

- Una formula logica  $\varphi$  è generata nel seg. modo:

$$\begin{cases} \varphi \rightarrow T | F | x | \varphi \circ \varphi | \neg \varphi \\ \circ \rightarrow \wedge | \vee | \Rightarrow | \Leftarrow | \end{cases}$$

Notiamo che  $x$  è la base della ricorsione, rappresenta una variabile booleana.

Denotiamo  $\varphi := \dots$

- $\rho : \text{Var} \rightarrow \{T, F\}$  è "l'interpretazione delle variabili di  $\varphi$  secondo  $\rho$ "
- Denotiamo una valutazione come

$\text{Val}(\rho, \varphi)$

Esempio:  $\text{Val}(\rho, T) = T$

$\text{Val}(\rho, x) = \rho(x)$

$\neg$   $\neq \wedge$   
ciò un AND  
semantico

$$\begin{aligned} \text{Val}(\rho, \varphi_1 \wedge \varphi_2) &= \text{Val}(\rho, \varphi_1) \text{ AND } \text{Val}(\rho, \varphi_2) \\ &= \begin{cases} 1, & \text{Val}(\rho, \varphi_1) = \text{Val}(\rho, \varphi_2) = 1 \\ 0, & \text{alt.} \end{cases} \end{aligned}$$

Def. Una formula si dice soddisfacibile se

$$\exists \rho \mid \text{Val}(\rho, \varphi) = T$$

Altrimenti si dice insoddisfacibile.

Una formula si dice tautologia (o valida)

$$\text{se } \forall \rho, \text{Val}(\rho, \varphi) = T$$

Oss. Formule valide  $\subseteq$  Formule soddisfacibili

Oss. Formule valide  $\subseteq$  Formule soddisfacibili

Inoltre  $\varphi$  valida  $\Leftrightarrow \neg\varphi$  insoddisfacibile



Def. Sia  $(\varphi_n)_n$  delle formule.  $(\varphi_n)_n$  è soddisfacibile  
sse  $\exists p \mid \forall_i, \text{Val}(p, \varphi_i) = T$

Exm.  $(\varphi, \neg\varphi)$  è insoddisfacibile

Not. Denotiamo l'insieme delle formule con  $S = \{\varphi_1, \dots, \varphi_n\}$

Un primo metodo per valutare la soddisfabilità / validità di una formula  $\varphi$  è quella di costruire la sua tavola della verità. Tuttavia, se abbiamo  $(x_n)_n$  variabili, allora devo costruire una relazione con  $2^n$  righe (combinazioni). Allora studieremo tecniche più "raffinate" per studiare le formule logiche. #

Prima settiamo uno "standard" per formare le nostre formule

Def. Siano  $\varphi, \varphi'$  delle formule booleane.  $\varphi, \varphi'$  si dicono

**equivalenti** sse  $\forall p, \text{Val}(p, \varphi) = \text{Val}(p, \varphi')$

In tal caso denotiamo  $\varphi \equiv \varphi'$

Thm. Siano  $\varphi, \varphi'$  t.c.  $\varphi \equiv \varphi'$

Allora se  $\exists \psi$  ch contiene  $\varphi$  come sotto formula

Allora se  $\exists \psi$  ch contiene  $\varphi$  come sotto formula  
 ho  $\underline{\Psi_{\varphi \leftarrow \varphi}} = \varphi$   
 $\hookrightarrow$  sostituisco  $\varphi$  con  $\varphi$ )

Prp.  $\neg\neg\varphi \equiv \varphi$

$$\varphi_1 \wedge \varphi_2 = \varphi_2 \wedge \varphi_1 \quad (\text{c.v.})$$

$$(\varphi_1 \wedge \varphi_2) \wedge \varphi_3 = \varphi_1 \wedge (\varphi_2 \wedge \varphi_3)$$

$$\varphi \wedge T \equiv \varphi \quad \varphi \wedge F \equiv F$$

$$\varphi \vee F \equiv \varphi \quad \varphi \vee T \equiv T$$

$$\varphi \Rightarrow \varphi' \equiv \neg\varphi \wedge \varphi'$$

$$\varphi \Leftrightarrow \varphi' \equiv \varphi \Rightarrow \varphi' \wedge \varphi' \Leftarrow \varphi$$

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \equiv (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3) \quad (\wedge \leftrightarrow \vee)$$

$$\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2 \quad (\text{De Morgan}) \quad (\wedge \leftrightarrow \vee)$$

Tlm.  $\forall \varphi$  costruita usando  $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$   
 $\exists \varphi'$  costruita usando  $\wedge, \vee$  t.c.  
 $\varphi \equiv \varphi'$

Def. Un letterale  $l$  è una variabile

booleansa. è positiva se è di forma  $l \in \text{Var}$ , negativa se è di forma  $\neg l \in \text{Var}$ .

Def. Una formula è in **forma normale congiuntiva (CNF)** se è della forma

$$\bigwedge_n (\bigvee_k l_{n,k})_n$$

Ossia se è formata da congiunzioni di disj.

Ossia se è formata da congiunzioni di disgiunzioni di letterali con lo stesso segno.

Not.  $(\bigvee l_n)_n =: C_n$  si dice **clausola**

Exm.  $\neg p \wedge (q \vee r) \wedge (\neg q \vee \neg r) \times$   
 $\neg p \wedge (\neg q \vee \neg r) \wedge (\neg q \vee \neg r) \checkmark$

Th.  $\forall \varphi, \exists \varphi' \text{ CNF} \mid \varphi \equiv \varphi'$

Per verificare la soddisfacciabilità di  $\varphi$  ho 2

modi:

- Semantica: trovo  $p$  che soddisfa  $\varphi$

- Sintattica: manipolo  $\varphi$  sfruttando le equivalenze

MET. (di Tableaux)

Obj. Determinare la <sup>(in-)</sup>soddisfacciabilità di  $\varphi$

Idee. Riduco la struttura di  $\varphi$  in un insieme di letterali Uniti da congiunzioni (se ho  $\vee$  splitto, sfrutto implicazione De Morgan; entrambe le split determinano se  $\varphi$  è soddisf.).

Not. Denotiamo una clausola soddisfacibile con  $\odot$   
" " " non- " con  $\otimes$

Prp. Sia  $\varphi$  formula. Se, usando Tableaux su  $\neg\varphi$ , ho:

- $\forall \otimes$ :  $\varphi$  è valida
- $\exists \odot$ :  $\varphi$  non è valida ma è soddisfacibile

Oss. Usando  $\varphi \equiv \neg(\neg(\varphi))$  ho un thm analogo che garantisce l'invalidità di una formula

Exm.  $\varphi := (p \vee q) \wedge (\neg p) \wedge (\neg q)$

Exm.  $\varphi := (p \vee q) \wedge (\neg p) \wedge (\neg q)$

Ho  $S = \{ (p \vee q), \neg p, \neg q \}$

Ossia due ramificazioni

$$\{ p \vee q, \neg p, \neg q \}$$



$$\{ p, \neg p, \neg q \}$$

$$\{ q, \neg p, \neg q \}$$



$$\{ F, \neg q \}$$

$$\{ F, \neg p \}$$



$\otimes$



$\otimes$

Ho che  $\neg \varphi$  è valida, ossia  $\varphi$  non è valida.

Prf. Proponiamo un paio di "regole" per fare gli split

a) (base)  $\{ \varphi_1 \wedge \varphi_2 \}$

$$\downarrow$$

$$\{ \varphi_1, \varphi_2 \}$$

b)  $\{ \varphi_1 \vee \varphi_2 \}$

$$\swarrow \quad \searrow$$

$$\{ \varphi_1 \} \quad \{ \varphi_2 \}$$

c)  $\{ \varphi_1 \Rightarrow \varphi_2 \}$  ( $\neg \varphi_1 \Rightarrow \varphi_2 \equiv \neg \varphi_1 \vee \varphi_2$ )

$$\swarrow \quad \searrow$$

$$\neg \varphi_1 \quad \varphi_2$$

d)  $\{ \neg(\varphi_1 \wedge \varphi_2) \}$

$$\swarrow \quad \searrow$$

$$\{ \neg \varphi_1 \} \quad \{ \neg \varphi_2 \}$$

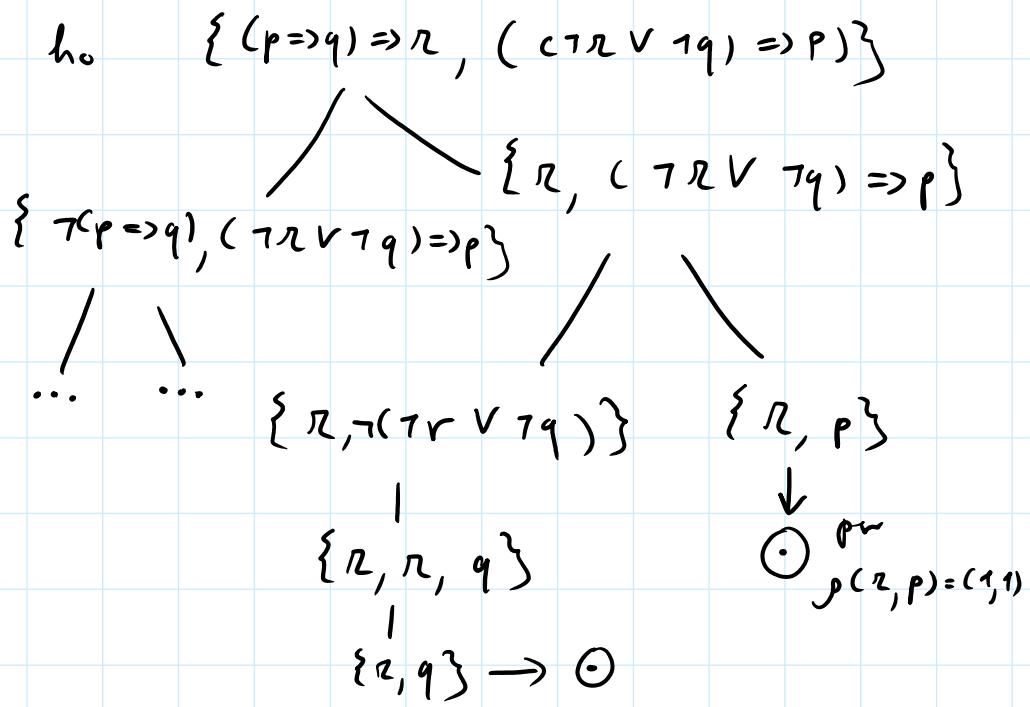
e)  $\{ \neg(\varphi_1 \vee \varphi_2) \}$

$$\downarrow$$

$$\{ \neg \varphi_1, \neg \varphi_2 \}$$

Exr.  $((p \Rightarrow q) \Rightarrow r) \wedge ((\neg r \vee \neg q) \Rightarrow p)$

Lxr.  $\neg(p \Rightarrow q) \rightarrow (\neg p \wedge (\neg q \vee q)) \Rightarrow p$



Allora  $\Psi$  è soddisfacibile

Tableaux Semantico → Determina la validità/invalidità di  $\varphi$   
 Vediamo un altro algoritmo x la soddisficiabilità di  $\varphi$

### MET. (Refutazione)

Sia  $\varphi$  una CNF insoddisfacibile; voglio trovare una "prima" che è insoddisfacibile.

