DISCO
DIPARTIMENTO DI INFORMATICA SISTEMISTICA E COMUNICAZIONE

Linguaggi di Programmazione 2022-2023

Prolog e Programmazione Logica II

Marco Antoniotti
Gabriella Pasi
Rafael Peñaloza

Prolog

- Lavora su strutture ad **albero**
  - *anche i programmi sono strutture dati manipolabili* (mediante predicati “extra-logici”)
- Utilizza ricorsione (non c’è una nozione semplice di assegnamento)

- Metodologia di programmazione
  - concentrarsi sulla specifica del problema rispetto alla strategia di soluzione
- Un programma Prolog è un insieme di **clausole di Horn** che rappresentano
  - **fatti** riguardanti gli oggetti in esame e le relazioni che intercorrono tra di loro
  - **regole** sugli oggetti e sulle relazioni (*se* … *allora* …)
  - **interrogazioni** (**goals** o **queries**; clausole «senza testa»), sulla base della conoscenza definita

Clausole

- **Implicazione**
  - Sappiamo che $\neg B \vee A$ corrisponde a $B \rightarrow A$, oppure «A implicato da B» (che si legge “A se B”), scritto anche come $A \leftarrow B$
  - Data una clausola

    $A_1 \vee A_2 \vee \ldots \vee A_n \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_m$

    usando la nota **formula di riscrittura** (teorema di De Morgan)

    $(A_1 \vee A_2 \vee \ldots \vee A_n) \vee \neg(B_1 \wedge B_2 \wedge \ldots \wedge B_m)$

    da cui otteniamo

    $(A_1 \vee A_2 \vee \ldots \vee A_n) \leftarrow (B_1 \wedge B_2 \wedge \ldots \wedge B_m)$
  - Ovviamente anche il percorso inverso è valido
  - **Definizione:** le A e le B nelle formule precedenti si dicono **letterali**, **negativi** quelli negati e **positivi** gli altri

Clausole di Horn

- Una **clausola** di **Horn** ha – al più – un solo letterale positivo

$$A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$$

- In particolare possiamo classificare le clausole di Horn nel modo seguente
  - Fatti $A \leftarrow$
  - Regole $A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$
  - Goals $\leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$
  - Contraddizione $\leftarrow$

- E, come già abbiamo visto, in Prolog questi diventano
  - Fatti `A.`
  - Regole `A :- B1, B2, …, Bm.`
  - Goals `:- B1, B2, …, Bm.`
  - Contraddizione `fail`

Un programma logico

- Un programma logico che manipola la rappresentazione «unaria» dei numeri naturali (l'insieme $\mathbb{N}$)

```
sum(0, X, X).
sum(s(X), Y, s(Z)) :- sum(X, Y, Z).
```

- Possiamo interpretare **`s(N)`** come il *successore* del numero **`N`**
- Quindi `0`, **`s(0)`**, **`s(s(0))`**, **`s(s(s(0)))`**… rappresentano 0, 1, 2, 3…
- Questo programma **definisce** la somma fra due numeri naturali (rappresentati in "unario")

Un programma logico

- Possiamo interrogare il «programma» nel modo seguente

```
∃X sum(s(0), 0, X)              {X / s(0)}
∃W sum(s(s(0)), s(0), W)        {W / s(s(s(0)))}
```

- Mediante il procedimento di negazione e di trasformazione in sintassi Prolog otteniamo

```
:- sum(s(0), 0, N)              {N / s(0)}
:- sum(s(s(0)), s(0), W)        {W / s(s(s(0)))}
```

Sostituzioni

- Nell'esempio precedente abbiamo visto le **sostituzioni** `{W / s(s(s(0)))}` e `{N / s(0)}`

- Una sostituzione ci dice con che «valori» (che possono essere altre variabili) possiamo sostituire le variabili in un termine

- Di solito si denota una sostituzione nel modo seguente

$$\sigma = \{X_1 / v_1, X_2 / v_2, ..., X_k / v_k\}$$

- Una sostituzione può essere considerata come una *funzione*, applicabile ad un termine (T è l'insieme dei termini)

$$\sigma : T \rightarrow T$$

- Esempio (data la sostituzione σ = `{X/42, Y/foo(s(0))}`)

$$\sigma(\texttt{bar(X, Y)}) = \texttt{bar(42, foo(s(0)))}$$

Esecuzione di un programma

- Una computazione corrisponde al tentativo di dimostrare, tramite la regola di risoluzione, che una formula segue logicamente da un programma (è un teorema).

