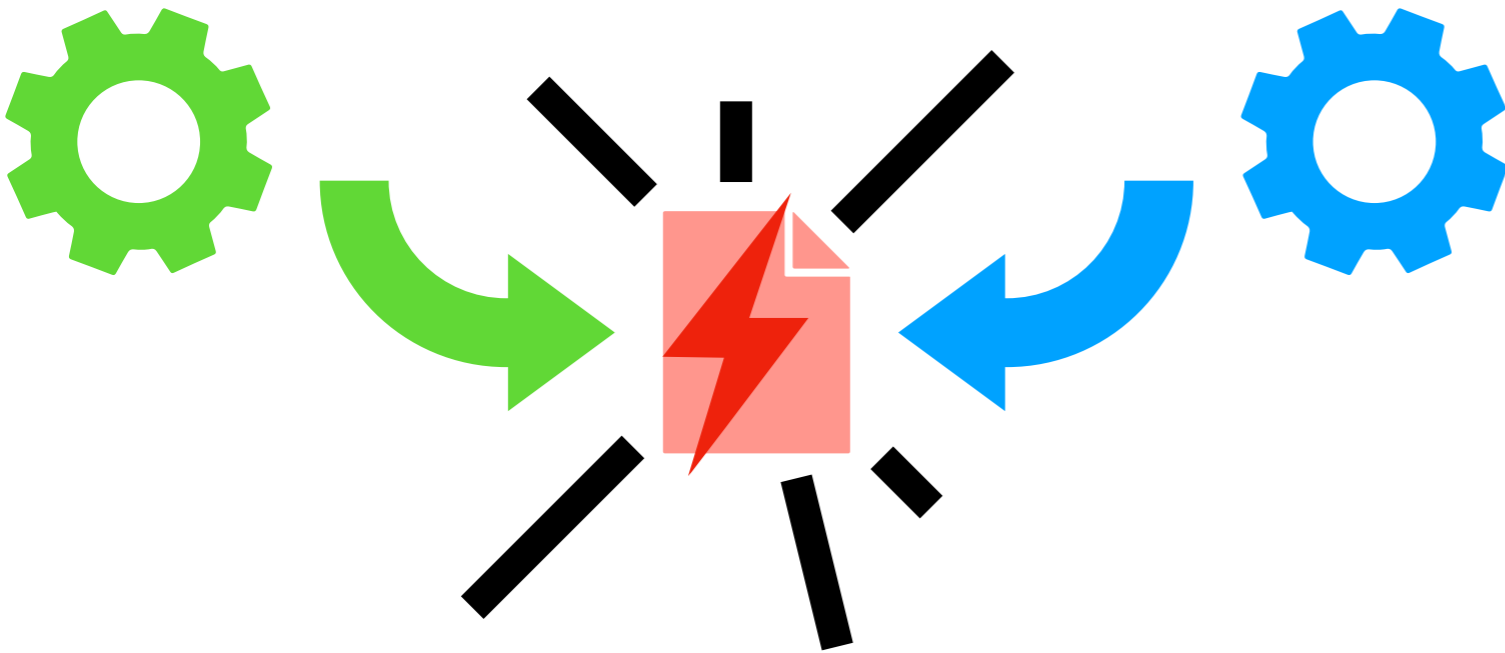


Linguaggi di Programmazione



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)

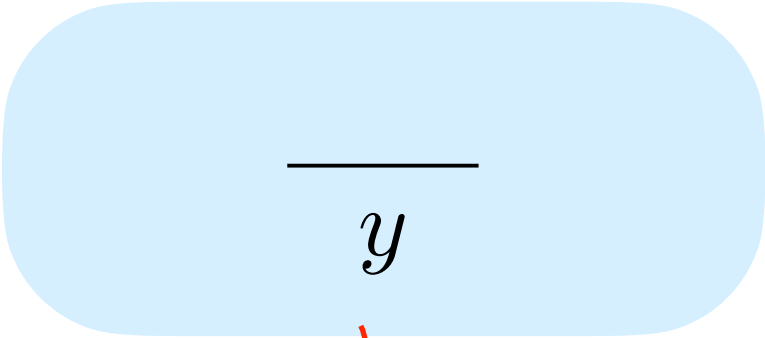
x_1, \dots, x_n, y sono formule

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


$$\frac{\quad}{y}$$

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 di 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 $d = (\frac{\quad}{y})$ è un assioma di R ;
- oppure $d = (\frac{d_1, \dots, d_n}{y})$ ed esiste un'istanza di regola di inferenza di R
 $\frac{x_1, \dots, x_n}{y}$
e n derivazioni
 $d_1 \Vdash_R x_1, \dots, d_n \Vdash_R x_n$.