Prima vediamo la nozione di risoluzione.

Def. Siano  $C_1, C_2$  due clausole col letterale in comune  $\ell$  (una deve avere  $\neg\ell$ , l'altra  $\ell$ )

Si dice risoluto di  $C_1, C_2$  la seg. clausola:

$$Res(C_1, C_2) = \underline{C_1 \setminus \{\ell\}} \cup \underline{C_2 \setminus \{\neg\ell\}}$$

Esemp.  $C_1 = \{p, q, \neg r\}, C_2 = \{\neg p, s, \neg t\}$

Allora  $Res(C_1, C_2) = \{q, \neg r, s, \neg t\}$

Prf.  $C_1 \wedge C_2 \Rightarrow Res(C_1, C_2)$

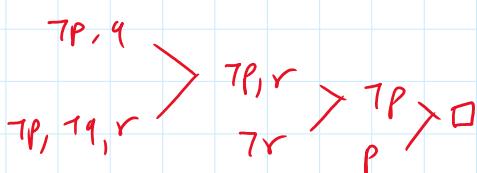
Torniamo alla refutazione:

- Sia  $S$  una CNF
- Loop: (finché non ho tutti i risolti  $\circ \square$ )
  - Scelgo 2 clausole con 2 letterali contrapposti
  - Calcolo il risoluto, ripeto

Ho ch.:

- C'è refutazione se termina con almeno 1  $\square$
- Non c'è refutazione se terminando non ho  $\square$

Esemp.  $S = \{\{p\}, \{\neg p, q\}, \{\neg r\}, \{\neg p, \neg q, r\}\}$



ho  $\square$ , per cui ci sono delle refutazioni

### MET. (Algoritmo di D-P)

Idee. Se  $S \neq \emptyset$  e  $\square \notin S$ , itero:

- Scelgo pivot  $x$  (preferibilmente appena nelle clausole più piccole)
- Calcolo i risoluti, rimuovo tautologie e sussunzioni ( $C \subseteq C'$ )

C. d.  $S = \emptyset$   $\Leftrightarrow \neg \varphi$

- Calcolo x-ri solventi, rimuovo tautologie e sussseguimi:  $C \subset E(C)$

$S \subseteq \Omega$  e  $S = \emptyset$ , allora è soddisfacibile  
 $S \subseteq \Omega$  e  $S \neq \emptyset$ , allora è insoddisfacibile

$$\{ \{A, B, C, D, \neg E\}, \{C, \neg B\}, \{D, \neg A\}, \{B\}, \{C, A, \neg E\}, \{\neg D, \neg A, \cancel{\neg B}, \neg C, D, E\}, \\ \{ \cancel{\neg C}, \cancel{\neg D} \}, \{\neg D, \neg A, B\}, \{E, \neg C, D\}, \cancel{\{D, \neg A\}}, \cancel{\{C, \neg E\}} \} \rightarrow \text{tautologia}$$

$$\underline{\text{Exv.}} \quad \{\{\underline{E}, A\}, \{-B, C\}, \{-A, B\}, \{A, B\}, \{-C, -D\}, \{-E, -A, -C, D\}\}$$

1. Non ho le sussunzioni in lotkali: scelgo A pivot

$$\text{Ris} = \{\{E, B\}, \{B, \neg C, D, \neg E\}, \{B\}, \{B\}, \{\neg(C, D, E, \neg E)\} \}$$

$$= \{\{E, B\}, \{B, \neg C, D, \neg E\}, \{B\}\}$$

Ahaa h.

$$\{\{ \neg B, C\}, \{\neg C, \neg D\}, \{B\}, \{E, B\}, \{B, \neg C, D, E\}\}$$

2.  $B$  unitaria  $\Rightarrow$  Assumo  $B = T$

$\{\{C\}, \{\neg C, \neg D\}\}$

$$3. C \text{ unitaria} \Rightarrow A_{\text{sim}} = C = I$$

{ { - D } }

$$4. D \text{ unitaria} \Rightarrow A_{symm}, D = T$$

$\{ \} = \emptyset$  Ho  $S$  soddisfacibile per

$$\rho(A, B, C, D, E) = (., T, T, T, .)$$

$$\{\{\neg B, C\}, \{\neg C, \neg D\}, \{B\}, \{E, B\}, \{B, \neg C, D, E\}\}$$

## 2. B Pivot

3 (Pixel)

$\{\cancel{C, D}, \{D, E\}\}$

4. D, E pivot  
 $\{ \} - \textcircled{X}$

## Algoritmo DPLL

Sia  $S$  un insieme di classi e sia

$\vdash$  un'interpretazione parziale di S.

Sí a ésta es una clausula.

$V_F(c) = T$  allora  $F$  soddisfa

clausola conflictus x F

Exm. Sıca  $S = \{ \{p, q, r\}, \{\neg p, q\}, \{\neg q, \neg r\}, \{\neg r\} \}$   
 Pseudonam  $\mathcal{F}$  t.c.

$$\overline{f}_{\gamma_1, r}^{(c_9)} = F \leftarrow \overline{f}_{\gamma_9, r} (r) = T$$

## Notions de

$$T = \langle \{r_1, r_2\} \rangle = T$$

Notiamo che

$$\overline{F}_{\neg q, r}(\{\neg p, q\}) = T$$

$$\overline{F}_{q, r}(\{\neg q, \neg r\}) = T$$

$$\overline{F}_{\neg q, r}(\{\neg r\}) = T$$

Però

$\overline{F}_{\neg q, r}(\{\neg p, q\})$  non può essere vera  
senza assegnare un valore di verità

a  $p$ . Pertanto  $\overline{F}_{\neg q, r}$  è in conflitto con  
 $\{\neg p, q\}$

Prendendo  $\overline{F}_r(r) := F$  si ha che

$\overline{F}_r$  entra in conflitto con  $\{\neg r\}$

Chiede l'estensione di  $\overline{F}_r$  che soddisfi  $S$

Idea: Prendere interpretazioni parziali, e fonderle  
ad un valore atomico senza valori e  
valutarlo (BRANCHING). Se ad un certo punto  
non ci sono estensioni soddisfacibili: mi  
arreto. ↳ L'interpretazione entra  
in conflitto

L'algoritmo è attualmente non deterministico x la  
scelta del valore da assegnare dell'atomo

Esemp.  $\{\{p, q, r\}, \{\neg p, q\}, \{\neg q, \neg r\}, \{r\}\}$

Branch su  $r$ :  $\overline{F}_r, \overline{F}_{\neg r}$

$\overline{F}_{\neg r}$  chiude, conflitto con  $\{r\}$

$\overline{F}_r$ : per soddisfare  $\{\neg q, \neg r\}$  devo  
prendere  $\overline{F}_{r, \neg q}$  quindi faccio  
branch

$\overline{F}_{r, \neg q}$ : faccio branch su  $p$

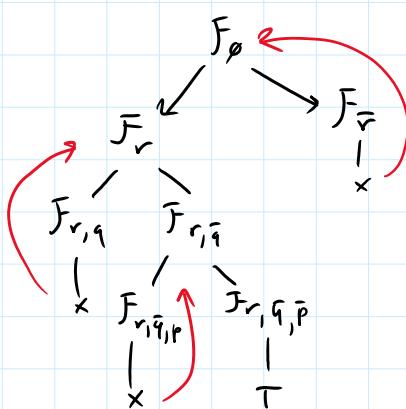
$\overline{F}_{r, \neg q, p}$ : Conflitto con  $\{\neg p, q\}$

$\overline{F}_{r, \neg q, \neg p}$ : Nessun conflitto e sono  
tutti soddisfatti ✓

Una soluzione soddisfacibile è

tutti soddisfatti ✓

Una soluzione soddisfacibile è



A handwritten diagram illustrating a proof tree for the formula  $\{P, Q, R\} \vdash \{\neg P, Q\}, \{\neg Q, \neg R\}, \{R\}$ . The tree is shown as a network of nodes connected by arrows, with some nodes labeled with their truth values.

- Root Node:**  $\{P, Q, R\}$
- Left Branch (Path 1):**  $\neg P, Q$  (labeled  $\top$ )
- Right Branch (Path 2):**  $\neg Q, \neg R$  (labeled  $\top$ )
- Bottom Branch (Path 3):**  $R$  (labeled  $\top$ )
- Bottom Level Labels:**
  - $\neg P = F$  (under  $\neg P, Q$ )
  - $\neg Q = F$  (under  $\neg Q, \neg R$ )
  - $Q = F$  (under  $R$ )
  - $\emptyset$  (an empty set symbol, indicating the tree is closed)

Possiamo rendere DPLL più "veloce" con delle variazioni, in particolare × ottimizzarne la unit propagation. In particolare vorremo un algoritmo iterativo basato su un ricorsivo

Uso delle "eristiche" di branding x determinare  
in una maniera SISTEMATICA lo branding e  
l'assegnazione dei valori.