- Inoltre, si deve determinare una sostituzione per le variabili del goal (detto anche **query**) per cui la query segue logicamente dal programma.

- Dato un programma **P** e la query

  `:- p(t1, t2, … , tm).`

  se **`X1`**, **`X2`**,…, **`Xn`** sono le variabili che compaiono in **`t1`**, **`t2`**,…, **`tm`**, il significato della query è

  $\exists$ **`X1`**, **`X2`**,…, **`Xn`** **`.p(t1, t2, … , tm)`**

  e l'obiettivo è quello di trovare una sostituzione

  `s` = {**`X1`**/s1, **`X2`**/s2, …, **`Xn`**/sn}

  dove gli s_i sono termini tali per cui **P** $\vdash$ **`s[p(t1, t2, … , tm)]`**

Esecuzione di un programma

- Dato un insieme di clausole di Horn è possibile derivare la clausola vuota solo se c`è n'è almeno senza testa – ovvero se abbiamo almeno una «query» (un «goal») G_0 da provare

- Si deve dimostrare che da $\mathbf{P} \cup \{G_0\}$ è possibile derivare la clausola vuota $\Rightarrow$ dimostrazione per assurdo mediante applicazione del *Principio di Risoluzione*.

- **Come?**
  - **Problema**: se si tentassero ad ogni passo tutte le risoluzioni possibili e si aggiungessero le clausole inferite all'insieme di partenza si avrebbe una **esplosione combinatoria**
  - Si deve adottare una strategia di soluzione opportuna (una variante più vincolata del Principio di Risoluzione).

Risoluzione ad Input Lineare (SLD)

- Il sistema Prolog dimostra la veridicità o meno di un'interrogazione (un goal) eseguendo *una sequenza di passi di risoluzione*
  - **L'ordine complessivo con cui questi passi vengono eseguiti rende i sistemi di prova di teoremi basati su risoluzione più o meno "efficienti"**

- In Prolog la risoluzione avviene sempre fra l'ultimo goal derivato in ciascun passo e una «clausola di programma»; mai fra due clausole di programma o fra una clausola di programma ed un goal derivato in precedenza
  - Questa particolare forma di risoluzione viene detta **Risoluzione-SLD** (*Selection* function for *Linear* and *Definite* sentences *Resolution*; dove le "frasi lineari" sono essenzialmente le clausole di Horn).

Risoluzione ad input lineare (SLD)

- A partire dal goal G_i

  $G_i \equiv$ **?- $A_{i,1}$, $A_{i,2}$, $A_{i,3}$, …, $A_{i,m}$.**

  e dalla regola

  A_r :- $B_{r,1}$, $B_{r,2}$, …, $B_{r,k}$.

  se esiste un unificatore σ tale che $\sigma[A_r] = \sigma[A_{i,1}]$,
  allora si ottiene un nuovo goal G_{i+1}

  $G_{i+1} \equiv$ **?- $B'_{r,1}$, $B'_{r,2}$, …, $B'_{r,k}$, $A'_{i,2}$, $A'_{i,3}$, …, $A'_{i,m}$.**

  Questo è **un** passo di risoluzione eseguito dal sistema Prolog (dove gli $A'_{\ldots}$ e i $B'_{\ldots}$ sono i risultati $\sigma[A_{\ldots}] = A'_{\ldots}$ e $\sigma[B_{\ldots}] = B'_{\ldots}$
- **Nota:** la scelta di unificare il **primo** sottogoal di G_i è **arbitraria** (anche se *comoda*); scegliere $A_{i,m}$ o $A_{i,c}$ con $c \in [1, m]$ casuale sarebbe ugualmente *lecito*

