Intelligenza Artificiale II

Logica del Primo Ordine

Parte 2: Automazione

Marco Piastra

1

Risoluzione, unificazione programmazione logica

Decidibilità ed automazione di L_{PO}

- Indecidibilità di L_{PO}
  - non esiste un algoritmo (di valore generale) in grado di stabilire se una fbf è un teorema o no
  - al contrario del caso proposizionale, in L_{PO} non si possono verificare direttamente tutte le possibili interpretazioni
- Qual è quindi la speranza di avere un calcolo automatico?
- In realtà, L_{PO} è **semi-decidibile**
  - E` possibile stabilire in modo automatico ed in un tempo finito se

    $\Gamma \models \varphi$
  - ... ma non se

    $\Gamma \not\models \varphi$
  - In altri termini, esistono algoritmi che:
    - posti di fronte al problema "$\Gamma \models \varphi$?"
    - terminano con successo se $\Gamma \models \varphi$
    - possono divergere (i.e. girare all'infinito) se $\Gamma \not\models \varphi$

Universo e base di Herbrand

- **Termini** e **atomi** di Herbrand
  - Dato un linguaggio L_{PO}
  - Un **termine** di Herbrand è un termine chiuso (*ground term*) (= che non contiene variabili)
    - Esempi: $f(a)$, $g(a,b)$, $g(f(a),b)$, $g(f(a),g(b,c))$, $g(f(a),g(f(b),c))$, ...
  - Un **atomo** di Herbrand è una fbf *atomica* (*ground atom*) (= che non contiene variabili)
    - Esempi: $P(f(a))$, $P(g(a,b))$, $Q(g(f(a),b), g(f(a),g(b,c)))$, ...
- **Universo** e **base** di Herbrand
  - L'**universo** di Herbrand è l'insieme di tutti i termini di Herbrand

    $U_H \equiv \{f(a), g(a,b), g(f(a),b), g(f(a),g(b,c)), g(f(a),g(f(b),c)), \ldots\}$
  - La **base** di Herbrand è l'insieme di tutti gli atomi di Herbrand

    $B_H \equiv \{P(f(a)), P(g(a,b)), Q(g(f(a),b), g(f(a),g(b,c))), \ldots\}$
- (La base di Herbrand B_H permette di definire interpretazioni di L_{PO} usando gli oggetti del linguaggio stesso ...)

Modelli di Herbrand

- **Struttura** di Herbrand per $\mathrm{L_{PO}}$
  - Una struttura $<\mathbf{U}_H, v_H>$ tale che
    - $\forall c \in \mathrm{Cost}(\mathrm{L_{PO}}),\ v_H(c) = c$
    - $\forall t \in \mathbf{U}_H,\ v_H(t) = t$
- **Interpretazione** v_H di Herbrand
  - un qualsiasi **sottoinsieme** della base di Herbrand $\mathrm{B_H}$
  - $v_H \equiv \{P(a), P(f(b)), P(c), Q(a,g(b,c)), Q(b,c) \ldots\}$ (solo formule atomiche chiuse)
- **Modello** di Herbrand

$\varphi \in \mathrm{Atomi}(\mathrm{L_{PO}}),\ <\mathbf{U}_H, v_H>[s] \models \varphi$ sse $\varphi \in v_H$

$\varphi \in \mathrm{Atomi}(\mathrm{L_{PO}}),\ <\mathbf{U}_H, v_H>[s] \models \neg\varphi$ sse $\varphi \notin v_H$

$<\mathbf{U}_H, v_H>[s] \models \neg\varphi$ sse $<\mathbf{U}_H, v_H>[s] \not\models \varphi$

$<\mathbf{U}_H, v_H>[s] \models \varphi \rightarrow \psi$ sse non $(<\mathbf{U}_H, v_H>[s] \models \varphi$ e $<\mathbf{U}_H, v_H>[s] \not\models \psi)$

$<\mathbf{U}_H, v_H>[s] \models \forall x\, \varphi$ se per ogni $c \in \mathrm{Cost}(\mathrm{L_{PO}})$ si ha $<\mathbf{U}_H, v_H>[s](x{:}c) \models \varphi$

Teorema di Herbrand

- **Sistema** di Herbrand di un enunciato
  - Dato un enunciato universale, cioè della forma
    $\forall x_1 \forall x_2 \ldots \forall x_n \varphi$ (φ non contiene quantificatori)
  - è l'insieme (anche infinito) di formule **chiuse** generato per sostituzione
    $\varphi\ [x_1/t_1, x_2/t_2 \ldots x_n/t_n]$
  - con tutte le possibili combinazioni $<t_1, t_2 \ldots t_n>$ di $t_i \in \mathbf{U}_H$

- **Sistema** di Herbrand di una teoria
  - Data una teoria Σ di enunciati universali
    è l'unione $H(\Sigma)$ di tutti i sistemi di Herbrand generati dagli enunciati Σ

- **Teorema di Herbrand**
  - Data una teoria di enunciati universali Σ,
    $H(\Sigma)$ ha un modello sse Σ ha un modello
  - ... ma qual'è l'utilità?
    - $H(\Sigma)$ può essere infinito anche quando Σ è finito
    - il teorema si applica solo agli enunciati universali