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

Esempio

$$S ::= \epsilon \mid (S) \mid SS$$

$$R = \left\{ \frac{}{\epsilon \in \mathcal{L}}, \frac{S \in \mathcal{L}}{(S) \in \mathcal{L}}, \frac{S_0 \in \mathcal{L} \quad S_1 \in \mathcal{L}}{S_0 S_1 \in \mathcal{L}} \right\}$$

$$d \Vdash_R ((())()) \in \mathcal{L}$$

$$((())()) \in \mathcal{L}$$

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

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

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

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

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

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

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

$$\nearrow 3 \longrightarrow 3, 4 \longrightarrow 4 \quad \nearrow 4 \longrightarrow 4 \quad \nearrow \square$$

derivazione
goal-oriented

niente da provare

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

Il Prolog è un linguaggio di programmazione dichiarativo semplice e potente, basato sulla logica dei predicati del primo ordine.

PROgrammation en LOGique

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

Ogni psichiatra è una persona.
Ogni persona che analizza è malata.
Jacques è uno psichiatra di Marsiglia.
Jacques è una persona?
Dove si trova Jacques?
Jacques è malato?
Si. A Marseille. Non so.

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

$$a_1, \dots, a_n \quad \begin{array}{l} \text{una congiunzione (anche} \\ \text{vuota) di formule atomiche} \end{array}$$
$$a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n$$

Esempio

$X = \{N, E, E_0, E_1, \dots, n, n_0, n_1, \dots\}$ un insieme di variabili

$\Sigma_0 = \{0, 1, 2, \dots, 0, 1, 2, \dots\}$ un insieme di costanti

$\Sigma_2 = \{- \oplus -, - \otimes -\}$ un insieme di operatori binari (infissi)

$\Pi_2 = \{- \longrightarrow -\}$ simbolo di predicato binario (infisso)

formula atomica

$$E \oplus 2 \longrightarrow 5$$

formula

$$E \oplus 2 \longrightarrow 5, E \otimes 7 \longrightarrow n$$

Programmi Logici

Clausole di Horn

una formula
atomica
(la TESTA)

$h :- r$

una formula
(il CORPO)

← Implicazione logica

$a :- a_1, \dots, a_n$

simile a

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

un insieme (o una lista) di clausole di Horn

Un programma
logico L

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

Esempio

"Chi vive nella stessa città di Anna?"
Il predicato

stessa-città(Anna, X)

e' vero?

"Chi vive a Torino?"
Il predicato

stessa-città(X, torino)

e' vero ??

vive(anna, roma).
vive(mario, milano).
vive(giovanni, torino).
vive(maria, roma).
vive(paolo, milano)

"Chi vive nella stessa città?"
Il predicato

stessa-città(X, Y)

e' vero ??

stessa-città(X, Y) :- vive(X, Z), vive(Y, Z).

"Anna vive a Roma?"

Il predicato

stessa-città(Anna, roma)

e' vero ??

Refutazione di un goal

Ci chiediamo se $a_1 \wedge a_2 \dots \wedge a_n$ e' una conseguenza logica della nostra base di conoscenza (il nostro insieme di clausole)

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

Aggiungiamo all'insieme delle regole la negazione della formula che vogliamo dimostrare

? $\neg a_1, \dots, a_n$

Goal

false $\leftarrow a_1, \dots, a_n$

e cerchiamo di ottenere una contraddizione

Un esempio

```
vive(anna, roma).
vive(mario, milano).
vive(giovanni, torino).
vive(maria, roma).
vive(paolo, milano)

stessa-citta(X, Y) :- vive(X, Z), vive(Y, Z), X \= Y.

% Goal
% "Anna vive a Roma?"
?- stessa-citta(anna, roma)

% Goal
% "Chi vive nella stessa città di Anna?"
?- stessa-citta(anna, X)

% Goal
% "Chi vive a Torino?"
?- stessa-citta(X, torino)

?- % Goal
% "Chi vive nella stessa città?"
?- stessa-citta(X, Y)
```

