Intelligenza Artificiale II

Risoluzione ed unificazione in L_{PO}

Marco Piastra

Automazione del calcolo logico-simbolico

Nella prospettiva del teorema di Herbrand

- Si cerca una procedura effettiva per dimostrare se $\Gamma \not\models \varphi$

  Si tratta di trovare un sottoinsieme finito e contraddittorio di $\mathrm{H}(sko(\Gamma \cup \{\neg\varphi\}))$

  Sperabilmente con un metodo più efficiente dell'enumerazione ricorsiva

  Nota: qualsiasi procedura può divergere (i.e. non terminare) se $\Gamma \not\models \varphi$

  (altrimenti L_{PO} sarebbe decidibile)

Forward chaining di base in L_{PO}

(vedi IA1)

- Descrizione:

Per stabilire se $\Gamma \models \varphi$

(Γ regole e fatti come clausole definite universalmente quantificate, φ fatto)

Si applicano a Γ le regole di inferenza *INST* e *GMP* in modo esaustivo

INST: $\forall x_1 \forall x_2 \ldots \forall x_n \varphi \vdash \varphi\, [x_1/c_1, x_2/c_2 \ldots x_n/c_n]$

GMP: $\alpha_1,\ \alpha_2,\ \ldots\ \alpha_n, \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta \ \vdash\ \beta$

($\alpha_1,\ \alpha_2,\ \ldots\ \alpha_n, \beta$ sono fbf *ground*, vale a dire dove non occorrono variabili)

Si ottiene un insieme Σ di fatti, $\Sigma = \{\psi : \Gamma \vdash \psi, \mathit{per\ INST\ e\ GMP}\}$

L'algoritmo termina con successo se $\varphi \in \Sigma$

Il metodo è completo per le clausole di Horn, in assenza di simboli funzionali

Un solo simbolo funzionale è sufficiente per causare divergenza

Tutti i fatti devono essere *base* (*ground*), cioè non contenere variabili

Il metodo prevede la generazione di tutte le *istanziazioni*

Esistono soluzioni migliorative: p.es. algoritmo Rete

Risoluzione proposizionale

Procedura per stabilire se $\Gamma \models \varphi$ (vedi IA1)

a) Refutazione $\Gamma \cup \{\neg\varphi\}$ e traduzione 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: $(\neg A \vee B) \wedge (C \vee \neg A)$

b) Traduzione di $\Gamma \cup \{\neg\varphi\}$ in 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: $\{\{\neg A, B\}, \{C, \neg A\}\}$

c) Applicazione esaustiva della regola di inferenza per risoluzione
   1) Selezione di due clausole da risolvere $\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}, \{\neg\alpha, \gamma_1, \gamma_2, \ldots, \gamma_m\}$
      La scelta di clausole e goal è irrilevante ai fini della completezza del metodo
   2) Generazione del risolvente
      $\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}, \{\neg\alpha, \gamma_1, \gamma_2, \ldots, \gamma_m\} \vdash \{\beta_1, \beta_2, \ldots, \beta_n, \gamma_1, \gamma_2, \ldots, \gamma_m\}$

Condizioni di terminazione:

1) Derivazione della clausola vuota (*successo*)
2) Non sono possibili nuove risoluzioni - *punto fisso* (*fallimento*)

Refutazione e forma a clausole in L_{PO}

a) Refutazione: $\Gamma \cup \{\neg\varphi\}$

b) Traduzione in forma normale prenessa e *skolemizzazione* $sko(\Gamma \cup \{\neg\varphi\})$:

Tutte le formule sono nella forma:

$\forall x_1 \forall x_2 \dots \forall x_n \psi$ (ψ non contiene quantificatori)

Essendo tutti universali, i quantificatori si possono omettere

c) Traduzione delle matrici ψ in FNC (con le stesse regole del caso proposizionale)

Si eliminano $\rightarrow$ e $\leftrightarrow$ in base alle regole di riscrittura

Si muove $\neg$ all'interno: $\neg(\alpha \wedge \beta) \Leftrightarrow (\neg\alpha \vee \neg\beta)$

$\neg(\alpha \vee \beta) \Leftrightarrow (\neg\alpha \wedge \neg\beta)$

Si distribuisce $\vee$: $((\alpha \wedge \beta) \vee \gamma) \Leftrightarrow ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$

e quindi in forma a clausole FC (con le stesse regole del caso proposizionale)

Esempio: traduzione di $\forall x\, (P(x) \rightarrow \exists y\, (Q(x,y) \wedge R(y)))$

1: $\forall x \exists y\, (P(x) \rightarrow (Q(x,y) \wedge R(y)))$ (forma normale prenessa)

2: $\forall x\, (P(x) \rightarrow (Q(x,k(x)) \wedge R(k(x))))$ (skolemizzazione, nuova funzione $k/1$)