Forma normale prenessa

- Forma normale **prenessa** FNP (i.e. tutti i quantificatori all'inizio)
  - Una fbf φ qualsiasi può essere trasformata in un enunciato equivalente
    $\mathbf{Q}_1 x_1 \mathbf{Q}_2 x_2 \ldots \mathbf{Q}_n x_n \psi$ (ψ è anche detta ***matrice***)
  - dove $\mathbf{Q}_i$ è $\forall$ oppure $\exists$ e ψ non contiene quantificatori
  - si ottiene usando la definizione $\neg\forall x\, \varphi \leftrightarrow \exists x\, \neg\varphi$ e le equivalenze logiche derivanti (p.es. $(\varphi \rightarrow \forall x\, \psi) \leftrightarrow \forall x\, (\varphi \rightarrow \psi)$, $(\forall x\, \varphi \rightarrow \psi) \leftrightarrow \exists x\, (\varphi \rightarrow \psi)$)
  - Attenzione: l'ordine dei quantificatori $\mathbf{Q}_1 x_1 \mathbf{Q}_2 x_2 \ldots \mathbf{Q}_n x_n$ è significativo
    - $\forall y \exists x\, Padre(x,y)$ "qualsiasi y ha un padre" NON equivale a
    - $\exists x \forall y\, Padre(x,y)$ "esiste un x che è padre di tutti"
  - Esempi:
    - $\exists y\, (P(y) \rightarrow \forall x\, P(x))$
      $\exists y \forall x\, (P(y) \rightarrow P(x))$ (FNP, usando $(\varphi \rightarrow \forall x\, \psi) \leftrightarrow \forall x\, (\varphi \rightarrow \psi)$)
    - $\exists y\, (\forall x\, P(x) \rightarrow P(y))$
      $\exists y \exists x\, (P(x) \rightarrow P(y))$ (FNP, usando $(\forall x\, \varphi \rightarrow \psi) \leftrightarrow \exists x\, (\varphi \rightarrow \psi)$)
    - $\forall x \exists y\, (Q(x,y) \rightarrow P(y)) \wedge \neg\forall x\, P(x)$
      $\forall x \exists y\, (Q(x,y) \rightarrow P(y)) \wedge \exists x\, \neg P(x)$ (definizione $\neg\forall x\, \varphi \leftrightarrow \exists x\, \neg\varphi$)
      $\forall x \exists y\, (Q(x,y) \rightarrow P(y)) \wedge \exists z\, \neg P(z)$ (ridenominazione di x in z)
      $\forall x \exists y \exists z\, ((Q(x,y) \rightarrow P(y)) \wedge \neg P(z))$ (FNP)

Forma normale di Skolem

- Forma normale di Skolem
  - si eliminano i quantificatori esistenziali in una forma normale prenessa
  - sostituendo ciascuna variabile esistenzialmente quantificata con una (nuova) costante o una (nuova) funzione:
  - Si opera sui quantificatori della formula $\mathbf{Q}_1 x_1 \mathbf{Q}_2 x_2 \ldots \mathbf{Q}_n x_n \psi$ partendo da sinistra
    - per ogni $\mathbf{Q}_i x_i$ della forma $\exists x_i$:
    1) si applica a ψ la sostituzione $[x_i/k_i(x_1, \ldots, x_{i-1})]$ (nuova funzione di Skolem) oppure $[x_i/k_i]$ (nuova costante di Skolem), se $i = 1$
    2) si elimina $\exists x_i$ dalla formula
  - Esempi:
    - $\exists y \forall x\ (P(y) \rightarrow P(x))$
      $\forall x\ (P(k) \rightarrow P(x))$ (k costante di Skolem: si espande il linguaggio)
    - $\forall x \exists y \exists z\ ((Q(x,y) \rightarrow P(y)) \wedge \neg P(z))$
      $\forall x\ ((Q(x, k(x)) \rightarrow P(k(x))) \wedge \neg P(j(x)))$ ($k(.)$ e $j(.)$ funzioni di Skolem)
- Teorema
  - Per qualsiasi enunciato φ,
    φ ha un modello sse $\boldsymbol{sko}(\varphi)$ (forma normale di Skolem di φ) ha un modello

Semi-decidibilità effettiva di L_{PO}

- Corollario del teorema di Herbrand
  - Le seguenti affermazioni sono equivalenti
    - $\Gamma \models \varphi$
    - $\Gamma \cup \{\neg\varphi\}$ non è soddisfacibile (= non ha un modello) (= è inconsistente)
    - Esiste un sottoiniseme **finito** di $\mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$ (sistema di Herbrand della forma di Skolem) che è **contraddittorio**
  - Quindi:
    - Una procedura che enumera *tutti* i sottoinsiemi finiti di $\mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$ prima o poi (in un tempo finito) ne trova uno contraddittorio (sse $\Gamma \models \varphi$)
  - Esempio:

1: $\forall x\,(P(f(x)) \rightarrow P(g(x))), \exists y\,\neg P(g(y)) \models \exists z\,\neg P(f(z))$	(problema iniziale)
2: $\{\forall x\,(P(f(x)) \rightarrow P(g(x))), \exists y\,\neg P(g(y)), \neg\exists z\,\neg P(f(z))\}$	($\Gamma \cup \{\neg\varphi\}$)
3: $\{\forall x\,(P(f(x)) \rightarrow P(g(x))), \exists y\,\neg P(g(y)), \forall z\,P(f(z))\}$	(definizione di $\exists$)
4: $\{\forall x\,(P(f(x)) \rightarrow P(g(x))), \neg P(g(k)), \forall z\,P(f(z))\}$	($\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\})$)
5: $\{(P(f(k)) \rightarrow P(g(k))), \neg P(g(k)), P(f(k))\}$	($\in \mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$, $[x/k, z/k]$)
6: $\{(\neg P(f(k)) \vee P(g(k))), \neg P(g(k)), P(f(k))\}$	(traduzione di $\rightarrow$)
7: $\{\neg P(f(k)), P(f(k))\}$	(risoluzione di $P(g(k))), \neg P(g(k))$, contraddizione)

Risoluzione per refutazione

- Una procedura effettiva
  - per trovare un sottoinsieme finito e contraddittorio di $\mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$
    - sperabilmente più efficiente dell'enumerazione ricorsiva
    - può divergere (i.e. non terminare) se $\Gamma \not\models \varphi$ (altrimenti L_{PO} sarebbe decidibile)

- **Risoluzione per refutazione**: caso proposizionale (vedi IA1)
  - Si traduce $\Gamma \cup \neg\varphi$ in forma normale congiuntiva (FNC)

    $\beta_1 \wedge \beta_2 \wedge \ldots \wedge \beta_n$ dove ogni β_i è una disgiunzione di letterali del tipo A o $\neg A$

    Esempio: $(\neg A \vee B) \wedge (C \vee \neg A)$
  - dalla $\Gamma \cup \neg\varphi$ in FNC si passa alla forma a clausole (FC)

    $\{\beta_1, \beta_2, \ldots, \beta_n\}$ dove ogni β_i è una fbf separata, in cui si omette il simbolo $\wedge$

    Esempio: $\{\{\neg A, B\}, \{C, \neg A\}\}$
  - si applica quindi in modo esaustivo la regola di risoluzione (ristretta)