Applicazioni

un programma logico serve a rispondere alla seguente domanda:

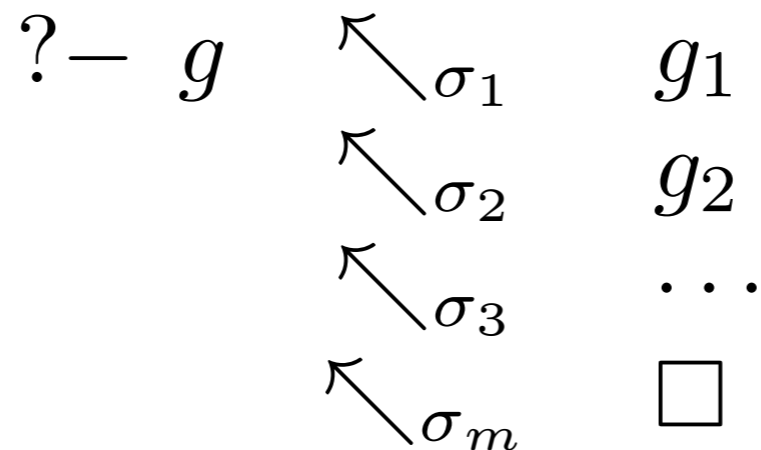
data una formula g (goal) che vogliamo dimostrare, quali sono le istanze di g valide?

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

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 l' mgu al nuovo obiettivo

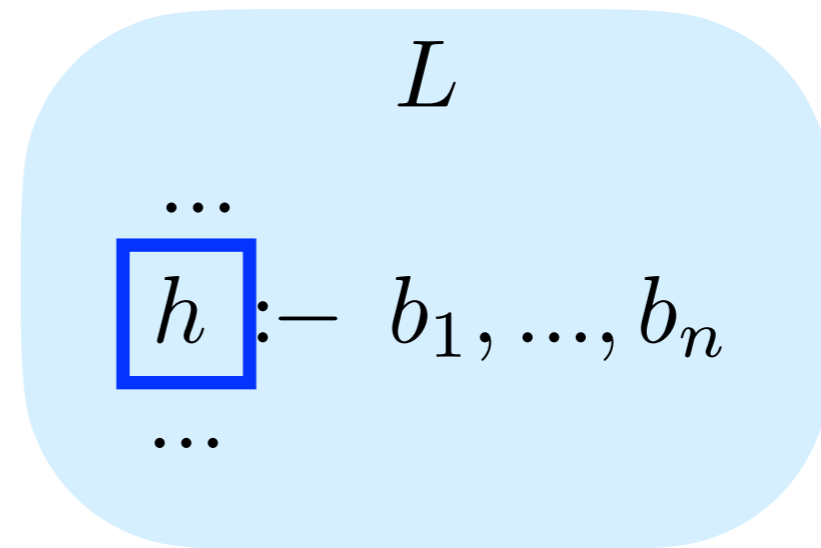


allora $g\sigma_1\sigma_2\dots\sigma_m$ è un teorema

Risoluzione SLD

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

a_i e h unificano!!



Ripeti i passi seguenti finche' 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 piú 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 conflitti sulle variabili

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 $h \text{ :- } b_1, \dots, b_n \in L$;
3. sia ρ un renaming per le variabili in $vars(h \text{ :- } b_1, \dots, b_n)$;
4. $(h \text{ :- } b_1, \dots, b_n)\rho$ 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 : $(a_1, \dots, a_{i-1}, (b_1, \dots, b_n)\rho, a_{i+1}, \dots, a_k)\sigma$.