Risoluzione ad input lineare (SLD)

- Altro caso: a partire dal goal

$G_i \equiv$ **?- $A_{i,1}$, $A_{i,2}$, $A_{i,3}$, …, $A_{i,m}$.**

e dalla regola (ovvero dal fatto):

A_r.

se esiste un unificatore σ tale che $\sigma[A_r] = \sigma[A_{i,1}]$,
allora si ottiene un nuovo goal

$G_{i+1} \equiv$ **?- $A'_{i,2}$, $A'_{i,3}$, …, $A'_{i,m}$.**

Ovvero, il goal G_{i+1} ha dimensioni minori rispetto a G_i avendo $m - 1$ sotto-goals.

Risoluzione ad input lineare (SLD)

- Ripetiamo: nella risoluzione SLD, il passo di risoluzione avviene sempre fra l'ultimo goal e una clausola di programma.

- Il risultato finale può essere
  - **Successo**
    viene generata la clausola vuota, ovvero se per *n* finito G_n è uguale alla clausola vuota $G_n \equiv :-$
  - **Insuccesso finito**
    se per *n* finito, G_n non è uguale a $:-$ e non è più possibile derivare un nuovo risolvente da G_n ed una clausola d programma
  - **Insuccesso infinito**
    se è sempre possibile derivare nuovi risolventi tutti diversi dalla clausola vuota

- La sostituzione di risposta è la sequenza di unificatori usati; applicata alle variabili nei termini del goal iniziale dà la risposta finale

Risoluzione ad input lineare (SLD)

- Durante il processo di generazione di goal intermedi si costruiscono delle varianti dei letterali e delle clausole coinvolti mediante la rinominazione di variabili
- Una variante per una clausola *C* é la clausola *C'* ottenuta da *C* rinominando le sue variabili (renaming)

- **Esempio**

```
p(X) :- q(X, g(Z)).
```

è equivalente alla clausola con variabili rinominate

```
p(X1) :- q(X1, g(FooFrobboz)).
```

Strategia di selezione di un sotto-goal

- Possono esserci più clausole di programma utilizzabili per applicare la risoluzione con il goal corrente.
- Si possono adottare diverse strategie di ricerca per queste clausole
  - **In profondità** (**Depth First**)
    si sceglie una clausola e si mantiene fissa questa scelta, finchè non si arriva alla clausola vuota o alla impossibilità di fare nuove risoluzioni; in questo ultimo caso si riconsiderano le scelte fatte precedentemente
  - **In ampiezza** (**Breadth First**)
    si considerano in parallelo tutte le possibili alternative

- *Il **Prolog** adotta una strategia di risoluzione in profondità con **backtracking***
  - Permette di risparmiare memoria
  - Non è completa per le clausole di Horn

Alberi di derivazione SLD

- Dato un programma logico **P**, un goal G_0 e una regola di calcolo ***R***, un **albero SLD** per $\mathbf{P} \cup \{G_0\}$ via ***R*** è definito sulla base del processo di prova visto precedentemente
  - Ciascun **nodo** dell'albero è un goal (possibilmente vuoto)
  - La **radice** dell'albero SLD è il goal G_0
  - dato il nodo

    $:- \mathbf{A}_1, \ldots, \mathbf{A}_{m-1}, \mathbf{A}_m, \mathbf{A}_{m+1}, \ldots, \mathbf{A}_k$

    se A_m è il sottogoal selezionato dalla regola di calcolo ***R***, allora questo nodo (genitore) ha un nodo figlio per ciascuna clausola del tipo