    $\{\beta_1, \beta_2, \ldots, \beta_n, A\}, \{\neg A, \gamma_1, \gamma_2, \ldots, \gamma_m\} \vdash \{\beta_1, \beta_2, \ldots, \beta_n, \gamma_1, \gamma_2, \ldots, \gamma_m\}$
  - fino a derivare la clausola vuota (successo) o al termine (fallimento)

Forma a clausole in $\mathrm{L_{PO}}$

- Molto simile al caso proposizionale …
  - Si parte da $\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\})$
    - cioè tutti gli enunciati di $\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\})$ sono nella forma

      $\forall x_1 \forall x_2 \ldots \forall x_n \varphi$ (φ non contiene quantificatori)
    - Essendo tutti universali, i quantificatori si possono omettere
    - La matrice φ viene tradotta in FNC (con le stesse regole del caso proposizionale)
    - e quindi in forma a clausole FC (con le stesse regole del caso proposizionale)
  - Esempio:
    - $\forall x\,(P(x) \rightarrow \exists y\,(Q(x,y) \wedge R(y)))$ (enunciato di partenza)
    - $\forall x \exists y\,(P(x) \rightarrow (Q(x,y) \wedge R(y)))$ (forma normale prenessa)
    - $\forall x\,(P(x) \rightarrow (Q(x,k(x)) \wedge R(k(x))))$ (skolemizzazione, nuova funzione $k(.)$)
    - $P(x) \rightarrow (Q(x,k(x)) \wedge R(k(x)))$ (eliminazione dei quantificatori)
    - $\neg P(x) \vee (Q(x,k(x)) \wedge R(k(x)))$ (equivalenza di $\rightarrow$)
    - $(\neg P(x) \vee Q(x,k(x))) \wedge (\neg P(x) \vee R(k(x)))$ (FNC, distribuitività di $\vee$)
    - $\{\neg P(x), Q(x,k(x))\},\ \{\neg P(x), R(k(x))\}$ (FC)

Forma a clausole e risoluzione

- Assai più complessa rispetto al caso proposizionale.
  - Esempio (due enunciati):
    - $\forall x \exists y\ (Q(x,y) \vee P(g(x,f(a)),a))$
      $\forall x \exists y\ (\neg P(g(b,f(x)),y) \vee \neg R(y))$ (enunciati di partenza)
    - $\forall x \exists y\ (P(g(x,f(a)),a) \vee Q(x,y))$
      $\forall w \exists z\ (\neg \mathrm{P}(g(b,f(w)),w) \vee \neg R(z))$ (diversificare le variabili: *standardizzazione*)
    - $\{P(g(x,f(a)),a),\ Q(x,\ j(x))\}$
      $\{\neg P(g(b,f(w)),w),\ \neg R(k(w))\}$ (FC, $j(.)$ e $k(.)$ funzioni di Skolem)

      L'unica coppia di **atomi** compatibili è
      $P(g(x,f(a)),a),\ \neg P(g(b,f(w)),w)$
      Tuttavia i due atomi sono sintatticamente **diversi**
      Applicando la sostituzione $\sigma = [x/b,\ w/a]$ si ottiene
      $\{P(g(b,f(a)),a),\ Q(b,j(b))\}$
      $\{\neg P(g(b,f(a)),a),\ \neg R(k(a))\}$ (applicazione di σ)
      $\{Q(b,j(b)),\ \neg R(k(a))\}$ (risoluzione $\{\psi, \alpha\}, \{\neg\alpha, \chi\} \vdash \{\psi, \chi\}$)
  - La sostituzione σ si dice **unificatore** delle due clausole
    - va applicata integralmente a tutte e due le clausole da risolvere

Unificazione

- **Unificatore** di due clausole
  - Una sostituzione $\sigma = [x_1/t_1, x_2/t_2 \ldots x_n/t_n]$ che rende risolvibile una coppia di atomi α e $\neg\beta$
  - in simboli: $\sigma(\alpha) \equiv \sigma(\beta)$
  - in ciascuna sostituzione x_i/t_i la variabile x_i **non può** comparire in t_i
  - la sostituzione σ deve essere applicata a tutte e due le clausole da risolvere
  - ovviamente, σ non esiste sempre:
    $P(g(x,f(a)),a)$, $\neg P(g(b,f(w)),k(w))$

- **Unificatore più generale** (MGU - *most general unifier*)
  - se esiste un unificatore di α e $\neg\beta$
  - esiste anche un unificatore più generale MGU μ

    $$\text{MGU } \mu \Leftrightarrow \forall\, \sigma\ \exists\, \sigma' \mid \sigma = \mu \cdot \sigma'$$

  - cioè qualsiasi altro unificatore può essere ottenuto per composizione da μ
  - (intuitivamente, μ è la minima sostituzione indispensabile)
  - esiste un algoritmo che trova μ (se la coppia α e $\neg\beta$ è unificabile, ovviamente)

Costruzione del MGU

- Algoritmo (Martelli, Montanari)
  - Input: $\{s_1 = t_1, s_2 = t_2 \ldots s_n = t_n\}$ (equazioni tra termini: gli argomenti dei due atomi)
  - Procedura:
    - Applicare le seguenti regole, in ordine qualsiasi
      (ciascuna applicazione riscrive l'insieme di equazioni)

(1) $f(s_1, ..., s_n) = f(t_1, ..., t_n)$	*replace by the equations* $s_1 = t_1, ..., s_n = t_n$,
(2) $f(s_1, ..., s_n) = g(t_1, ..., t_m)$ where $f \neq g$	*halt with failure,*
(3) $x = x$	*delete the equation,*
(4) $t = x$ where t is not a variable	*replace by the equation* $x = t$,
(5) $x = t$ where x does not occur in t and x occurs elsewhere	*apply the substitution* $\{x/t\}$ *to all other equations*
(6) $x = t$ where x occurs in t and x differs from t	*halt with failure.*

    - La procedura termina quando non vi sono più regole applicabili o quando si ha un fallimento (regole (2) e (6))
    - Si ha **successo** sse tutte le equazioni sono del tipo $x_i = t_i$

