definisci il predicato `mirror`)
2. Poi estendi il codice per controllare se un albero è simmetrico (definisci il predicato `sym`).

$$\Sigma_0 = \{\text{nil}\} \quad \Sigma_3 = \{\text{node}\} \quad \Pi_1 = \{\text{sym}\} \quad \Pi_2 = \{\text{mirror}\}$$

Esercizio

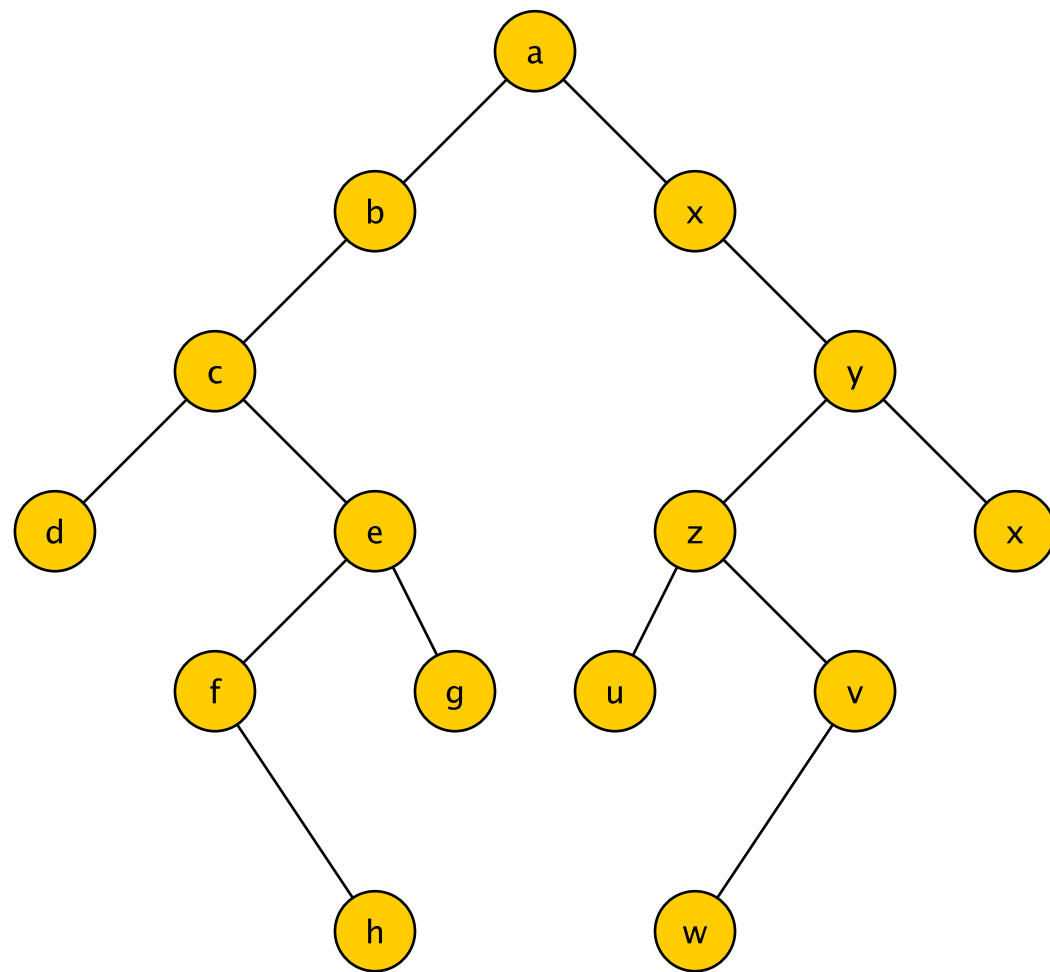
esempio:
alberi come
termini



```
node( a , node( b , node( d , nil , nil ) ,  
nil ) ,  
node( c , node( e , nil , nil ) ,  
nil ) )
```

Esercizio

un esempio di
albero simmetrico



e uno che non e'
simmetrico