    $C_i \equiv \mathbf{A}_i :- \mathbf{B}_{i,1}, \ldots, \mathbf{B}_{i,q}$
    $C_k \equiv \mathbf{A}_k.$

    di **P** tale che $\mathbf{A}_i$ e $\mathbf{A}_m$ ($\mathbf{A}_k$ e $\mathbf{A}_m$) sono unificabili attraverso la sostituzione più generale σ
  - Il nodo figlio è etichettato con la clausola goal

    $:- \sigma[\mathbf{A}_1, \ldots, \mathbf{A}_{m-1}, \mathbf{B}_{i,1}, \ldots, \mathbf{B}_{i,q}, \mathbf{A}_{m+1}, \ldots, \mathbf{A}_k]$
    $:- \sigma[\mathbf{A}_1, \ldots, \mathbf{A}_{m-1}, \mathbf{A}_{m+1}, \ldots, \mathbf{A}_k]$

    e il ramo dal nodo padre al figlio è etichettato dalla sostituzione σ e dalla clausola selezionata C_i (o C_k)
  - Il nodo vuoto (indicato con "`:-`") non ha figli

Alberi di derivazione SLD

- **Ripetiamo**

- La regola ***R*** è variabile
  - Può essere "scelta del sottogoal più a **sinistra**" (se c'è)
    Detta regola **Left-most**
  - Può essere "scelta del sottogoal più a **destra**" (se c'è)
    Detta regola **Right-most**
  - Oppure "scelta di un sottogoal **a caso**"
  - O "scelta del sottogoal **migliore**" (data un'opportuna definizione di "migliore")

- Il Prolog adotta una regola **Left-most**

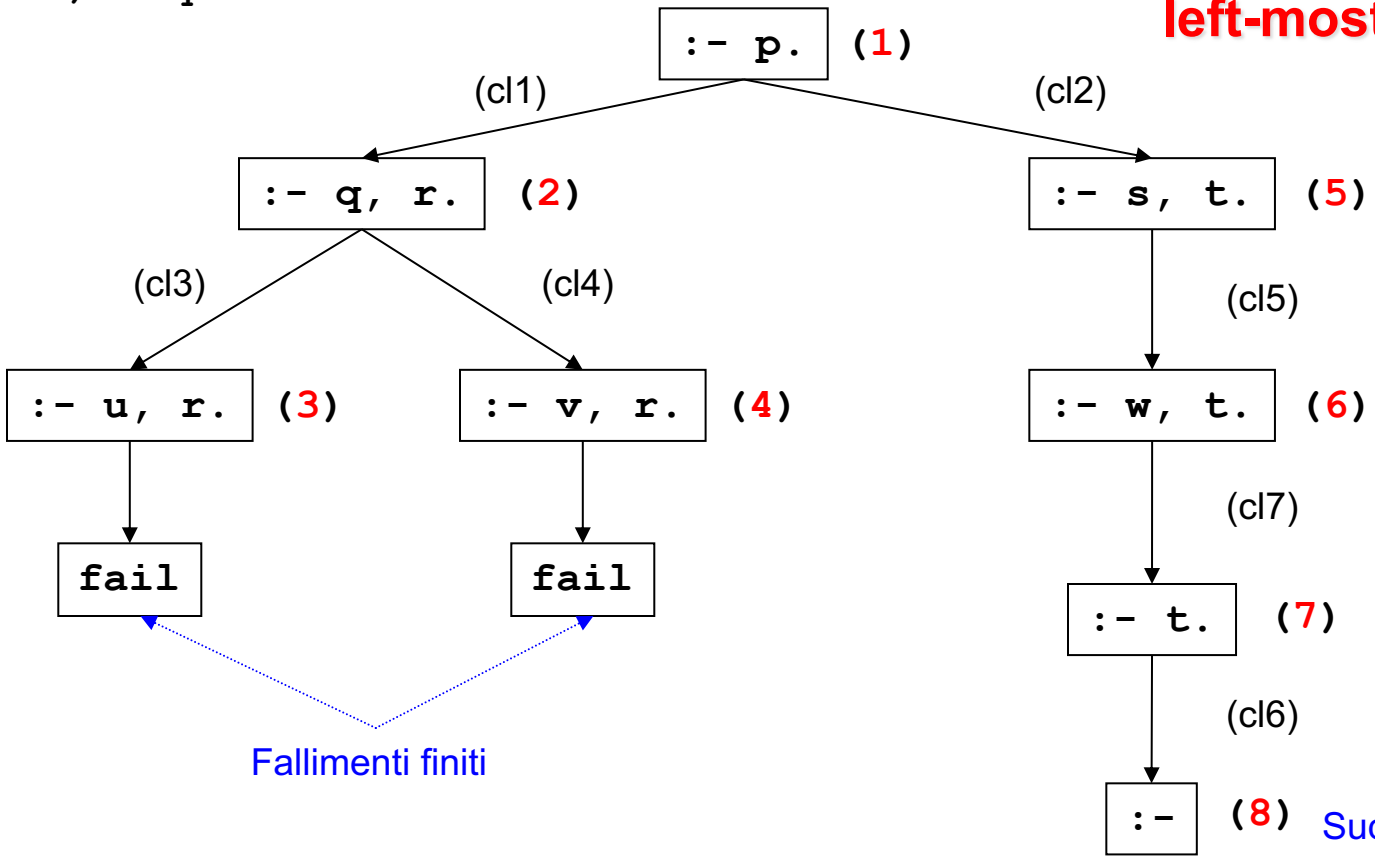
- L'albero SLD (implicito!) generato dal sistema Prolog ordina i figli di un nodo secondo l'ordine dall'alto verso il basso delle regole e dei fatti del programma **P**