Esempio 8

- Costruzione del MGU
  - $\{f(x,a) = f(g(z),y),\ h(u) = h(d)\}$

    $\{x = g(z),\ y = a,\ h(u) = h(d)\}$ — Regola (1) su $f(x,a) = f(g(z),y)$

    $\{x = g(z),\ y = a,\ u = d\}$ — Regola (1) su $h(u) = h(d)$, MGU

  - $\{f(x,a) = f(g(z),y),\ h(x,z) = h(u,d)\}$

    $\{x = g(z),\ y = a,\ h(x,z) = h(u,d)\}$ — Regola (1) su $f(x,a) = f(g(z),y)$

    $\{x = g(z),\ y = a,\ h(g(z),z) = h(u,d)\}$ — Regola (5) su $x = g(z)$

    $\{x = g(z),\ y = a,\ u = g(z),\ z = d\}$ — Regola (1) su $h(g(z),z) = h(u,d)$

    $\{x = g(d),\ y = a,\ u = g(d),\ z = d\}$ — Regola (5) su $z = d$, MGU

  - $\{f(x,a) = f(g(z),y),\ h(x,z) = h(d,u)\}$

    $\{x = g(z),\ y = a,\ h(x,z) = h(d,u)\}$ — Regola (1) su $f(x,a) = f(g(z),y)$

    $\{x = g(z),\ y = a,\ h(g(z),z) = h(d,u)\}$ — Non vi sono regole applicabili e $h(g(z),z) = h(d,u)$ non è del tipo atteso (fallimento)

Risoluzione per refutazione in L_{PO}

- Problema: $\Gamma \models \varphi$?
  - Metodo di risoluzione (per refutazione):
    1) Insieme $\Gamma \cup \{\neg\varphi\}$ come punto di partenza
    2) Forma normale prenessa + skolemizzazione: $\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\})$
    3) Standardizzazione delle variabili
       (ogni quantificatore opera su una variabile diversa)
    4) Traduzione di $\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\})$ in forma a clausole (FC)
    5) Applicazione iterativa della regola di risoluzione:
       a) Selezione di due clausole da risolvere $\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}, \{\neg\alpha', \gamma_1, \gamma_2, \ldots, \gamma_m\}$
       b) Costruzione del MGU μ di α e α' (se μ esiste)
       c) Generazione del risolvente $\{\beta_1, \beta_2, \ldots, \beta_n, \gamma_1, \gamma_2, \ldots, \gamma_m\}$ e applicazione di μ

          $\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}, \{\neg\alpha', \gamma_1, \gamma_2, \ldots, \gamma_m\} \vdash \mu(\{\beta_1, \beta_2, \ldots, \beta_n, \gamma_1, \gamma_2, \ldots, \gamma_m\})$
  - Terminazione:
    - Quando si produce la clausola vuota $\{\}$ come risolvente (passo c - *successo*)
    - Quando non è più possibile applicare la regola di risoluzione (passo a - *fallimento*)
    - Il metodo può divergere (i.e. continuare all'infinito)

Esempio 9

- Problema: $\Gamma \models \varphi$?

  $\Gamma \equiv \{\forall x\,(Filosofo(x) \rightarrow Umano(x)), \forall x\,(Umano(x) \rightarrow Mortale(x)), Filosofo(Socrate)\}$

  $\varphi \equiv Mortale(Socrate)$

  – Procedura (risoluzione per refutazione):

  1: $\{\forall x\,(Filosofo(x) \rightarrow Umano(x)), \forall x\,(Umano(x) \rightarrow Mortale(x)), Filosofo(Socrate), \neg Mortale(socrate)\}$

  ($\Gamma \cup \{\neg\varphi\}$ è già in forma prenessa, non serve la skolemizzazione)

  2: $\{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\}, \{Filosofo(socrate)\}, \{\neg Mortale(socrate)\}\}$

  (standardizzazione e forma a clausole)

  3: Applicazione iterativa della regola di risoluzione:

  4: $\{Filosofo(socrate)\}, \{Umano(x), \neg Filosofo(x)\}, [x/socrate] \vdash \{Umano(socrate)\}$

  5: $\{Umano(socrate)\}, \{Mortale(y), \neg Umano(y)\}, [y/socrate] \vdash \{Mortale(socrate)\}$

  6: $\{Mortale(socrate)\}, \{\neg Mortale(socrate)\}, [] \vdash \{\}$

  (Successo)

Completezza del metodo di risoluzione

- Il metodo di risoluzione (per refutazione) è **corretto** in $\mathrm{L_{PO}}$
- Il metodo di risoluzione (per refutazione) è **completo** per $\mathrm{L_{PO}}$
  - Si intende che:
    - Se $\Gamma \models \varphi$, il metodo trova un sottoinsieme di $\mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$ che è finito e contraddittorio

      In generale, il metodo a risoluzione è più efficiente dell'enumerazione ricorsivadi $\mathrm{H}(\boldsymbol{sko}(\Gamma \cup \{\neg\varphi\}))$
    - Se $\Gamma \not\models \varphi$, il metodo può terminare (per fallimento) o divergere

- Limitazione
  - Tale risultato si applica a $\mathrm{L_{PO}}$ senza identità (i.e. senza $\mathrm{Ax7}$, $\mathrm{Ax8}$ e la regola semantica speciale per '=')
    - Si ritornerà su questo punto in seguito …
- *In aggiunta*
  - Applicata alle forme a clausole di Horn (in $\mathrm{L_{PO}}$) il metodo di risoluzione può fare di più …

Clausole di Horn in L_{PO}

- Definizione quasi identica al caso proposizionale
  - Forma a clausole (della skolemizzazione di un insieme di formule)
  - In ciascuna clausola occorre al massimo un atomo in forma positiva
  - Fatti, regole e goal
    - **Fatti**: clausola con un singolo atomo in forma positiva

      $\{Umano(socrate)\}$, $\{Pyramid(a)\}$, $\{Sorella(alba,madreDi(paolo))\}$
    - **Regole**: clausola di due o più atomi, uno in forma positiva

      $\{Umano(x), \neg Filosofo(x)\}$,
      $\forall x\ (Filosofo(x) \rightarrow Umano(x))$

      $\{\neg Femmina(x), \neg Genitore(k(x),x), \neg Genitore(k(y),y), Sorella(x,y)\}$
      $\forall x \forall y\ ((Femmina(x) \wedge \exists z\ (Genitore(z,x) \wedge Genitore(z,y))) \rightarrow Sorella(x,y))$

      $\{\neg Above(x,y), On(x,k(x))\}$, $\{\neg Above(x,y), On(j(y),y)\}$
      $\forall x \forall y\ (Above(x,y) \rightarrow (\exists z\ On(x,z) \wedge \exists v\ On(v,y)))$
    - **Goal**: clausola di atomi in forma negativa