3: $P(x) \rightarrow (Q(x,k(x)) \wedge R(k(x)))$ (eliminazione dei quantificatori)

4: $\neg P(x) \vee (Q(x,k(x)) \wedge R(k(x)))$ (equivalenza di $\rightarrow$)

5: $(\neg P(x) \vee Q(x,k(x))) \wedge (\neg P(x) \vee R(k(x)))$ (FNC, distributività di $\vee$)

6: $\{\neg P(x), Q(x,k(x))\}, \{\neg P(x), R(k(x))\}$ (FC)

Risoluzione in L_{PO}

Esempio: $\forall x\, Append(nil, x, x) \models \exists y\, \forall x\, Append(nil, cons(y, x), cons(a, x))$

1: $\forall x\, Append(nil, x, x), \neg\exists y\, \forall z\, Append(nil, cons(y, z), cons(a, z))$

(refutazione e *ridenominazione* delle variabile x)

2: $\forall x\, Append(nil, x, x), \forall y\, \exists z\, \neg Append(nil, cons(y, z), cons(a, z))$ (forma normale prenessa)

3: $\{Append(nil, x, x)\}, \{\neg Append(nil, cons(y, k(y)), cons(a, k(y)))\}$

($k/1$ funzione di Skolem, forma a clasuole)

(N.B. il Prolog *non* fa la skolemizzazione)

La coppia di **letterali**

$Append(nil, x, x), \neg Append(nil, cons(y, k(y)), cons(a, k(y))))$

... è compatibile (stesso predicato $Append/3$) ma i letterali hanno argomenti **diversi**

Se tuttavia si applica una sostituzione $\sigma = [x/cons(a, k(a)), y/a]$ si ottiene

$\{Append(nil, cons(a,k(a)), cons(a,k(a)))\}, \{\neg Append(nil, cons(a, k(a)), cons(a, k(a)))\}$

Da cui, per risoluzione, si ottiene la clausola vuota

- La sostituzione σ si dice **unificatore** delle due clausole

  va applicata integralmente a tutte e due le clausole da risolvere

Unificazione

- **Unificatore**

  Una sostituzione $\sigma = [x_1/t_1,\ x_2/t_2\ \ldots\ x_n/t_n]$ che rende risolvibili due letterali α e $\neg\beta$

  in simboli: $\sigma(\alpha) \equiv \sigma(\beta)$

  - Niente sostituzioni ricorsive: in x_i/t_i la variabile x_i **non può** comparire in t_i
  - Non sempre esiste un unificatore:
    ad esempio $\{P(g(x, f(a)), a)\}$ e $\{\neg P(g(b, f(w)), k(w))\}$ non sono unificabili

- **Unificatore più generale** (MGU - *most general unifier*)

  Se esiste un unificatore di α e $\neg\beta$ esiste anche un unificatore più generale MGU μ

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

  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 di Martelli e Montanari

Input: $\{s_1 = t_1, s_2 = t_2 \ldots s_n = t_n\}$ (equazioni tra termini: gli argomenti dei due letterali)

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

- Costruzione del MGU

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

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

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

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

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

$\{x = g(z), y = a, h(x, z) = h(d, u)\}$ 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 con unificazione in L_{PO}

Procedura per stabilire se $\Gamma \models \varphi$ in L_{PO}

a) Refutazione $\Gamma \cup \{\neg\varphi\}$,

b) Forma normale prenessa e skolemizzazione $sko(\Gamma \cup \{\neg\varphi\})$

c) Traduzione di $sko(\Gamma \cup \{\neg\varphi\})$ in FNC e quindi in forma a clausole (FC)

d) Applicazione iterativa della regola di risoluzione:

1) Selezione di due clausole da risolvere $\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}, \{\neg\alpha', \gamma_1, \gamma_2, \ldots, \gamma_m\}$

2) Standardizzazione delle variabili
(*ridenominazione* delle variabili in entrambe la clausole)

3) Costruzione del $\mathrm{MGU}\ \mu$ (se esiste) dei due letterali α e α'

4) Applicazione di μ alle due clausole e generazione del risolvente
$\{\beta_1, \beta_2, \ldots, \beta_n, \alpha\}[\mu], \{\neg\alpha', \gamma_1, \gamma_2, \ldots, \gamma_m\}[\mu] \ \vdash \ \{\beta_1, \beta_2, \ldots, \beta_n, \gamma_1, \gamma_2, \ldots, \gamma_m\}[\mu]$

Condizioni di terminazione:

1) Derivazione della clausola vuota (*successo*)

2) Non sono possibili nuove risoluzioni - *punto fisso* (*fallimento*)

Il metodo può anche divergere (i.e. continuare all'infinito)

Esempio

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

  Applicazione iterativa della regola di risoluzione

  4: $\{Filosofo(socrate)\}, \{Umano(x_1), \neg Filosofo(x_1)\}, [x_1/socrate] \vdash \{Umano(socrate)\}$

  5: $\{Umano(socrate)\}, \{Mortale(x_2), \neg Umano(x_2)\}, [x_2/socrate] \vdash \{Mortale(socrate)\}$

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

  (*Successo*)

Esempio
Come il precedente, diversa scelta della clausola da risolvere

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

$\varphi \equiv Mortale(Socrate)$

Procedura (*risoluzione per refutazione*):

1: ...