Flö 2 euristisch principdi:

- Frequenza: Fa riferimento sull'attore che compie più frequentemente in S, così l'insieme delle classi che si "riduce velocemente"

$$\begin{array}{ccccccccc} \bar{P}, q, \triangleright & P, \bar{q}, \triangleright & P, q, \bar{\triangleright} & \bar{P}, \bar{q}, \bar{\triangleright} & \bar{P}, t, \vee & P, \bar{t}, \bar{\vee} \\ \downarrow p=1 & & & & & & \\ q, \triangleright & \bar{q}, \bar{\triangleright} & t, \vee & \bar{t}, \bar{\vee} & & & \end{array}$$

- Dimension: Scelgo il lettore da appan nella classe più piccola; così ottimizzo la unit propagation

## la unit propagation

Un altro metodo è quello di fare la "non chronological" backtracking, i.e. quando ho insoddisfabilità torno al nodo genitore.

X

## Algoritmi Stocastici

Un altro algoritmo per verificare la soddisfabilità è quello di introdurre elementi di casualità.

Si mostra che hanno un'altra probabilità di ritrovare soluzioni corrette.

Viceversa solo per trovare modelli, in quanto non può dichiarare che una formula è insoddisfacibile (ci metterebbe tempo...)

Idee. Prendo  $F$  casuale e itero su:

- Se  $F(C) = 1$  concluso
- Se  $F(C) = 0$ :
  - scelgo atomo p corrente e lo flippo in  $F$ , rivaluto

X

## Formati Speciali di Clausi

Domin. Certe clausole con certe caratteristiche possono rendere certi algoritmi più veloci ("fulminei").

Dif. Una clausola  $C$  con al più 2 letterali si dice clausola di Krom

Dif. Una clausola  $C$  con al più 2 letterali  
si dice clausola di Horn

Le clausole di Horn ci garantiscono, con la DP,  
che il calcolo dei nuclei risolventi non cada in un  
"esplosione" di ulteriori clausole

Prp. Una CNF  $C$  con tutte le clausole di Horn  
con  $n$  variabili, è t.c. che ogni iterazione <sup>tutta</sup> di DP  
avrà al più

$$n := 2(n-t)^2 + n - t + 1$$

clausole.

Proof. (idea)

Al tempo  $t$ -esimo ho al più  $n-t$  variabili.

Allora  $S_t$  può assumere una delle seguenti forme:

- Clausole vuote
- $2(n-t)$  clausole con un letterale
- $2(n-t)(2(n-t)-1)/2$  clausole, con  
ognuna 2 letterali

Siccome il risolvente di 2 clausole di Horn è  
un'unica clausola di Horn, abbiamo che sommiamo

$$2(n-t)^2 + n - t + 1$$

clausole  $\#$

Cor. DDP con Horn è in P

Dif. Una clausola  $C$  è di Horn se  
contiene al più un letterale positivo

Esemp.  $\square$  (clausola vuota)  
 $\{A\}$   
 $\{\neg A, \neg B, C\}$

{A}

{ $\neg A$ ,  $\neg B$ , C}

{ $\neg A$ ,  $\neg B$ }

Notiamo che  $\neg A \vee \neg B \vee C \equiv (\neg A \wedge \neg B) \Rightarrow C$ . Quindi

se A, B sono vere allora lo sarà C; invece

se C è falsa allora A, B lo saranno pur.

Prp. Sia  $S_0$  un insieme di classi di  $\mathcal{M}_{\text{DN}}$ ,

senza tautologie e sussintini (WLOG)

Allora d'ultimo passo di DP ho:

- Se  $D \in S_{t-1} \Rightarrow S$  è insoddisfacibile
- Se  $S_{t-1}$  non contiene  $\neg$  clauses unitarie  $\Rightarrow S$  è soddisfacibile
- Se  $S_{t-1}$  contiene una clause unitaria,  
allora  $S_t$  è l'insieme di classi  
di  $\mathcal{M}_{\text{DN}}$  con meno classi di  $S_{t-1}$

Problemi di logica → Devo decidere se una formula  $C$  in CNF (wlog) è soddisfacibile o valida.

<u>Algoritmo</u>	Soddisfabilità	Validità	Tempo
Tabelle di verità	✓	✓	$2^n$
Tableaux Sémantico	✓	✓	NP
DDP	✓	✗	NP
Risoluzione	✓ (in-)	✓	NP
Stoccati (SAT)	✓ (sol. sub.)	✗	NP
DP LL	✓	✗	NP

Si dimostra che SAT è un problema NP-hard (Cook-Levin)

NP-hard (Cook- Levin)

## FOL: Intro

La logica dei predici ci permette di dire se certe formule formate da congiunzioni e disgiunzioni sia vere o no.

Tuttavia questo non basta; in certi casi i predici sono interpretati sotto una relazione di un dominio  $D$ .

Ad esempio,  $P(x) \in \mathbb{N}$  ci indica  $x \in \mathbb{N}$  è un numero primo o no.

Quindi studiamo la logica dei predici alla logica del primo ordine, per poter scrivere e manipolare formule logiche che coinvolgono relazioni su valori di insiemi (domini) arbitrari.

Sia  $R \subset D^n$  una relazione relazionale su  $D$ .

Allora essa ha una rappresentazione logica data da  $P_R$ , in

$$P_R(d_1, \dots, d_n) = T \Leftrightarrow (d_1, \dots, d_n) \in R$$

Vediamo di sviluppare la sintassi della FOL

- $\wp$ : simboli di predici
- $A$ : simboli costanti
- $V$ : simboli di variabili

**Formule atomiche**: Per ogni simbolo  $p \in \wp$  lo associamo con un numero  $n \geq 1$  di "argomenti" da prendere.

Argomenti

- Un argomento è una variabile  $x \in V$  o costante  $a \in A$

**Composizione  
di formule**

- Una formula è composta da più formule, <sup>con lo stesso</sup>  $n$  in particolare possono:
- $p_1 \wedge p_2$
- $p_1 \vee p_2$
- $\neg p_1$
- $\forall x \in V \ p(x)$
- $\exists x \in V \ p(x)$

Esempi:

$$\forall x \forall y (p(x, y) \Rightarrow p(y, x)) \text{ è FOL}$$

$$\forall x \ p(a, x)$$

Come nella programmazione strutturata "a blocchi", ogni variabile può avere una sua "portata" associata ad una formula.

Una formula si dice quantificata se ho un  $\forall$  o  $\exists$ , e le sue variabili associate sono le "variabili quantificate" e il loro scopo è la formule.

$$\forall x (p(x))$$


Notiamo che non è necessaria che appaia  $x$  nella portata di quantificazione.

Esempio.  $\forall x p(x, y) \rightarrow y$  è comunque libera!

- Una variabile si dice libera nella formula  $A$  se  $x$  non è nella portata di una variabile quantificata.
- Altrimenti si dice VINCOLATA
- Se una formula non ha variabili libere, si dice CHIUSA.
- Date  $(x_1, \dots, x_n)$  variabili su  $A$

- Date  $(x_1, \dots, x_n)$  variabili su A  
diciamo:
  - la chiusura universale  
 $\forall_{x_1} \forall_{x_2} \dots \forall_{x_n} A$
  - la chiusura esistenziale  
 $\exists_{x_1}, \dots, \exists_{x_n} A$
- Diciamo  $A(x_1, \dots, x_n)$  che l'istanza delle var. libere di A sta in un Sottosistema di  $\{x_1, \dots, x_n\}$

Oss. Una variabile può aver occorrenze sia vincolate che libere in una stessa formula  
 $(\forall x A(x) \wedge B(x))$

Q. Come possiamo, data una formula generica A, farci dei calcoli e valutare se è vera o falsa?

Logica Proposizionale  $\Rightarrow$  Sostituiamo le variabili con, dai

variabili con dei valori arbitrari e mappatura ai valori diversi)

FOL  $\Rightarrow$  ANCORA PRIMA dobbiamo definire il dominio  $\mathcal{D}$  di cui verranno assegnati le costanti e le variabili. Poi interpretiamo i predici: come delle relazioni su  $\mathcal{D}$ .

Quindi data una  $A \sim ((p_m)_m, (a_n)_n)$   
un'interpretazione

$\xrightarrow{\text{predici}}$ ,  
 $\xrightarrow{\text{costanti}}$

$\mathcal{F}_A$  è una tripla

$(\mathcal{D}, (R_m)_m, (d_n)_n)$

dove  $\mathcal{D}$  è un insieme,  
 $R_i$  è una relazione  $n_i$ -aria del predicato  $p_i$ :

di  $\in \mathcal{D}$

E.xm. Sia  $A = \forall x p(a, x)$

$d_1$

$p_1$

$\mathcal{F}_1 = (\mathbb{N}, \{\leq\}, \{0\})$

$$\mathcal{F}_1 = (\mathbb{N}, \{\leq\}, \{0\})$$

$$\mathcal{F}_2 = (\mathbb{N}, \{<\}, \{1\})$$

1:  $\forall x \in \mathbb{N}, \quad t \leq x$

2:  $\forall x \in \mathbb{N}, \quad 0 \leq x$

$$\mathcal{F}_3 = (\{\star\}, \{\leq\}, \{\varepsilon\})$$

3:  $\forall x \in \Sigma^*, \quad \varepsilon \leq x$

(" $\leq$ "  $\geq$  inteso nel senso lessicografico)

Se una formula non è chiusa, l'interpretazione  $\mathcal{F}$  non è sufficiente per determinare i valori di verità di  $A$ , in quanto è ancora dipendente da ulteriori assegnamenti di valori di verità.