      $\{\neg Umano(socrate)\}$

      $\{\neg Sorella(alba,x), \neg Sorella(x,paola)\}$
      Negazione di $\exists x\ (Sorella(alba,x) \wedge Sorella(x,paola))$

Clausole di Horn e modelli di Herbrand

- Corollario del teorema di Herbrand
  - Sia Γ un insieme di clausole di Horn, le seguenti affermazioni sono equivalenti:
    - Γ è soddisfacibile
    - Γ ha un modello di Herbrand
  - Non vale in generale: solo se Γ è un insieme clausole di Horn

- **Modello minimo** di Herbrand
  - Il modello minimo M_Γ è l'intersezione di tutti i modelli di Herbrand M_i di Γ :

    $M_\Gamma \equiv \cap_{\forall i} M_i$

- Teorema (van Emden e Kowalski, 1976)
  - Sia Γ un insieme di clausole e φ una clausola di Horn, le seguenti affermazioni sono equivalenti:
    - $\Gamma \models \varphi$
    - $\varphi \in M_\Gamma$
    - L'unione di tutte le clausole φ che sono conseguenza logica di Γ coincide con M_Γ

Programmi e modello minimo

- Teorema (Apt e van Emden, 1982)
  - Sia P un **programma** (= *un insieme di clausole di Horn*).
    Applicata a P, la procedura di risoluzione genera il modello minimo $\mathrm{M_P}$
    - La procedura termina se $\mathrm{M_P}$ è finito
  - Esempio:

    $\mathrm{P} \equiv \{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\},$
    $\{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Filosofo(aristotele)\}\}$
    - Applicando la procedura di risoluzione in modo esaustivo, si ottiene:

      $\mathrm{M_P} \equiv \{\{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Filosofo(aristotele)\},$
      $\{Umano(socrate)\}, \{Umano(platone)\}, \{Umano(aristotele)\},$
      $\{Mortale(socrate)\}, \{Mortale(platone)\}, \{Mortale(aristotele)\}\}$

      (assomiglia alla generazione di un database, ma il database è *implicito* ...)

Programmi e goal

- Un dimostratore di teoremi, applicato ad un programma logico P, risponde solo a domande del tipo "$\mathrm{P} \models \varphi$?"
  - Si rammenti che, se $\mathrm{P} \models \varphi$, allora $\mathrm{P} \cup \{\neg\varphi\}$ è insoddisfacibile

- Un sistema di programmazione logica può anche generare un particolare sottoinsieme di $\mathrm{M_P}$
  - Un goal $\{\neg\alpha_1, \neg\alpha_2, ..., \neg\alpha_m\}$, dove occorrono le variabili $x_1, x_2, ..., x_m$ equivale all'enunciato $\forall x_1 \forall x_2 \ldots \forall x_n (\neg\alpha_1 \vee \neg\alpha_2 \vee ... \vee \neg\alpha_m)$ che equivale a $\neg\exists x_1 \exists x_2 \ldots \exists x_n (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_m)$

  - Un sistema di programmazione logica genera tutte le **sostituzioni** $[x_1/t_1, x_2/t_2, ..., x_n/t_n]$ tali per cui $\mathrm{P} \cup \{\neg(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_m)[x_1/t_1, x_2/t_2, ..., x_n/t_n]\}$ è insoddisfacibile (vale a dire $\mathrm{P} \models (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_m)[x_1/t_1, x_2/t_2, ..., x_n/t_n]$) (vale a dire $(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_m)[x_1/t_1, x_2/t_2, ..., x_n/t_n] \in \mathrm{M_P}$)

Esempio 10

- Un programma logico P:

$$\mathrm{P} \equiv \{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\}, \{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Filosofo(aristotele)\}\}$$

$$\varphi \equiv \exists x\, Mortale(x)$$

$$\begin{aligned} \neg\varphi &\equiv \neg\exists x\, Mortale(x) \\ &\equiv \forall x\, \neg Mortale(x) \\ &\equiv \{\neg Mortale(x)\} \end{aligned}$$

(goal in forma di clausola di Horn)

  - Applicando la procedura di risoluzione in modo esaustivo
    - Si ottengono le sostituzioni:

      $\mathrm{K} \equiv \{[x/socrate], [x/platone], [x/aristotele]\}$

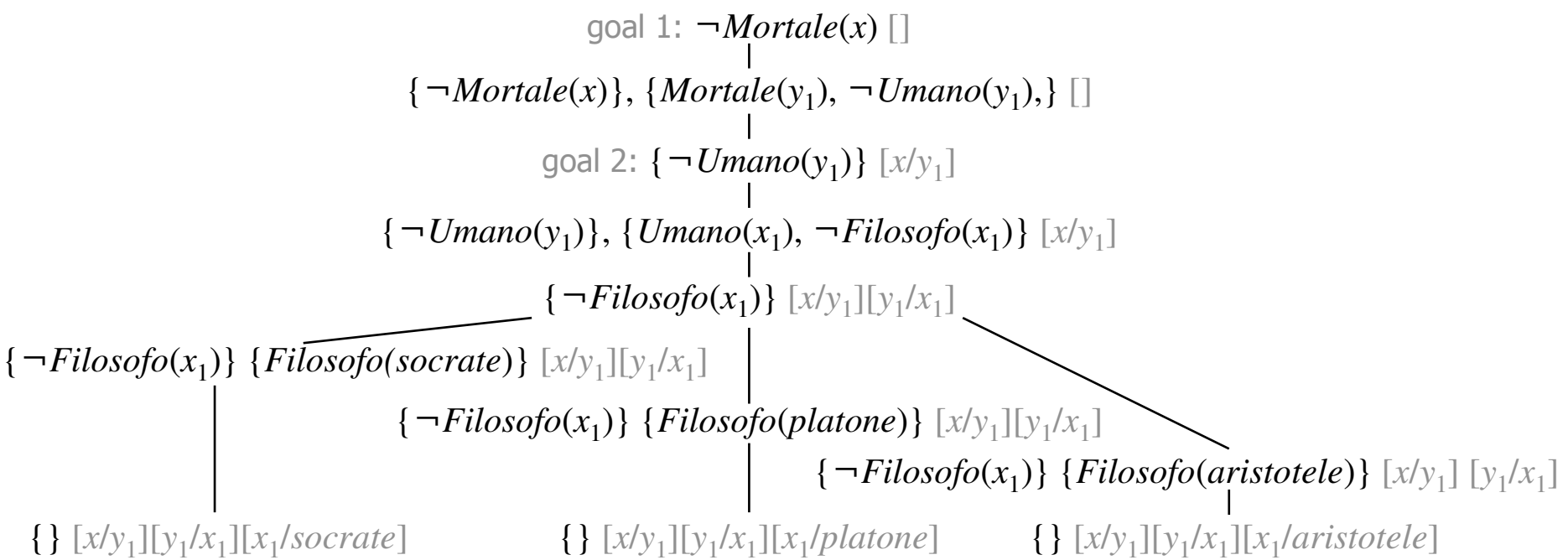
      (assomiglia alla query su un database, *implicito* ...)