ρ ridenominazione
delle variabili

Attenzione

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

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

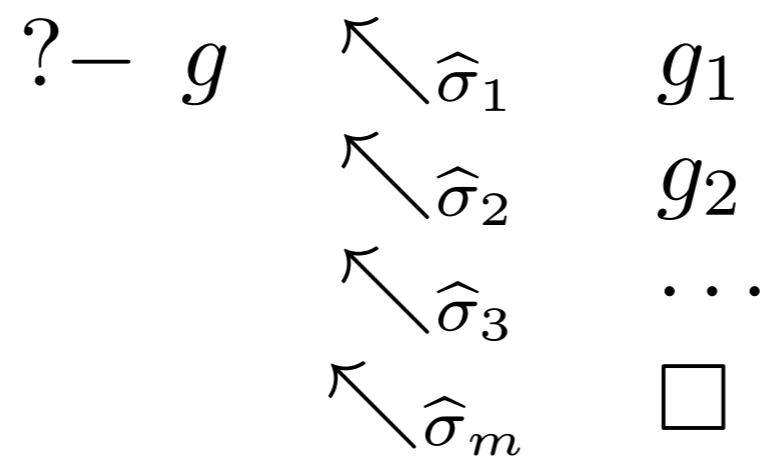
riportiamo solo questa informazione parziale

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

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

$$? - a_1, \dots, \boxed{a_i}, \dots, a_k \quad \boxed{h} :- b_1, \dots, b_n$$

Risposte Calcolate



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

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 $x + y = z$

L

(un fatto)

$\text{sum}(0, y, y).$

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

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

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

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

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

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

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

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

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

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

0+2=?

nuovo goal

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

L

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

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

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

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

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

$$\nwarrow \hat{\sigma}_1$$

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

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

$$\nwarrow \hat{\sigma}_2$$

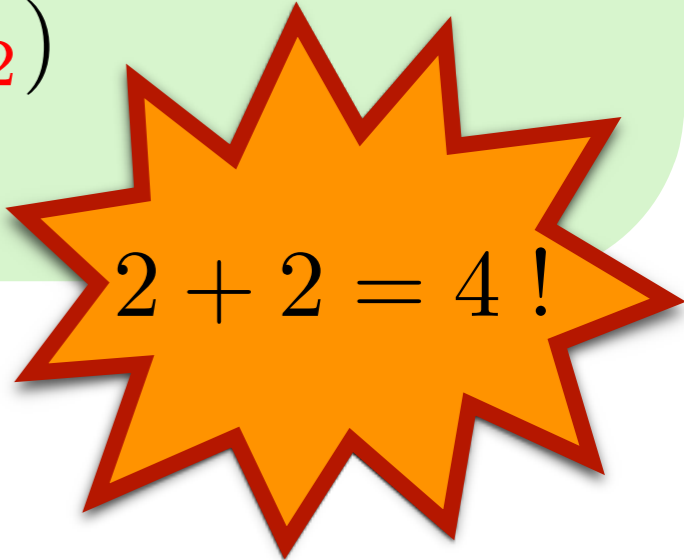
$$\text{sum}(0, \text{s}(\text{s}(0)), z_2)$$

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

$$\nwarrow \hat{\sigma}_3$$

□

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



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

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

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

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

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

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

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

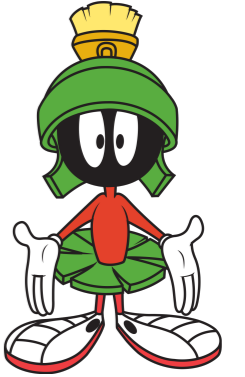
$$\hat{\sigma}_2 = [n = s(s(0))]$$

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



$$(1 + n = 3) \Rightarrow n = 2!$$

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?

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?

```
green(X) :- jumping(X).
martian(X) :- small(X) , jumping(X).
intelligent(X) :- green(X) , martian(X).
small(ngtrks).
green(ngtrks).
jumping(pgvdrk).
martian(pgvdrk).
?— intelligent(W).
```

intelligent(*W*)

↙ green(*W*) , martian(*W*)

↙ jumping(*W*) , martian(*W*)

↙ martian(pgvdrk)

↙ □ [*W* = pgvdrk]

martian(ngtrks)

↙ small(ngtrks) , jumping(ngtrks)

↙ jumping(ngtrks)

fail!

Antenati

Supponiamo di avere un predicato `parent(x,y)` che ci dice che `x` e' genitore di `y`.