$\{x : p(x, a) \text{ in } \mathcal{F} = (\mathbb{N}, \{\leq\}, \{\varepsilon\})$  è incompleto!

Denotiamo  $\sigma_{\mathcal{F}_A} : V \rightarrow D$  come l'assegno della variabile relativa ad un'interpretazione

della variabile relativa ad un'interpretazione  $\mathcal{F}_A$ .

Ese.  $p(x, a)$  nel  $\mathcal{F} = (\mathbb{N}, \{\leq\}, \{(3)\})$  è incompleto!

$$\rightarrow \sigma_{\mathcal{F}}(x) = 3 \rightarrow \text{completo}$$

Oriale si può mappare alle altre costanti:

$$\sigma_{\mathcal{F}}(x) = a$$

Quindi, data  $\mathcal{F}_A$  e  $\sigma_{\mathcal{F}_A}$  definiamo

$V_{\sigma_{\mathcal{F}_A}}(A)$ : i valori di verità di  $\mathcal{F}_A$   
per induzione strutturale.

- Let  $A = p_k(c_1, \dots, c_n)$  be an atomic formula where each  $c_i$  is either a variable  $x_i$  or a constant  $a_i$ .  $v_\sigma(A) = T$  iff  $(d_1, \dots, d_n) \in R_k$  where  $R_k$  is the relation assigned by  $\mathcal{I}_A$  to  $p_k$ , and  $d_i$  is the domain element assigned to  $c_i$ , either by  $\mathcal{I}_A$  if  $c_i$  is a constant or by  $\sigma_{\mathcal{I}_A}$  if  $c_i$  is a variable.
- $v_\sigma(\neg A_1) = T$  iff  $v_\sigma(A_1) = F$ .
- $v_\sigma(A_1 \vee A_2) = T$  iff  $v_\sigma(A_1) = T$  or  $v_\sigma(A_2) = T$ , and similarly for the other Boolean operators.
- $v_\sigma(\forall x A_1) = T$  iff  $v_{\sigma[x \leftarrow d]}(A_1) = T$  for all  $d \in D$ .
- $v_\sigma(\exists x A_1) = T$  iff  $v_{\sigma[x \leftarrow d]}(A_1) = T$  for some  $d \in D$ .

x

Validità e Soddisfabilità

Per semplicità consideriamo soltanto le FORMULE CHIUSE, così da non dover dipendere sull'assegno  $\sigma$ .

Infatti:

$$V_{\sigma_F}(A) = T \Leftrightarrow \exists(x_1, \dots, x_n) \mid V_F(\exists(x_1, \dots, x_n)) = T$$

$$V_{\sigma_F}(A) = T \Leftrightarrow \forall(x_1, \dots, x_n) \mid V_F(\forall(x_1, \dots, x_n)) = T$$

Quindi ci riconduciamo sempre alle chiusure di una formula.

Diamo delle def.:

- $A$  è vera in  $F$  o se  $V_F(A) = T$   
 $\bar{F}$  è un modello  
in  $A$
- Not.:  $\bar{F} \models A$
- $A$  è valida se  $\forall F, \bar{F} \models A$ 
  - Not.  $\models A$
- $A$  è soddisfacibile se  $\exists F, \bar{F} \models A$ 
  - Altrimenti insoddisfacibile

- Altrimenti insoddisfacibile
- $A$  è falsificabile  $\rightarrow$  se non è valida

Exm:

$\forall x (p(x)) \Rightarrow p(a)$  è valida

$\forall x \forall y (p(x,y) \Rightarrow p(y,x))$  è soddisfacibile  
sotto relazioni di equivalenza

## Proprietà delle FOL.

Vediamo un po' di proprietà che possono avere delle formule FOL:

- Date  $A_1, A_2$  formule chiuse, si dicono equivalenti se  $\mathcal{V}\mathcal{F}_{\{A_1, A_2\}} = \mathcal{F}$ ,  $\mathcal{V}\mathcal{F}(A_1) = \mathcal{V}\mathcal{F}(A_2)$

In tal caso scriviamo

$$A_1 \equiv A_2$$

- Sia  $A$  una formula chiusa e  $\mathcal{U}$  un insieme di formule chiuse.

Se  $\mathcal{V}\mathcal{F}, \mathcal{V}A_i \in \mathcal{U}$ ,

$$\mathcal{V}\mathcal{F}(A_i) \Rightarrow \mathcal{V}\mathcal{F}(A)$$

Allora si dice che  $A$  è una conseguenza logica di  $\mathcal{U}$

- Nat.  $\mathcal{U} \models A$

Prp.  $A \equiv B$  se  $A \Leftrightarrow B$

$\mathcal{U} \models A$  se  $(A_1, \dots, A_n) \Rightarrow A$

## Prp. (dualità)

$$\forall x A(x) \equiv \neg \exists x \neg A(x)$$

$$\exists x A(x) \equiv \neg \forall x \neg A(x)$$

Con un abuso di notazione,  $\exists \sim \neg \forall$

## Prp. (commutatività)

### Prop. Commutatività

$$\forall x \forall y A \Leftrightarrow \forall y \forall x A$$

$$\exists^* \exists^* \Leftrightarrow \exists^* \exists^*$$

$$\vdash \exists x \forall y A(x, y) \Rightarrow \forall y \exists x A(x, y)$$

### Prop. Distributività

$$\forall x (A(x) \wedge B(x)) \equiv \forall x A(x) \wedge \forall x B(x)$$

$$\exists x (A(x) \vee B(x)) \equiv \exists x A(x) \vee \exists x B(x)$$

ma

$$\vdash \forall x (A(x) \vee B(x)) \Rightarrow \forall x A(x) \vee \forall x B(x)$$

$$\vdash \exists x (A(x) \wedge B(x)) \Rightarrow \exists x A(x) \wedge \exists x B(x)$$

### Esempio

$$\exists x (A(x) \Rightarrow B(x)) \Leftrightarrow \exists x (\neg A(x) \vee B(x))$$

$$\Leftrightarrow \exists x \neg A(x) \vee \exists x B(x)$$

$$\Leftrightarrow \neg \underline{\exists x \neg A(x)} \vee \exists x B(x)$$

$$\Leftrightarrow \neg \exists x \neg A(x) \Rightarrow \exists x B(x)$$

$$\Leftrightarrow \forall x A(x) \Rightarrow \exists x B(x)$$

### TABELLA SEMANTICO

Idee. Stessa x la logica proposizionale,  
ma aggiungo una regola in più:  
associato ai quantificatori  $\forall, \exists$ )

Ripetuta δ:

## Regola $\delta$ :

$$\exists x A(x) \xrightarrow{\delta} A(a)$$

$$\neg \forall x A(x) \xrightarrow{\delta} \neg A(a)$$

## Regola $\sigma$ :

$$\forall x A(x) \xrightarrow{\sigma} \forall x A(x), A(a)$$

$$\neg \exists x A(x) \xrightarrow{\sigma} \quad " \quad "$$

Ovvvero vado ad istanziare la formula  
A con un simbolo costante  $a \in A$ .

- Per  $\exists$  introduco (derivo) una costante mai usata
- Per  $\forall$  introduco quanti termini necessari (quindi tengo bruna la formula originale)

## E.x.m.

$$\neg (\forall x (p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x)))$$



$$\forall x (p(x) \Rightarrow q(x)), \neg (\forall x p(x) \Rightarrow \forall x q(x))$$



$$\forall x (p(x) \Rightarrow q(x)), \forall x p(x), \neg \forall x q(x)$$



$$\forall x (p(x) \Rightarrow q(x)), \forall x p(x), \exists x \neg q(x)$$



$$", ", \neg q(a) \xrightarrow{\delta} a \in A \text{ f.c.}$$

$$\text{", ", } \neg q(a) \rightarrow a \in A \text{ f.c.} \\ \downarrow \sigma \qquad \qquad \qquad q(a) = F \Rightarrow \neg q(a) < T$$

$$\forall x(p(x) \rightarrow q(x)), \forall x p(x), \underset{\downarrow \sigma}{p(a)}, \neg q(a)$$

$$\forall x(p(x) \rightarrow q(x)), \forall x p(x), \underset{\downarrow \sigma}{p(a) \rightarrow q(a)}, p(a), \neg q(a)$$

Tuttavia  $p(a) \rightarrow q(a) \Leftrightarrow p(a) \wedge \neg q(a)$ ,  
prorante da la formula è INVALIDA.

Per formule invalidi si tablcaux può  
concludere, ma per quelle soddisfacenti?

$$\begin{aligned} \forall x \exists y p(x, y) \\ \downarrow \sigma \\ \forall x \exists y p(x, y), \exists y p(x_0, y) \\ \downarrow \delta \\ \forall x \exists y p(x, y), \underset{\text{red}}{p(x_0, x_1)} \end{aligned}$$

Per costruzione  $p(x_0, x_1)$  ma è vera  
ma rimane cmq. la FORMULA originale;  
così ho dei BRANCH INFINITI se A

non è un insieme finito (al più numerabile).

$\hookrightarrow$  Non si può sapere a priori se

↳ Non si può sapere a priori se il tableau terminerà o meno (Halting Problem)

Tuttavia, se la formula fosse interpretata in un dominio finito, allora potrebbe essere soddisfacibile.

Thm. Sia  $\varphi$  una formula FOL. Se  $\mathcal{T}(\varphi)$  chiude, allora  $\varphi$  è insoddisfacibile.

Thm. Sia  $\varphi$  una formula valida. Allora  $\mathcal{T}(\neg\varphi)$  diude.