Risoluzione SLD

- Un metodo per la risoluzione di programmi
  - S: *selection function*, una funzione di selezione degli atomi da unificare
    L: *linear resolution*, risoluzione lineare, cioè in sequenza
    D: *definite clause*, un altro nome per le clausole di Horn
- Descrizione
  - Goal: $\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_k$
  - Regole: $\beta \vee \neg\gamma_1 \vee \neg\gamma_2 \vee \ldots \vee \neg\gamma_n$
  - Fatti: δ
  - Procedura:
    - I goal vengono considerati secondo l'ordine definito dalla *selection function*
    - Per ciascun goal $\neg\alpha_i$ viene tentata la risoluzione (con unificazione) di tutte le regole (o fatti) che hanno un atomo positivo compatibile
    - Prima di tentare l'unificazione, vengono sempre ridenominate le variabili
    - Le risposte sono le assegnazioni che permettono di derivare la clausola vuota
    - L'insieme delle risposte (in teoria) è $\mathrm{M_P}$

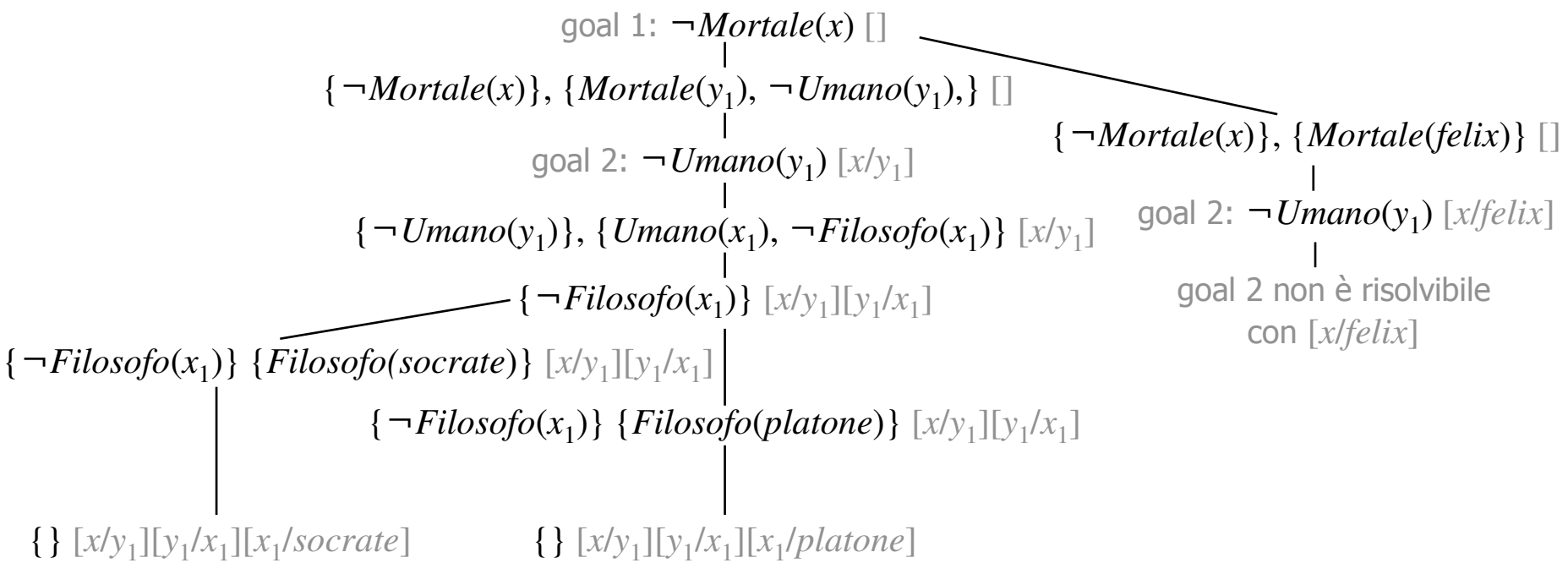
Alberi SLD

- Una traccia del metodo di risoluzione SLD
  - Esempio:
    - P ≡ $\{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\}, \{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Filosofo(aristotele)\}\}$
    - goal ≡ $\{\neg Mortale(x), \neg Umano(x)\}$ "Chi è mortale ed umano?"



Esempio 11

- Non tutti i rami SLD si chiudono con successo
  - P ≡ $\{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\}, \{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Mortale(felix)\}\}$
  - goal ≡ $\{\neg Mortale(x), \neg Umano(x)\}$ "Chi è mortale ed umano?"



Esempio 12

- Non tutti gli alberi SLD sono finiti
  - P ≡ $\{\{Loop(x), \neg Loop(x)\}\}$
  - goal ≡ $\{\neg Loop(x)\}$

goal: $\neg Loop(x)$ []

$\{\neg Loop(x)\}, \{Loop(x_1), \neg Loop(x_1),\}$ []

$\{\neg Loop(x_1)\}$ $[x/x_1]$

$\{\neg Loop(x_1)\}, \{Loop(x_2), \neg Loop(x_2),\}$ $[x/x_1]$

$\{\neg Loop(x_2)\}$ $[x/x_1]$ $[x_1/x_2]$

...