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

Applicazione iterativa della regola di risoluzione

4: $\{Mortale(x_1), \neg Umano(x_1)\}, \{Umano(x_2), \neg Filosofo(x_2)\}, [x_1/x_2] \vdash \{Mortale(x_2), \neg Filosofo(x_2)\}$

5: $\{Mortale(x_3), \neg Filosofo(x_3)\}, \{Filosofo(socrate)\}, [x_3/socrate] \vdash \{Mortale(socrate)\}$

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

(*Successo*)

Diversa scelta delle clausole da risolvere (rispetto al caso precedente)

Notare che nella risoluzione 4 si unificano due letterali con variabili ed il risolvente contiene variabili. Si tratta del *lifting*:

Esempio

- Il metodo di risoluzione può divergere

$\Gamma \equiv \forall x\,(Q(f(x)) \to P(x))$

$\varphi \equiv \exists x\,(P(f(x)) \land \neg Q(f(x)))$

$\neg\varphi \equiv \neg\exists x\,(P(f(x)) \land \neg Q(f(x)))$

$\equiv \forall x\,\neg(P(f(x)) \land \neg Q(f(x)))$

$\equiv \forall x\,(\neg P(f(x)) \lor Q(f(x)))$

Procedura (*risoluzione per refutazione*):

1: $\{\forall x\,(\neg Q(f(x)) \lor P(x)), \forall x\,(\neg P(f(x)) \lor Q(f(x)))\}$

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

2: $\{\{\neg Q(f(x)), P(x)\}, \{\neg P(f(x)), Q(f(x))\}\}$ (forma a clausole)

Applicazione iterativa della regola di risoluzione:

4: $\{\neg Q(f(x_1)), P(x_1)\}, \{\neg P(f(x_2)), Q(f(x_2))\}, [x_1/f(x_2)] \vdash \{\neg Q(f(f(x_2))), Q(f(x_2))\}$

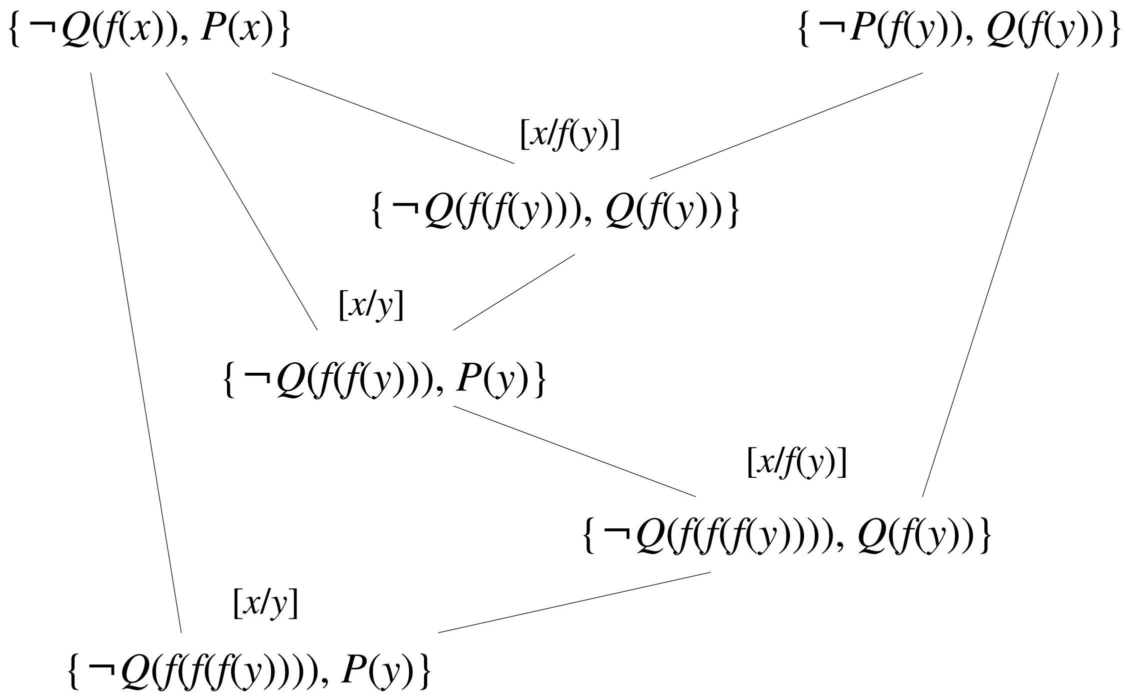
5: $\{\neg Q(f(x_3)), P(x_3)\}, \{\neg Q(f(f(x_4))), Q(f(x_4))\}, [x_3/x_4] \vdash \{\neg Q(f(f(x_4))), P(x_4)\}$