- Definiamo un predicato `sibling(x,y)` che e' vero quando `x` e `y` hanno un genitore in comune

```
sibling(x,y) :- parent(z,x),parent(z,y) .
```

- Definiamo un predicato `cousin(x,y)` che e' vero quando `x` e `y` sono cugini

```
cousin(x,y) :- parent(z,x),parent(v,y), sibling(z,v) .
```

- Definiamo un predicato `ancestor(x,y)` che e' vero quando `x` e' un antenato diretto (padre, nonno, bisnonno...) di `y`

```
ancestor(x,y) :- parent(x,y) .
```

```
ancestor(x,y) :- parent(x,z),ancestor(z,y) .
```

Supponendo i seguenti fatti per parent

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

ancestor (x, y) :- parent (x, y) .
ancestor (x, y) :- parent (x, z) , ancestor (z, y) .
sibling (x, y) :- parent (z, x) , parent (z, y) .
cousin (x, y) :- parent (z, x) , parent (v, y) ,
                 sibling (z, v) .
```

Posso derivare i seguenti goal?

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

Esercizio 1

Sia $\Sigma_0 = \{0\}$ e $\Sigma_1 = \{s\}$, estendere il programma logico che definisce il predicato $\text{sum} \in \Pi_3$ (visto in classe) per definire:

1. un predicato $\text{prod} \in \Pi_3$ per calcolare il prodotto di due numeri;
2. un predicato $\text{pow} \in \Pi_3$ per calcolare la potenza;
3. un predicato $\text{div} \in \Pi_2$ per dire se il primo argomento divide il secondo.

Esercizio 2

Data la sintassi dell'Es. 1, risolvete i problemi di unificazione seguenti

1. $G1 = \{\text{prod}(s(x), y, s(z)) \text{ ?= } \text{prod}(y, z, x)\}$