Esempio di albero di derivazione

```
(cl1)   p :- q, r.        (cl5)   s:- w.
(cl2)   p :- s, t.        (cl6)   t.
(cl3)   q :- u.           (cl7)   w.
(cl4)   q :- v.
```

goal `:- p.`

Albero di risoluzione left-most



Il Prolog adotta una strategia di attraversamento dell'albero (implicito) SLD in profondità (**Depth First**) con backtracking. Il numero in rosso rappresenta l'ordine di attraversamento usato dal sistema Prolog

Regola di calcolo

- Ad ogni ramo di un albero SLD corrisponde una derivazione SLD
  - Ogni ramo che termina con il nodo vuoto (“:-”) rappresenta una derivazione SLD di successo

- La regola di calcolo influisce sulla struttura dell’albero per quanto riguarda sia l’ampiezza sia la profondità

- Tuttavia non influisce su *correttezza* e *completezza*; quindi, qualunque sia ***R***, il *numero di cammini di successo* (se in numero finito) è lo stesso in tutti gli alberi SLD costruibili per $\mathbf{P} \cup \{G_0\}$

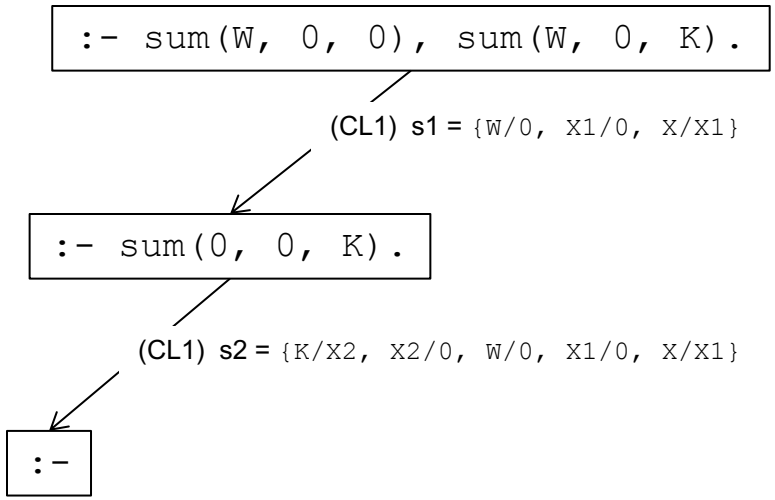
- ***R*** influenza solo il numero di cammini di fallimento (finiti ed infiniti).