  - Data la ridenominazione delle variabili, le clausole progressivamente prodotte sono fbf *diverse* …

    … al contrario, nel caso ***proposizionale***, le fbf sono identiche ed il ciclo infinito non è possibile

SLD e programmazione logica

- **Insieme delle risposte**
  - Insieme di tutte le sostituzioni complete delle variabili,
    nei rami dell'albero SLD che si chiudono con successo (=con una clausola vuota)

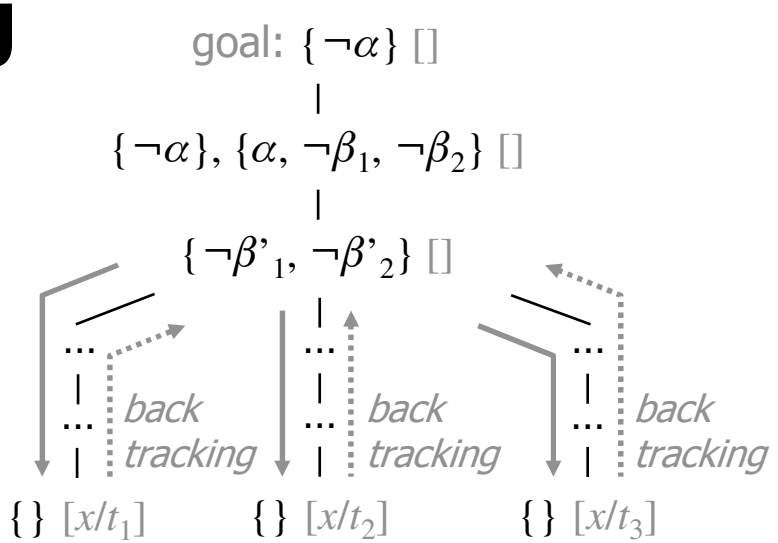
- Procedure effettive
  - L'ordine testuale del programma dovrebbe essere irrilevante, ma …
  - La *selection function* influisce sulla completezza del metodo SLD
    - in ***ampiezza*** (*breadth-first*)
    - in ***profondità*** (*depth-first*)
    - Il metodo SLD con selezione in *ampiezza* è **completo** (si dice anche SLD **fair**)

      Trova tutti i rami finiti (con successo o meno) dell'albero SLD
      (= *procedura completa di semi-decisione per* $\mathrm{P} \models \varphi$, con P e φ a clausole)
    - In pratica si utilizza la selezione in *profondità*

      (*Il metodo SLD non è completo - può divergere anche quando* $\mathrm{P} \models \varphi$)

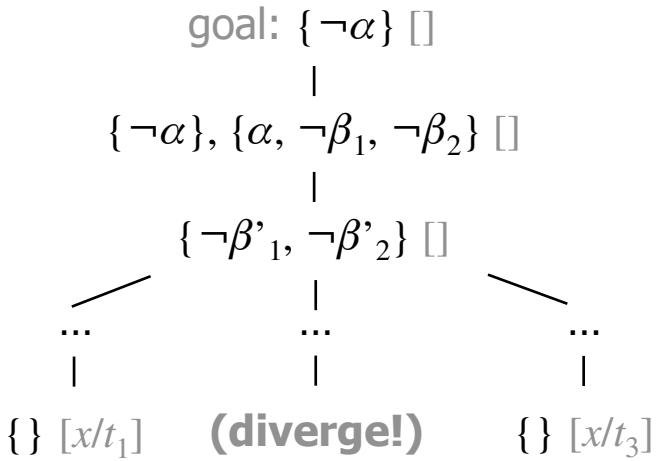
Risoluzione SLD in Prolog

- Scelte implementative
  - Risoluzione SLD *depth-first*
    - Più efficiente della *breadth-first*
    - si esplora una sola alternativa alla volta, e si risparmia memoria (*backtracking*)
    - E` **incompleta**: p.es. un ramo divergente impedisce di trovare tutte le risposte dei rami 'alla destra'



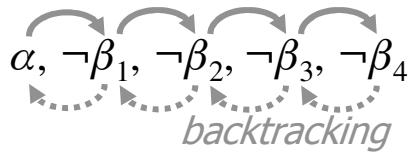
  - Selezione dei goal da sinistra e dall'alto
    - L'ordinamento testuale del programma diventa rilevante

Il metodo SLD *depth-first* non troverà $[x/t_3]$



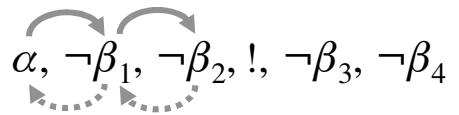
Controllo del *backtracking*

- ***cut*** **(!)**
  - Interrompe l'esplorazione dell'albero al primo successo



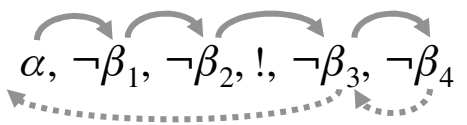
senza *cut*

  - Di fatto, 'taglia' il *backtracking*
    Prima del cut: backtracking libero



con *cut*

    Dopo il cut: backtracking libero solo fino al cut

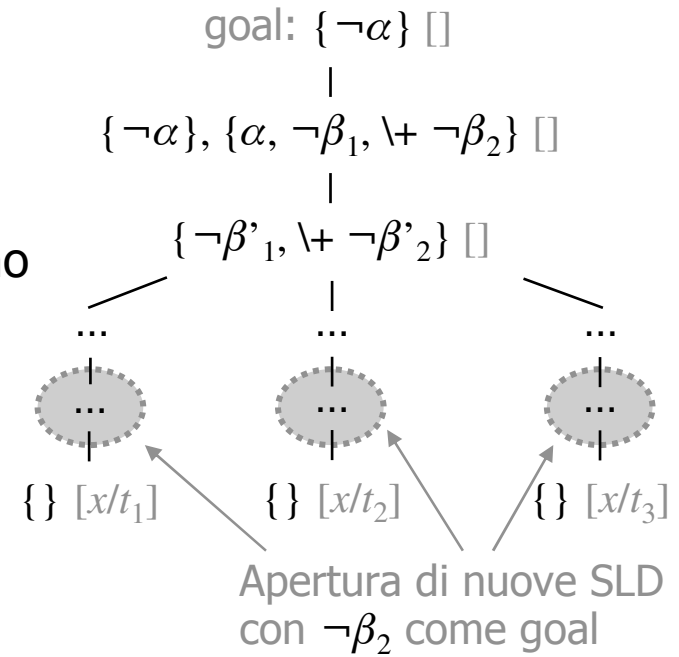


  - Il cut blocca il *backtracking* sui goal alla sinistra (incluso α: anche altre regole su α verranno ignorate)
- ***fail***
  - Forza il fallimento del ramo
- (Vedere esempio "library.pl")
  - *cut* e *fail* non hanno una semantica logica: sono un 'controllo di flusso'