2. $G2 = \{\text{pow}(x, s(y), x) \text{ ?= } \text{pow}(s(y), z, z)\}$

3. $G3 = \{\text{div}(x, s(y)) \text{ ?} = \text{div}(z, x) , \text{div}(y, s(z)) \text{ ?} = \text{div}(u, s(u))\}$

Esercizio 3

Dati i programmi logici dell'Es. 1, scrivere alcune possibili derivazioni per i goal seguenti :

1. $\text{somma}(x, s(0), s(s(0)))$
2. $\text{prod}(s(s(0)), y, s(s(0)))$
3. $\text{div}(z, s(s(0)))$

Esercizio 4

sintomo(john, tosse).

sintomo(john, febbre).

sintomo(maria, mal_di_testa).

sintomo(maria, nausea).

diagnosi(X, influenza) :- sintomo(X, tosse), sintomo(X, febbre).

diagnosi(X, influenza) :- sintomo(X, febbre), sintomo(X, nausea).

diagnosi(X, emicrania) :- sintomo(X, mal_di_testa), sintomo(X, nausea).

Cosa calcoliamo per il seguente goal

?-diagnosi(X, Y).

Esercizio da consegnare

(per mail entro la prox settimana)

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

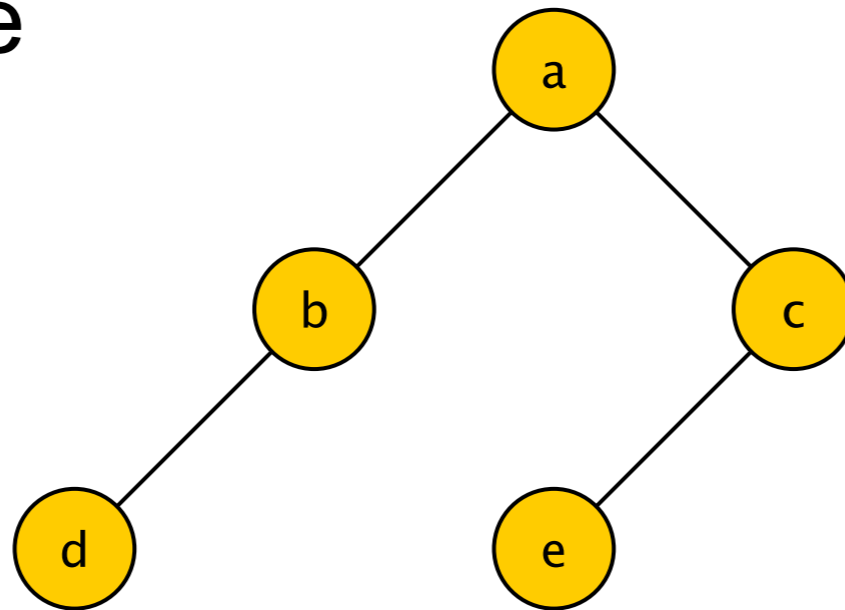
T è simmetrico se è possibile tracciare una linea verticale attraverso il nodo radice e quindi il sottoalbero destro è l'immagine speculare del sottoalbero sinistro (siamo interessati solo alla struttura, i valori non sono rilevanti).

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 sum).

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

Esercizio

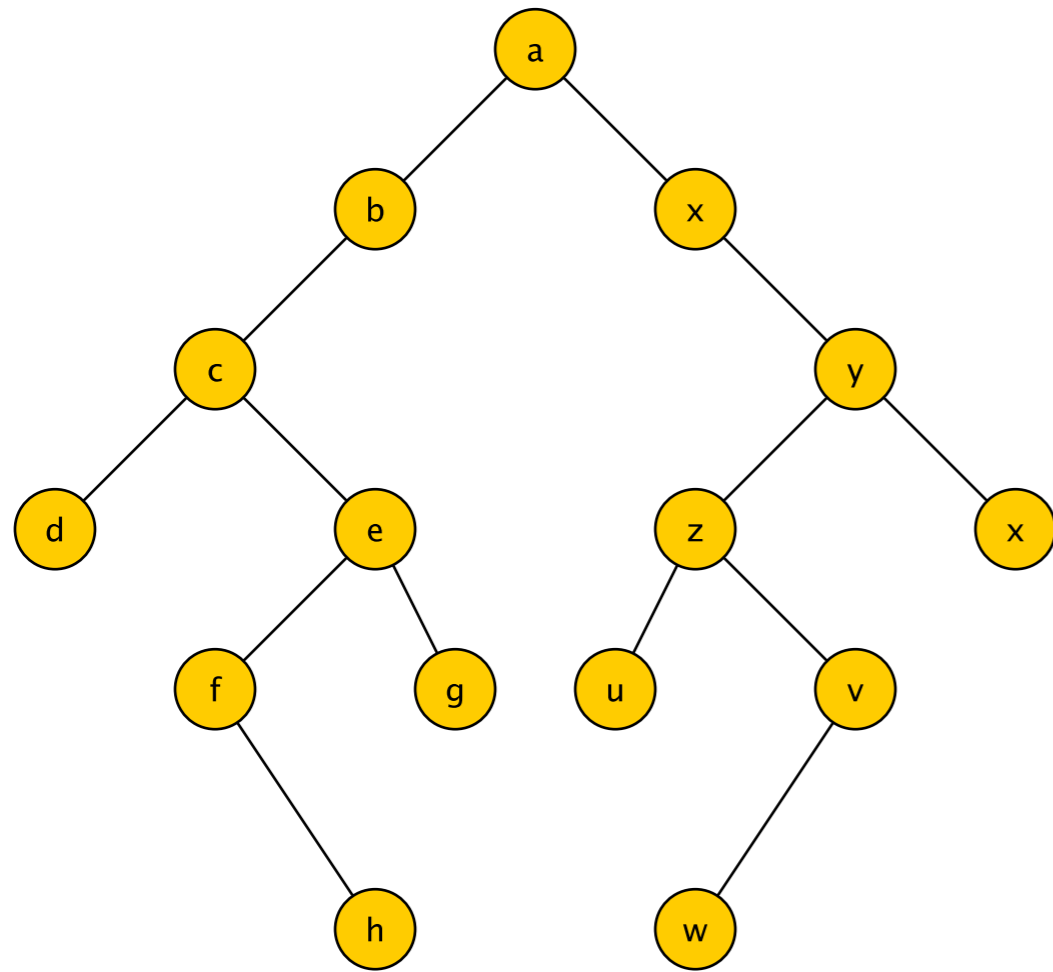
esempio:
alberi come
termini



```
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