Un altro esempio

```
sum(0,X,X).                                   (CL1)
sum(s(X), Y, s(Z)):- sum(X, Y ,Z).            (CL2)
```

G_0 = `:- sum(W, 0, 0), sum(W, 0, K).`

Albero SLD con regola di calcolo "left-most"



Le variabili `X1` e `X2` sono il risultato dell'operazione di ridenominazione (renaming) della variabile `X` in CL1; appena una clausola viene presa in considerazione le sue variabili sono ridenominate.

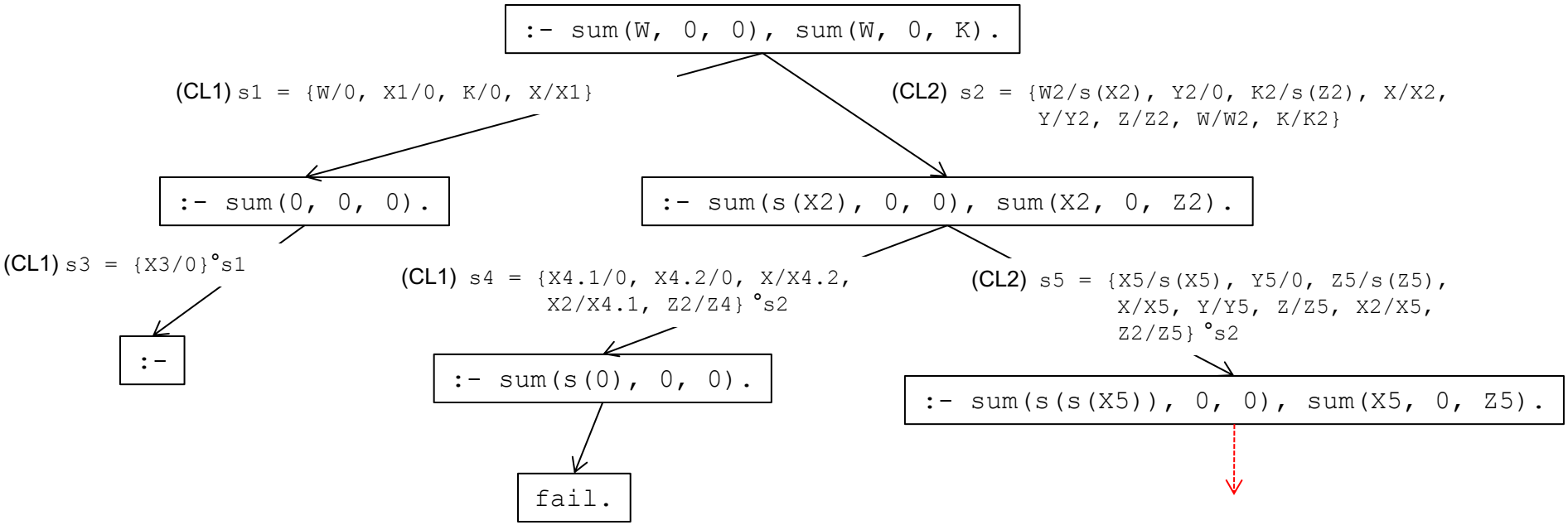
La notazione ° indica la **composizione** di sostituzioni

Stesso esempio, regola diversa

```
sum(0,X,X).                                   (CL1)
sum(s(X), Y, s(Z)):- sum(X, Y ,Z).            (CL2)
```

G_0 = `:- sum(W, 0, 0), sum(W, 0, K).`

Albero SLD con regola di calcolo "right-most"