SLDNF

- Clausole in forma negata (\+)
  - In generale, nelle clausole di Horn, le premesse di una regola devono essere in forma positiva
  - In Prolog le premesse in forma negativa sono interpretate come negazione per fallimento (*Negation as Failure – NAF, vedi oltre*)
    - Per il goal $\backslash+ \neg\beta_2$

      si apre una nuova procedura SLD: il goal ha successo se il goal $\neg\beta_2$ fallisce (*in un tempo finito*)



- (Vedere esempio "library.pl")

FAQ 4

- In Prolog, come viene rappresentata l'identità?
  - Il predicato '=' significa **unificabilità**

    $t_1 = t_2$ sse t_1 e t_2 sono unificabili
  - Il predicato 'is' significa **unificabilità** per i valori numerici
    - I valori e funzioni numeriche in Prolog sono trattate in modo speciale:

      $x \text{ is } (y + 1)$ sse i valori numerici sono identici
- Esempi

```
?- A is 22/7.
  A = 3.14286
?- (1 is (2-1)).
  Yes
?- (1 = (2-1)).
  No
```

- Attenzione:
  - Il predicato '==' significa **identità sintattica**

    $t_1 = t_2$ sse t_1 e t_2 sono letteralmente identici (senza ulteriori unificazioni)

Esempio 14

- Unificabilità ed identità non sono la stessa cosa
  - Esempio (Plaza, 1994)

```
p(X,Y):- \+ X=Y, q(X,Y).
q(a,a).
q(a,b).
```

X e **Y** sono sempre unificabili, p.es. [**X/Y**], quindi il goal negato fallisce

```
?- p(X,Y).
  No
```

```
p(X,Y):- \+ X==Y, q(X,Y).
q(a,a).
q(a,b).
```

X e **Y** sono termini diversi, quindi il goal negato ha sempre successo

```
?- p(X,Y).
  Yes [X=a,Y=a]
  Yes [X=a,Y=b]
```

Attenzione, però, all'ordine dei goal:

```
p(X,Y):- q(X,Y), \+ X=Y.
q(a,a).
q(a,b).
```

```
?- p(X,Y).
  Yes [X=a,Y=b]
```

FAQ 5

- Un altra particolarità: omissione dell'*occur check*
  - Regola (5) della procedura di costruzione del MGU

(5)	$x = t$ where x does not occur in t and x occurs elsewhere	*apply the substitution* $\{x/t\}$ *to all other equations*

  - Il test di occorrenza di x in t è il passo più dispendioso della procedura e viene solitamente disabilitato (o omesso) in Prolog

  – Risultato:

```
p(X,f(X)).
test :- p(Y,Y).
?- test.
Yes
```

`test` non è derivabile, in quanto non unificabile

(con un inesistente unificatore `[X=Y,Y=f(X)]`)

```
q(Y,f(Y)) :- q(Y,Y).
test2 :- q(X,X).
?- test2.
<infinite loop>
```

`test2` non è derivabile, in quanto non unificabile

(applica all'infinito la regola (5) con `[Y=f(Y)]`)

Relazione tra CWA, NAF e SLDNF

- *Closed-world assumption* (CWA)

  $\{\Gamma \not\models \varphi\} \vdash_{CWA} \neg\varphi$

  - Notare che $\Gamma \not\models \varphi$ non è decidibile in $\mathrm{L_{PO}}$, quindi nemmeno la relazione $\vdash_{CWA}$
- *Negation as Failure* (NAF)

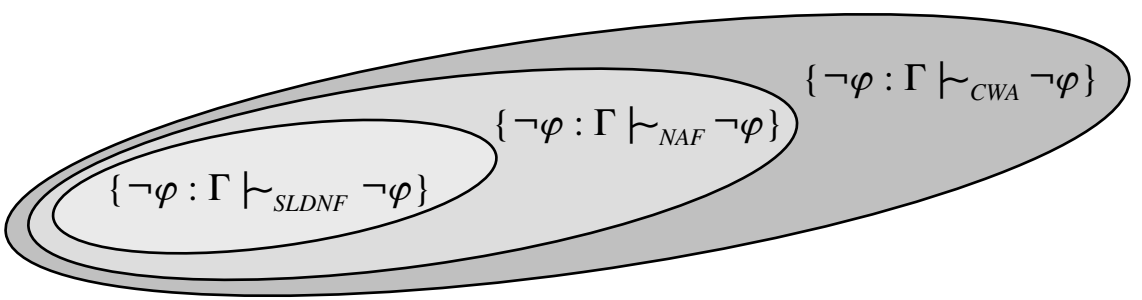
  $\{\varphi \in FF(\Gamma)\} \vdash_{NAF} \neg\varphi$

  - Se per φ esiste **un albero SLD finito** di $\Gamma \cup \{\neg\varphi\}$ che fallisce, si assume $\neg\varphi$ (solo una procedura SLD *fair*, cioè completa, lo trova certamente)
- SLDNF

  $\{\varphi \in FF_{SLD}(\Gamma)\} \vdash_{SLDNF} \neg\varphi$

  - Se la prova di $\Gamma \cup \{\neg\varphi\}$ fallisce con una strategia SLD si assume $\neg\varphi$ (si intende una strategia SLD qualsiasi, non necessariamente *fair*)

Relazioni di inclusione tra insiemi di clausole derivabili per CWA, NAF e SLDNF



Considerazioni sul Prolog

- (K. Apt, 2001)
  - Controllo 'idiosincratico' (*sic*)
    - Apprendere la scrittura di programmi in Prolog non è facile e lo stile è originale:

      le differenze sono più significative delle somiglianze rispetto ad altre forme di programmazione
    - E` possibile introdurre errori di difficile individuazione (p.es. usando il *cut*)
  - Standard limitato
    - L'ISO Prolog non prevede moduli ed oggetti (che comunque sono inclusi in molte implementazioni effettive)
  - Il più vicino alle idee della programmazione logica e dichiarativa
    - Costituisce la base di molti sistemi di ragionamento automatico
  - Comunque ancora in uso
    - Per essere un linguaggio progettato negli anni settanta …