---

### Prenex - CNF

Obiettivo: estendere la forma standard della logica dei predicati sulla logica FOL

Dichiamo che una formula è **PCNF** se è di forma

$$Q_1 x_1 \dots Q_n x_n M$$

- $Q_i x_i$  è una quantificazione su  $x_i$  ( $Q = \exists, \forall$ )
- $M$  è una formula priva di quantificatori,

• M è una formula priva di quantificatori,  
Si dice matrice

Così possiamo associare a  $\varphi$  una PCNF ad

un insieme di clausole  $(A) \rightarrow$  a condizione che  
tutte le Q siano t

Esempio.  $\forall y \forall z ((p(f(y)) \vee \neg p(g(z))) \wedge q(y))$

$$\downarrow \\ \{\{p(f(y)), \neg p(g(z))\}, \{q(y)\}\}$$

Al contrario delle CNF, non è garantito che

$$\forall \varphi \exists \bar{\varphi}_{PCNF} \mid \varphi \equiv \bar{\varphi}$$

Infatti è possibile che siano "ugualmente soddisfribili",

ovvero  $\varphi \approx \bar{\varphi}$  (thm. di Skolem)

## Skolemizzazione.

$$\text{PART 1: } \varphi \rightarrow \varphi_{PCNF}$$

$\downarrow$

Procedura x trasformare  
una formula FOL qualsiasi  
in una sua PCNF  
equisoddisfacibile

Idea:

$$\forall x (p(x) \Rightarrow q(x)) \Rightarrow (\forall x p(x) \Rightarrow \forall x q(x))$$

- Andiamo a rinominare le variabili limitate  
con nomi diversi, così non appaiono in più

con nomi diversi, così non appaiono in più quantificatori diversi

$$\forall x_1 (p(x_1) \Rightarrow q(x_1)) \Rightarrow (\forall x_2 p(x_2) \Rightarrow \forall x_3 q(x_3))$$

- Riconduciamo tutte le op. booleans a  $\vee, \wedge$ :

$$\neg \forall x_1 (\neg p(x_1) \vee q(x_1)) \vee \neg \forall x_2 p(x_2) \vee \forall x_3 q(x_3)$$

- Portiamo "dentro" tutte le negazioni, causando doppie negazioni, finché si applichino solo a formule atomiche

$$\exists x_1 (p(x_1) \wedge \neg q(x_1)) \vee \exists x_2 \neg p(x_2) \vee \forall x_3 q(x_3)$$

- Estrarre i quantificatori dalla matrice, e considerarli come la "più esterna", usando le seg. equivalenze:  $A \circ Q_x B(x) \equiv Q_x(A \circ B(x))$   
 $Q_x A(x) \circ B \equiv Q_x(A(x) \circ B)$

$$\exists x_1 (p(x_1) \wedge \neg q(x_1)) \vee \exists x_2 \neg p(x_2) \vee \forall x_3 q(x_3)$$



$$\exists x_1 ((p(x_1) \wedge \neg q(x_1)) \vee \exists x_2 \neg p(x_2) \vee \forall x_3 q(x_3))$$



...



$$\exists x_1 \exists x_2 \forall x_3 ((p(x_1) \wedge \neg q(x_1)) \vee \neg p(x_2) \vee q(x_3))$$

$$\exists x_1 \exists x_2 \forall x_3 ((p(x_1) \wedge \neg q(x_1)) \vee (\neg p(x_2) \vee q(x_3)))$$

Alla fine trasformo la matrice in una CNF:

$$((p(x_1) \vee \neg p(x_2) \vee q(x_3)) \wedge (\neg q(x_1) \vee \neg p(x_2) \vee q(x_3)))$$

Concludendo.  $\square$

Notiamo che in questo caso ci siano pure garantiti "≡".

Adesso manca l'ultima parte, ovvero di trasformare una PCNF in una forma clausale; essa richiede che tutti quantificatori siano  $\forall$ , ma come trattano gli  $\exists$ ?

$$\text{PART}_2: \Phi_{\text{PCNF}} \rightarrow \Phi$$

Idea. Per ogni quantificatore universale  $\exists x A(x)$ , definiamo  $y_1, \dots, y_n$  le variabili universalmente quantificate da la precedono, e definiamo  $f^n(y_1, \dots, y_n)$ . Allora rimoviamo  $\exists x$  e la sostituiamo con  $f^n(y_1, \dots, y_n)$ .

$r_n$

$r$

$c_1 c_2$

Sostituiamo con  $T(y_1, \dots, y_n)$ .

$f_i^n$  si dice **funzione di Skolem**

e tale processo si dice **Skolemizzazione**

Oss. Se  $n=0$  allora usiamo una costante  
 $a \in A$  che è una funzione 0-aria.

$$\exists x_1 \exists x_2 \forall x_3 ((p(x_1) \vee \neg p(x_2) \vee q(x_3)) \wedge (\neg q(x_1) \vee \neg p(x_2) \vee q(x_3)))$$

$\overbrace{\quad}^{n=0}$       ↓

$$\forall x_3 ((p(a_1)) \vee \neg p(a_2) \vee q(x_3)) \wedge (\neg q(a_1) \vee \neg p(a_2) \vee q(x_3))$$



Oss. La Skolemizzazione garantisce solo  
l'equivisoddisfattiabilità

## Modelli di Herbrand.

Sia  $\varphi$  una PCNF; con la semantizzazione abbiamo introdotto funzioni nzoarie.

Tuttavia, così diretta difficile descrivere l'insieme delle possibili interpretazioni.

Infatti,  $D$  è arbitrario! Quinti  $f: D \rightarrow D$  diretta ancora più arbitraria.

GOAL. Definire, data una PCNF  $\varphi$ , l'insieme delle interpretazioni canoniche.

Chiameremo tali interpretazioni come modelli di Herbrand.

Prp.  $\varphi$  clausale PCNF  $\Rightarrow \exists \bar{F}_H$  interpretazione di Herbrand canonica

Def. Sia  $S$  una PCNF in f. clausale,  
su cui:  $\mathcal{K}$  è l'insieme delle costanti.  
 $F$  è " " " funzioni

Definiamo  $\mathcal{H}_S$  l'universo di Herbrand con la def. induttiva:

- $\forall a_i \in A, a_i \in \mathcal{H}_S$  ( $A \subseteq \mathcal{H}_S$ )
- $\forall f_i^o \in F, f_i^o \in \mathcal{H}_S$  ( $F \subseteq \mathcal{H}_S$ )
- $\forall n \geq 1, \forall f_i^n \in F, \forall (t_1, \dots, t_n) \in \mathcal{H}_S$   
 $f_i^n(t_1, \dots, t_n) \in \mathcal{H}_S$

In altre parole,  $\mathcal{H}_S$  è l'insieme dei "termini ground" che possono essere formati dai suoi simboli

Notiamo che di solito è infinita, infatti se  $A = \{a\} \subset F = \{f(t)\}$

Allora

$$\begin{aligned} a \in \mathcal{H}_S &\Rightarrow f(a) \in \mathcal{H}_S \\ &\Downarrow \\ f(f(a)) &\in \mathcal{H}_S \\ &\Downarrow \\ &\vdots \\ f(f(\dots(f(a)\dots)) &\in \mathcal{H}_S \\ &\Downarrow \\ &\vdots \end{aligned}$$

Ex. Sia data la PCNF clausola

$$\begin{aligned} \{P(x); D(x)\} &\quad \{P(0)\} \quad \{\neg P(x); D(\text{succ}(x))\} \\ \{\neg D(x); P(\text{succ}(x))\} &\quad \{P(\text{succ}(0))\} \end{aligned}$$

Allora  $\mathcal{H} = \{0, \text{succ}(0), \text{succ}^2(0), \dots, \text{succ}^n(0), \dots\}$

$\vdash \mathbb{N}$

Sia  $\mathcal{H}_S$  universo di Herbrand su  $S$ .

Come possono creare un'interpretazione  $\times S$

su  $\mathcal{H}_S$ ? Ci servono delle assegnazioni  $\times$ :

- Costanti;
- Funzioni;
- Predicati;

Le prime due sono già in  $\mathcal{H}_S$  ("dominio degli identificativi") quindi restano così. Ci basta

dover definire le RELAZIONI su  
 $\mathcal{H}_S$  e completare.

Def. Sia  $S$  una formula in forma clausale dove:

$P_S$  è l'insieme dei predicati ( $P_S = \{P_1, \dots, P_n\}$ )  
 $F_S, A_S$  " delle funzioni e ( $F_S = \{f_1, \dots, f_e\}$ )  
costanti

Una interpretazione di Herbrand di  $S$

è una quadrupla

$$\mathcal{F}^S_H = \{\mathcal{H}_S, \{R_1, \dots, R_n\}, \{f_1, \dots, f_e\}, A_S\}$$

Dove  $R_i$  è una relazione su  $\mathcal{H}_S$  di aritma appropriata.

Inoltre  $f_i$  sono definite su un sottoinsieme appropriato di  $\mathcal{H}_S$ :

$$f_i(t_1, \dots, t_{j_i}), (t_1, \dots, t_{j_i}) \in \mathcal{H}_S$$

Definiamo la valutazione di  $\mathcal{F}_{\mathcal{H}}^S$  con

$$V(a) = a$$

$$V(f(t_1, \dots, t_n)) = f(V(t_1), \dots, V(t_n))$$

Se  $\mathcal{F}_{\mathcal{H}}^S \models S$  allora  $\mathcal{F}_{\mathcal{H}}^S$  è un modello di Herbrand per  $S$ .

Thm. (Completeness di Gödel)

Una formula in f. clausale



$$\exists \mathcal{F}_{\mathcal{H}}^S \mid \mathcal{F}_{\mathcal{H}}^S \models S$$

In termini pratici, se fissiamo il dominio sintattico e trovo una interpretazione che la soddisfa, posso sicuramente trovarla in  $\mathcal{H}$ .

$$\underline{\text{Ex.}} (\forall x, y, z)(P(x) \wedge R(f(y)) \rightarrow G(g(h(x, z)))$$

Allora un sottoinsieme dei funzioni grandi è  $f_1, f_2, \dots$

Allora un sostituzion dei termini

ground è  $f(a), f^2(a), a$

Sostituisce  $x \leftarrow f(a) | y \leftarrow f^2(a) | z \leftarrow a$

Allora ho

$P(f(a)) R(f^2(a)) \rightarrow G(g(h(f(a), a)))$

( $h$  è sicure soddisfacibile)

Dif. Dato un insieme di clausole FOL  $C$ ,  
e  $\exists I$  il dominio di Herbrand allora  
si ottien

$$C_{\exists I} = \{ C_{i,\sigma} : C_i \in C, \begin{array}{l} \sigma \text{ è una} \\ \text{sost. ground} \end{array} \text{ in } \exists I \}$$

l'insieme di tutte le possibili sostituzioni  
di  $C$  da  $\exists I$ .