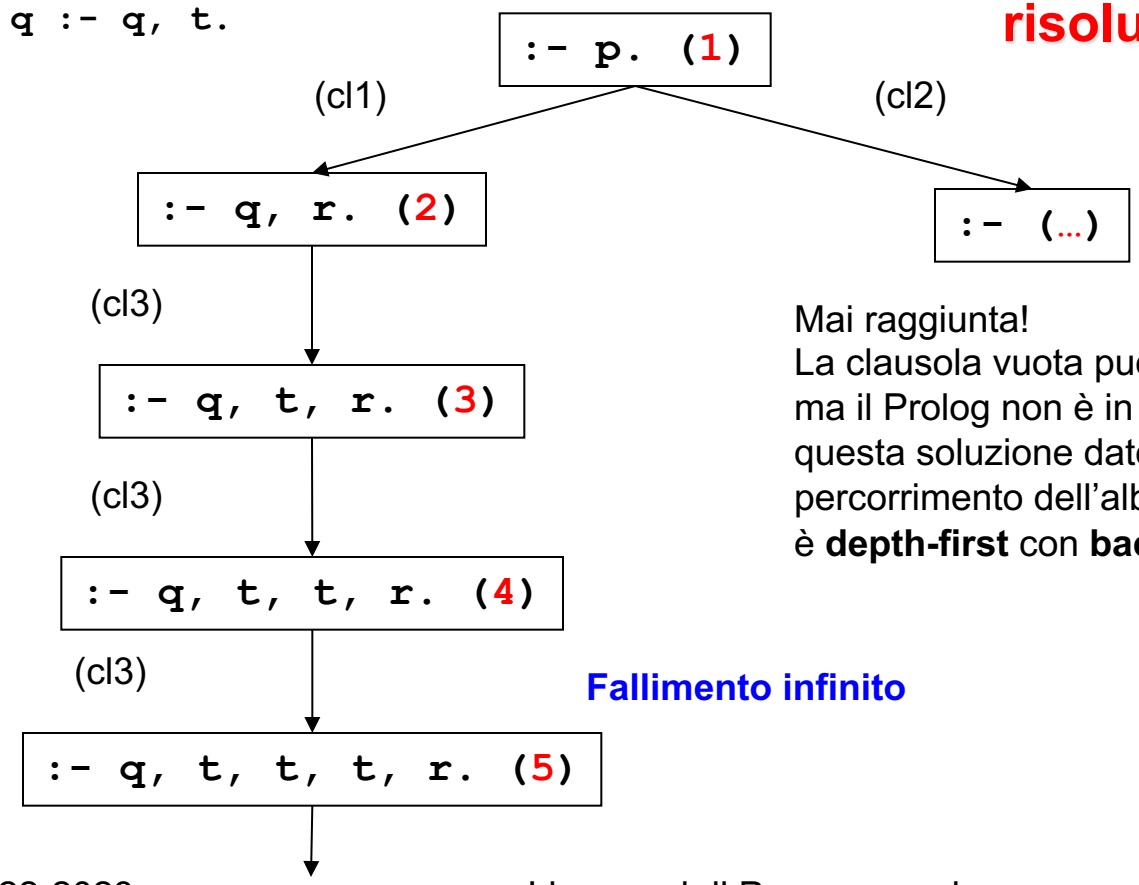


Esempio: fallimento infinito

goal **:- p.**

```
(cl1)    p :- q, r.
(cl2)    p.
(cl3)    q :- q, t.
```

Albero di risoluzione left-most



Mai raggiunta!
La clausola vuota può essere generata ma il Prolog non è in grado di trovare questa soluzione dato che la sua strategia di percorrimento dell'albero (implicito) di soluzioni è **depth-first** con **backtracking**

Fallimento infinito

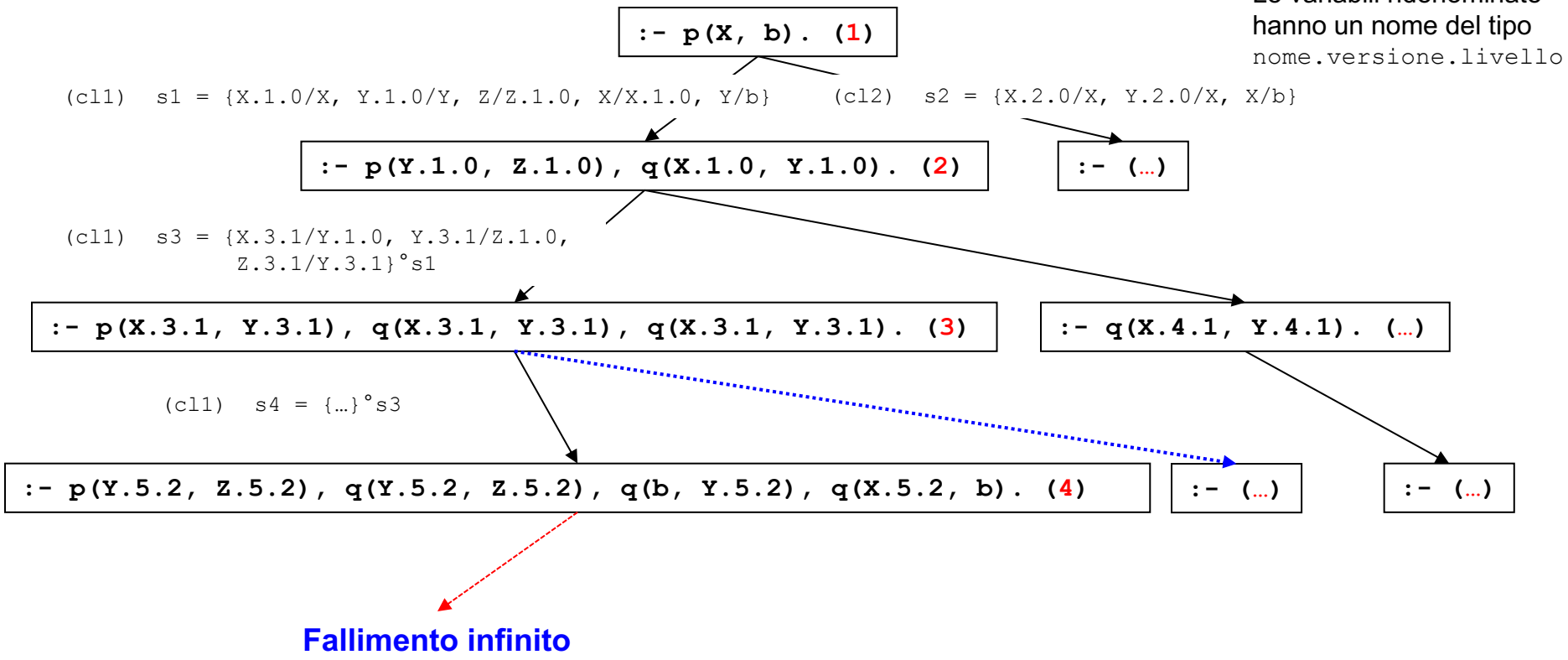
Esempio: fallimento infinito

goal `:- p(X, b).`

```
(cl 1)   p(X, Y) :- p(Y, Z), q(X, Y).
(cl 2)   p(X, X).
(cl 3)   q(a, b).
```

Albero di risoluzione left most

Le variabili ridenominate hanno un nome del tipo `nome.versione.livello`



Fallimento infinito

Sommario

- Relazione tra Calcoli Logici (del Primo Ordine) e Prolog
  - Forma normale
  - Clausole di Horn

- Metodo di *computazione* mediante **prova di teoremi** (*goals*)
  - Regola di inferenza per **Risoluzione**

- Alberi di derivazione (SLD)
  - Un albero SLD rappresenta l'intera computazione potenziale di un sistema di prova di teoremi (date le restrizioni iniziali)
  - Un sistema Prolog attraversa un albero di derivazione **SLD/left-most** in maniera **depth-first** con **backtracking**.