Ovvio che  $\exists I$  infinito  $\Rightarrow C_{\exists I}$  infinito

Prop. Sia  $C = \{c_1, \dots, c_n\}$

Sia scelto  $c_i \in C$  e sia  $c_{i,\sigma}$  una sua  
sostituzione ground. Allora

$$C \Rightarrow c_{i,\sigma}$$

Ossia:

• Se usciamo la refutazione troverem  $\square$   
vull dire che non è possibile

- Se usciamo la rettazione trovam  $\square$   
vuol dire provare l'insoddisfacibilità di  $C$
- Se è infinito, non possiamo dire niente

### Thm. (completanza di Gödel)

Sia  $C$  un insieme di clausole e

$C_{\exists I}$  l'insieme delle sostituzioni di  $C$   
ground

Se  $C$  è insoddisfacibile allora  $\exists$  sottinsieme  
finito di  $C_{\exists I}$  insoddisfacibile

Quindi posso usare algoritmi proposizionali  
sulla insieme finiti di  $C_{\exists I} = \overline{C_{\exists I}}$ .

X

### Risoluzione

Def. Sono  $C_1, C_2$  clausole ground t.c.

$\ell \in C_1, \bar{\ell} \in C_2$  sono letterali complementari.

Allora  $C_1, C_2$  sono in conflitto e  
definiamo il nucleo risolvente

$$\text{Res}(C_1, C_2) = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\bar{\ell}\})$$

$$\text{Res}(C_1, C_2) = (C_1 \setminus \{\ell\}) \cup (C_2 \setminus \{\bar{\ell}\})$$

Ex.  $\underline{q(f(b))}, r(a, f(b)) \quad p(a), \neg \underline{q(f(b))}, r(f(a), b)$

$\swarrow \searrow$

$r(a, f(b)), p(a), r(f(a), b)$

Thm.  $\text{Res}(C_1, C_2)$  soddisfacibile



$C_1, C_2$  entrambi soddisfacibili

Strutturare questo col seguente algoritmo e controllare la validità:

- Nego formula
- Trasforma in f. clausale
- ( $\Rightarrow$ ) • Genero sottoinsieme finito di clausole grandi
- Controllo se è insoddisfacibile

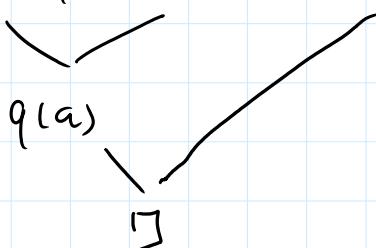
Exm.  $\neg p(x), q(x) \quad p(y) \quad \neg q(z)$

$\downarrow$

$\sigma:$

$x \leftarrow a$
$z \leftarrow a$
$y \leftarrow a$

$\neg p(a), q(a) \quad p(a) \quad \neg q(a)$



Allora la formula è VALIDA

Se non neghiamo, allora REFUTAMO e proviamo l'insoddisfabilità.

---

SOSTITUZIONE. Vogliamo effettuare la risoluzione su clausole "non ground" cercando delle sostituzioni intelligenti, i.e. che creino delle clausole in conflitto

Siano date  $x_1, \dots, x_n$  insiemi di variabili e  $t_1, \dots, t_n$  insiemi di termini. Diciamo che una sostituzione è una formula del tipo

$$\theta = (x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n)$$

Quindi data una clausola  $C$  abbiamo che  $C\theta$  è l'istanza di  $C$  dove tutte le sue occorrenze  $x_i$  sono sostituite da  $t_i$

Esempio.  $E = \{p(x), q(f(y))\}$  e  $\theta = (x \leftarrow y, y \leftarrow f(a))$

$$Allora E\theta = \{p(y), q(f(f(a)))\}$$

$$= \{p(f(a)), q(f^2(a))\}$$

Esempio.  $C = \{P(x), P(y), P(+(\text{succ}(x), \text{succ}(y)))\}$

Allora data  $\Theta = \{x \leftarrow \text{succ}(z), y \leftarrow \text{succ}(w)\}$

ottengo

$$C\Theta = \{P(\text{succ}(z)), P(\text{succ}(w)), P(+(\text{succ}^2(z), \text{succ}^2(w)))\}$$

Adesso consideriamo la composizione di sostituzioni, i.e. non devono essere grandi

$$\text{Sia } \Theta = (x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n)$$

$$\text{e } \sigma = (y_1 \leftarrow s_1, \dots, y_m \leftarrow s_m)$$

Allora

$$\Theta\sigma = \{x_i \leftarrow t_i\sigma \mid x_i \in X, x_i \neq t_i\sigma\} \cup$$

$$\{y_j \rightarrow s_j \mid y_j \in Y, y_j \notin X\}$$

Esempio. Sia  $F = P(u, v, x, y, z)$

$$\text{Se } \Theta = (x \leftarrow f(g), y \leftarrow f(a), z \leftarrow u)$$

$$\text{e } \sigma = (y \leftarrow g(a), u \leftarrow z, v \leftarrow f^2(a))$$

$$\text{Allora } \Theta\sigma = (x \leftarrow f(g(a)), y \leftarrow f(a), z \leftarrow f^2(a), v \leftarrow f^2(a))$$

Ovvvero

$$E(\theta\sigma) = p(z, f(a), f(g(a)), f(u), u)$$

Notiamo che

$$(E\theta)\sigma = E(\theta\sigma)$$

Prop.  $E(\theta\sigma) = (E\theta)\sigma$

$$\theta(\sigma\lambda) = (\theta\sigma)\lambda$$

X

Unificazione. Il processo di trovare una "sostituzione intelligente", i.e. da far entrare in conflitto due letterali, si dice **unificazione**

Esemp.  $\ell = p(f(x), g(y))$  e  $\ell' = \neg p(f(f(a)), g(z))$

non sono in conflitto

ma sostituendo

$$\theta_1 = (x \leftarrow f(a), y \leftarrow f(g(a)), z \leftarrow f(g(a)))$$

ottengo

$$\ell\theta_1 = p(f(f(a)), g(f(g(a))))$$

$$\ell'\theta_1 = \neg "$$

Allora  $\ell\theta_1, \ell'\theta_1$  sono in conflitto

Analoghe per

$$f_b = (x \leftarrow f(u), y \leftarrow a, z \leftarrow a)$$

Unificare

$$\theta_2 = (x \leftarrow f(a), y \leftarrow a, z \leftarrow a)$$

$$\mu = (x \leftarrow f(a), z \leftarrow y)$$

La sostituzione più "semplice" è  $\mu$

Notiamo che  $\theta_1 = \mu(y \leftarrow f(g(a)))$ ,  $\theta_2 = \mu(y \leftarrow a)$

Def. Sia  $U = (A_1, \dots, A_n)$  un insieme  
di formule atomiche.

Un unificatore  $\theta$  è una sostituzione  
t.c.

$$A_1\theta = \dots = A_n\theta$$

Ancora più generalmente, un unificatore  $\theta$  è  
la più generale sse vale

da  $\forall \theta$  unificatore,  $\exists \lambda$  sost. |

$$\theta = \mu \lambda$$

Q. Quando degli atomi sono unificabili e NON?

Prp. Se due atomi sono t.c.:

- Hanno predici diversi (e.g.  $p(x), q(x)$ )
- Hanno predici uguali ma funzioni esternne diverse  
(e.g.  $p(f(x)), p(g(x))$ )
- Sono di forma  $p(x), p(f(x))$

Allora NON sono unificabili. Altrimenti

- sono o. variaz p", o  
Allora NON sono unificabili. Altrimenti  
lo sono.

Dato un insiem di termini da unificare,  
li scriviamo come un insiem di termini di equazioni

Ex.  $\ell = p(f(x), g(y))$  e  $\ell' = \neg p(f(f(a)), g(z))$   
 $\left\{ \begin{array}{l} x = f(a) \\ y = z \end{array} \right.$

Un insieme di termini d'equazione si dice  
risolta se:

- Tutte le equazioni sono di forma

$$x_i = t_i$$

- Ogni variabile non appaia in nessun altro elento dell'insiem
- Possiamo def. la seguente sostituzion:

$$(x_1 = t_1, \dots, x_n = t_n)$$

## ALGORITMO.

Dati un insiem di termini d'equazione:

- 1) • Trasform  $t = x$ , dove  $t$  non è una variabile, in  $x = t$
- 2) • Eliminare  $x = x$
- 3) • Per  $t' = t''$  (non variabili),
  - Se le funzioni più esten sono  $\neq$ :
  - N/N unificabili.

- Se le funzioni più esterne sono  $f$ :
  - NON unificabile
  - Altrimenti  $f(t'_1, \dots, t'_{t_n}) = f(f''_1, \dots, f''_{t_n})$   
per tutte le equazioni di forma  
 $t'_i = t''_n$

4) Se  $x=t$  è f.c.  $x$  appaia già in un'altra equazione,

- Se  $x$  appaia in  $t$  ma  $x \neq t$  allora non è unificabile
- Altrimenti sostituisco  $x \leftarrow t$  nelle altre equazioni

Esempio.  $\underset{x}{g}(\gamma) = x$   
 $f(x, \underset{h(x)}{h(x)}, y) = f(\underset{y}{g(z)}, w, z)$

↓

$$x = g(\gamma)$$

$$x = g(z)$$

$$\underset{h(x)}{h(x)} = w$$

$$y = z$$

↓

$$g(z) = g(\gamma)$$

$$x = g(z)$$

$$w = h(x)$$

$$y = z$$

↓

$$z = \gamma$$

$$x = g(z)$$

$$\begin{aligned}z &= y \\x &= g(z) \\w &= h(x) \\y &\cancel{=} z\end{aligned}$$

Allora una sostituzione è

$$\mu = (z \leftarrow s, x \leftarrow g(z), w \leftarrow h(x))$$

## Programmazione Logica

Lavoriamo ancora nella logica del 1° ordine (i.e. dei predicati).

Con la risoluzione abbiamo stabilito un modo x dimostrare i teoremi automaticamente.

Tuttavia, x programmare le computazioni possiamo prendere una forma ristretta di risoluzione. Tale approccio si chiama **programmazione logica**.

Tale approccio consiste in:

- Insieme di clausole  $\rightarrow$  PROGRAMMA
- Clausola ulteriore  
che potrebbe entrare  $\rightarrow$  QUERY  
in conflitto col programma

Una query viene assunta come negazione del risultato del programma

- Se la refutazione succede, allora  
 $\text{Program} \not\models \text{Query}$

Ossia  $\text{Program} \rightarrow \neg \text{Query}$

Le unificazioni svolte sono fatte x dare "risposte in più" sulla query, oltre al fatto di farla sua negazione.

dare risposte in più sulla query, oltre ai fatti della sua negazione sia vera o meno

→ programma viene composto dalle "classi di programmazione" che sono:

- Predicati universali chiusi  
 $\forall x (x + 0 = x) \rightarrow$  hanno una loro versione clausale
  - Implicazioni universali chiusi dove  
 la premessa è una congiunzione
 
$$\forall x \forall y \forall z (x \leq z \wedge y \leq z) \Rightarrow (x \leq z)$$
  - Clausole di Horn, dove tutte sono negative tranne l'ultima (non quantificata!)

Quindi, dato un formula  $G_1 \wedge \dots \wedge G_n$ , vogliamo dimostrare che essa è una implicazione del programma. Lo facciamo facendo la refutazione di  $\neg G_1 \vee \dots \vee \neg G_n$  con il programma.

Definiamo  $\neg G_1 V \dots \neg G_n$  come la **clausola goal**, che può <sup>solo</sup> esser in conflitto con sbarale il letterale positivo del programma.

Sia dunque  $B_1, V \dots V \neg B_m$  il programma, si ha che  $G_i, B_i$  possono essere unitificati da  $\sigma$ :

- (1)  $C = \{V \neg C, V \neg B_i \mid V B_i\}$

da  $\sigma$ :

$$(A \rightarrow G_2 \vee \dots \vee G_n \vee B_2 \vee \dots \vee B_m) \sigma$$

Quindi facciamo la risoluzione su (a).

Notiamo che facendo risoluzioni su risoluzioni otteniamo delle sostituzioni, che sono le risposte alla query!

Quindi la programmazione viene in un paradigma

DECHIARATIVO, dove le "regole" descrivono la relazione tra l'input e l'output. In particolare, usa le clausole di Horn. La **risoluzione SLD** si occuperà di eseguire le clausole di Horn!

Facciamo un esempio di prog. logica.

→ Consideriamo le stringhe su un alfabeto

{}, dotato della concatenazione • e

dei seg. predici:

- Sub( $x, y$ ):  $x$  è sottostringa di  $y$
- Prefix( $x, y$ ):  $x$  è prefisso di  $y$
- Suffix( $x, y$ ):  $x$  è suffisso di  $y$

→ Ricaviamo gli seg. assiomi, che andranno a "formare" il programma

, , , , ,

## r. programma

1.  $\text{Sub}(x, x)$
2.  $\text{Suffix}(x, y \cdot x)$
3.  $\text{Prefix}(x, x \cdot y)$
4.  $(\text{Sub}(x, y) \wedge \text{Suffix}(y, z)) \Rightarrow \text{Sub}(x, z)$
5.  $(\text{Sub}(x, y) \wedge \text{Prefix}(y, z)) \Rightarrow \text{Sub}(x, z)$

In forma clausale 4, 5 diventano

4.  $\neg \text{Sub}(x, y) \vee \neg \text{Suffix}(y, z) \vee \text{Sub}(x, z)$
5.  $\neg \text{Sub}(x, y) \vee \neg \text{Prefix}(y, z) \vee \text{Sub}(x, z)$

→ Vogliamo verificare se  $\text{Sub}(\text{abc}, \text{aabcc})$   
sia vera!

→ La negazione è  $\neg \text{Sub}(\text{abc}, \text{aabcc})$

Refutiamo

$$\neg \text{Sub}(\underline{\text{a} \cdot \text{b} \cdot \text{c}}, \underline{\text{a} \cdot \text{a} \cdot \text{b} \cdot \text{c} \cdot \text{c}})$$

$$1 \downarrow \sigma = \{x \leftarrow \text{abc}, y \leftarrow y_1, z \leftarrow \text{aabcc}\}$$

$$\neg \text{Sub}(\text{a} \cdot \text{b} \cdot \text{c}, \text{a} \cdot \text{a} \cdot \text{b} \cdot \text{c} \cdot \text{c}) \vee \neg \text{Sub}(\text{abc}, y_1)$$

$$\vee \neg \text{Suffix}(y_1, \text{aabcc})$$

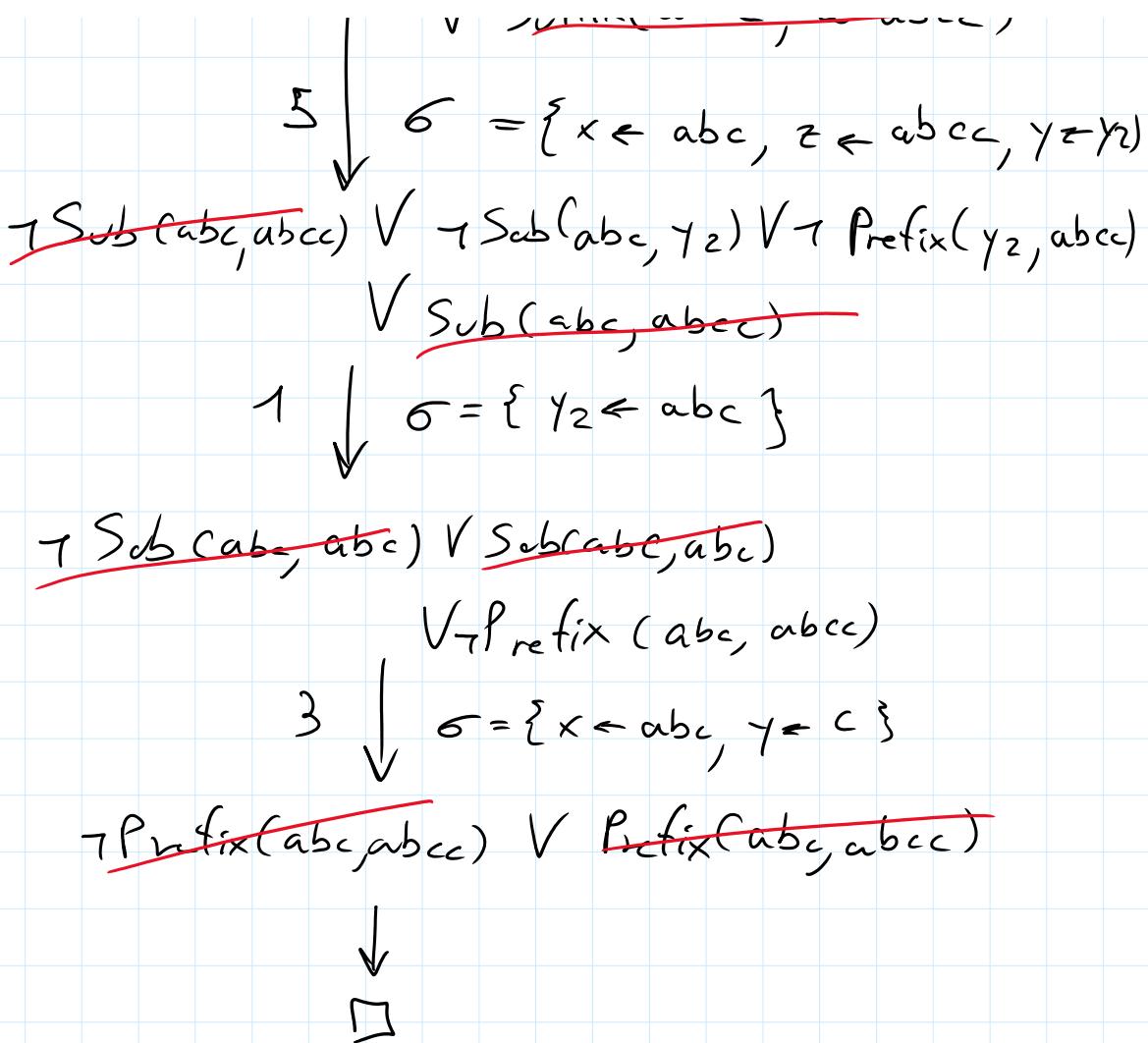
$$\vee \neg \text{Sub}(\text{abc}, \text{aabcc})$$

$$2 \downarrow \sigma' \circ \sigma = \{ \dots, y \leftarrow a, y_1 \leftarrow \text{abcc} \}$$

$$\neg \text{Sub}(\text{abc}, \text{abc}) \vee \neg \text{Suffix}(\text{abc}, \text{aabcc})$$

$$\vee \neg \text{Suffix}(\text{abcc}, \text{aabcc})$$

$$\Sigma | - s . . . ,$$



Dimostrando #

---

X

Per definire la risoluzione SLD, dobbiamo  
definire:

- COME scegliere le clausole di programma
- COME effettuare le sostituzioni

La scelta è altamente non deterministica!

Facciamo un poio di notazioni

- Una clausola di Horn è di forma

$$A \Leftarrow B_1, \dots, B_n \equiv A \vee \neg B_1 \vee \dots \vee \neg B_n$$

- $A$  si dice testa e  $\neg B_1, \dots, \neg B_n$  il corpo.
- Una clausola goal è una clausola di Horn senza letterali positivi
- Una clausola è del programme se è una di Horn con un letterale positivo o >1 letterali negativi
- Un insiem di clausole non-goal avanti identica testa è una procedura
- Un insiem di procedure è un programma (logico)
- Una procedura solo con termini grandi è un database

Exm.

$$1. q(x, y) \Leftarrow p(x, y)$$

$$2. q(x, y) \Leftarrow p(x, y), q(z, y)$$

$$3. p(b, a)$$

$$4. p(c, a)$$

5  
⋮

Programma logico

Programma

Database

Sia  $P$  un programma e  $G$  una clausola goal. Una sostituzione  $\theta$  per le variabili in  $G$  si dice **corretta** se

$$P \models \forall \exists G\theta$$

Ovvero soddisfa il quantificatore universale sull'insieme di variabili libere in  $\exists G\theta$

Esempio. Sia  $P$  gli assiomi dell'aritmetica,

$$\text{Sia } G := \exists (G + y = 13)$$

Allora la sostituzione  $\theta = (y \leftarrow 7)$  è corretta, i.e.  $P \models (6 + 7 = 13)$

Allora  $\forall \theta$  corretta e  $\forall \sigma$  sostituzione

si ha che  $(G_1 \wedge \dots \wedge G_n)\theta\sigma$  è sempre vera in  $P$ .

### RISOLUZIONE SLD.

Sia  $P$  il programma,  
 $R$  una regola di  
computazione e  $G$   
una clausola goal.

Si definisce  $G_0 := G$  e deriviamo  
 $G_i$  da  $G_j$ , selezionando un letterale

$A_i^j \in R$ . ... al until  $R \in P$  tr. (2)

$A_i^j \in G_i$ , una clausola CGP t.c.

la f.t.a di  $C$  si unifichi con

$A_i^j$  con una sostituzione  $mgu \Theta_i^j$  e  
risolvendo dunque

$$\{ G_i : \Leftarrow A_i^1, \dots, A_i^{j-1}, A_i^j, \dots, A_i^{n_i} \}$$

$$C_i = B_i^0 \Leftarrow B_i^1, \dots, B_i^{k_i}$$

$$A_i^j \Theta_i^j : B_i^0 \Theta_i^j$$

$$G_{i+1} : \Leftarrow (A_i^1, \dots, A_i^{j-1}, B_i^0, A_i^j, \dots, A_i^{n_i}) \Theta_i^j$$

Una SLD-derivazione di  $\square$  è una  
SLD refutazione.

Thm. Siamo  $\Theta = \Theta_1 \dots \Theta_n$  gli unificatori

usati x la SLD, allora la restrizion

di  $\Theta$  su  $G$  è una certa

sostituzion. Ovvero,

$$P \models G_\sigma \quad (\text{correttezza})$$

Thm. Inoltre la SLD-refutazion è completa,

i.e. per ogni  $P, R, G$  e  $\sigma$

allora ci sarà una SLD-refutazion

t.c.  $\sigma$  è la restrizion della  
scq. di unificatori  $\Theta_1, \dots, \Theta_n$  csat;

→)

Scelta di  $A_i$ : regola di  
computazion

Scelta di  $C_i$ : legge  
di ricerca

I.c.  $\sigma$  è un'operazione di scrittura di unificazione  $\theta_1, \dots, \theta_n$  csat; durante la SLD.

Esemp.

1. $q(x, y) \leftarrow p(x, y)$	7. $p(f, b)$
2. $q(x, y) \leftarrow p(x, z), q(z, y)$	8. $p(h, g)$
3. $p(b, a)$	9. $p(i, h)$
4. $p(c, a)$	10. $p(j, h)$
5. $p(d, b)$	
6. $p(e, b)$	

Goal:

$$\Leftarrow q(y, b), q(b, z)$$

$$G_0 = \Leftarrow q(y, b), q(b, z)$$

1: Scelgo  $q(y, b)$  e la risolvo con 1 (con sost. implicita  $(x \Leftarrow y, y \Leftarrow b)$ )

$$\cancel{q(y, b)} \Leftarrow \cancel{q(y, b)}, q(b, z), p(y, b)$$

$$G_1 = \Leftarrow q(b, z), p(y, b)$$

2: Scelgo  $p(y, b)$  e refuto con 5 con sost.t.  $\theta_1 = (y \Leftarrow d)$

$$G_2 = \Leftarrow q(b, z)$$

3: Rimane da scrivere  $q(b, z)$  con 1:

$$G_3 = \Leftarrow p(b, z)$$

↳ Refuto con 3. usando  $\theta_2 = (z \Leftarrow a)$

$$G_4 = \square$$

Quindi refut con  $\theta = \{y \Leftarrow d, z \Leftarrow a\} \#$

Notiamo che in 2. potrò usare anche

$$\theta'_2 = (y \Leftarrow c)$$

Quindi a seconda delle regole di ricerca

(Quindi a seconda delle regole di ricerca  
ho risposte diverse (ma sempre giuste!))

Il thm. di completezza ci dice che la  
SLD-refutazione esiste indipendentemente dalle  
regole di computation.

Perciò non vale la stessa cosa x la regola  
di ricerca, posso avere effetti diversi!

- |     |                                       |
|-----|---------------------------------------|
| 1.  | $q(x, y) \leftarrow p(x, y)$          |
| 2.  | $q(x, y) \leftarrow p(x, z), q(z, y)$ |
| 3.  | $p(b, a)$                             |
| 4.  | $p(c, a)$                             |
| 5.  | $p(d, b)$                             |
| 6.  | $p(e, b)$                             |
| 7.  | $p(f, b)$                             |
| 8.  | $p(h, g)$                             |
| 9.  | $p(i, h)$                             |
| 10. | $p(j, h)$                             |

Goal:

$$\Leftarrow q(\gamma, b), q(b, z)$$

1. Scelgo  $q(\gamma, b)$  vs 2

$$\Leftarrow p(\gamma, z), q(z, b), q(b, z)$$

2. Scelgo  $p(\gamma, z)$  vs 6, ( $\gamma \leftarrow e$ ,  $z' \leftarrow b$ )

$$\Leftarrow q(b, b), q(b, z)$$

3. ??? Non posso fare NIENTE con  
 $q(b, b)$ , non si cancella mai...

Quindi x una refutazion devo provare,  
non deterministica, tutte le ricerche possibili.

Rappresento le SLD-derivazioni w/  
seguito obbligo:

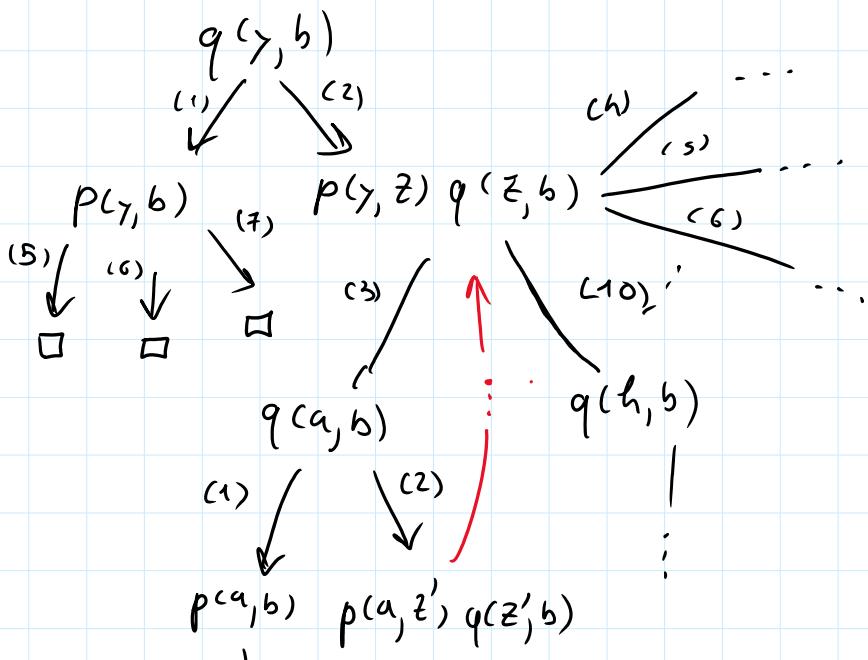
- $G_0$  è il nodo radice

- $G_0$  è il nodo radice
- Per ogni nodo  $G_n$  contiene  $n$ ; figli,  
dove il  $G_{ni}$  è ottenuto risolvendo  
il letterale scelto da  $R$  con la  
testa della clausola scelta di  $P$
- Se:
  - $\gamma_1$  branch ha refutazione  $\Rightarrow$  SUCCESSO
  - Non si unifica  $\Rightarrow$  FALLIMENTO
  - Non termina  $\Rightarrow$  INFINITO

Ex. Dato  $P$

1.	$q(x, y) \leftarrow p(x, y)$
2.	$q(x, y) \leftarrow p(x, z), q(z, y)$
3.	$p(b, a)$
4.	$p(c, a)$
5.	$p(d, b)$
6.	$p(e, b)$
7.	$p(f, b)$
8.	$p(g, g)$
9.	$p(h, h)$
10.	$p(j, h)$

Sia  $G = \Leftarrow q(\gamma, b)$ . Allora una sua SLD-tree è



$$\begin{array}{c} p^{(a,b)} \quad p^{(a,z')} q^{(z',b)} \\ | \qquad | \\ x \qquad \infty \end{array}$$

Q. Come implementiamo la ricerca?

Preferibile una BFS x evitare cicli infiniti