6: $\{\neg Q(f(f(x_5))), P(x_5)\}, \{\neg P(f(x_6)), Q(f(x_6))\}, [x_5/f(x_6)] \vdash \{\neg Q(f(f(f(x_6)))), Q(f(x_6))\}$

6: $\{\neg Q(f(x_7)), P(x_7)\}, , \{\neg Q(f(f(f(x_8)))), Q(f(x_8))\}, [x_7/x_8] \vdash \{\neg Q(f(f(f(x_8)))), P(x_8)\}$

...

Esempio
Vista alternativa dell'esempio precedente



.
.
.

La standardizzazione delle variabili spesso non viene mostrata, per semplicità

Completezza del metodo di risoluzione

- Il metodo di risoluzione con unificazione è **corretto** in L_{PO}
  - Se il metodo trova una contraddizione in $sko(\Gamma \cup \{\neg\varphi\})$ allora $\Gamma \models \varphi$

- Il metodo di risoluzione con unificazione è **completo** per L_{PO}?
  - Sì, nei limiti della semi-decidibilità di L_{PO} (Robinson, 1963)
    - Se $\Gamma \models \varphi$, il metodo trova una contraddizione in $sko(\Gamma \cup \{\neg\varphi\})$
      - In generale (ma non nel caso peggiore) il metodo a risoluzione è più efficiente dell'enumerazione ricorsiva e verifica di $\mathrm{H}(sko(\Gamma \cup \{\neg\varphi\}))$
      - Il vantaggio principale è il *lifting*
    - Se $\Gamma \not\models \varphi$, il metodo può divergere

  - Tuttavia, il metodo (inteso come algoritmo) potrebbe divergere anche quando $\Gamma \models \varphi$
    - Elementi critici:
      - Criterio di selezione delle clausole da risolvere
      - Strategia di esplorazione delle alternative
    - (studieremo solo il caso della risoluzione SLD)

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

$\{Umano(socrate)\}, \{Pyramid(x)\}, \{Sorella(alba, madreDi(paolo))\}$

Regole: clausola di due o più atomi, uno in forma positiva

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

$\forall x\ (Filosofo(x) \to Umano(x))$

$\{\neg Femmina(x), \neg Genitore(k(x),x), \neg Genitore(k(y),y), Sorella(x,y)\}$

$\forall x \forall y\ ((Femmina(x) \wedge \exists z\ (Genitore(z,x) \wedge Genitore(z,y))) \to Sorella(x,y))$

$\{\neg Above(x,y), On(x,k(x))\}, \{\neg Above(x,y), On(j(y),y)\}$

$\forall x \forall y\ (Above(x,y) \to (\exists z\ On(x,z) \wedge \exists v\ On(v,y)))$

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 di Horn 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 Π un **programma** (= *un insieme di clausole di Horn*).
  Applicata a Π, la procedura di risoluzione genera il modello minimo M_Π

  La procedura termina se M_Π è finito (raggiungimento del *punto fisso*)

  Esempio:

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

  Applicando la procedura di risoluzione in modo esaustivo, si ottiene:

  $$\begin{aligned} M_\Pi \equiv \quad & \{\{Mortale(x), \neg Filosofo(x)\}, \\ & \{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Filosofo(aristotele)\}, \\ & \{Umano(socrate)\}, \{Umano(platone)\}, \{Umano(aristotele)\}, \\ & \{Mortale(socrate)\}, \{Mortale(platone)\}, \{Mortale(aristotele)\}\} \end{aligned}$$

  (assomiglia alla generazione di un database, *implicitamente descritto* da Π...)

Programmi e goal

Un dimostratore di teoremi, applicato ad un programma logico Π, risponde solo a domande del tipo "$\Pi \models \phi$?"

Si rammenti che, se $\Pi \models \phi$, allora $\Pi \cup \{\neg\phi\}$ è insoddisfacibile

- Un sistema di programmazione logica è in grado di generare un particolare sottoinsieme di M_Π

Un goal $\{\neg\alpha_1, \neg\alpha_2, \dots, \neg\alpha_m\}$, dove occorrono le variabili $x_1, x_2, \dots, x_m$

equivale all'enunciato $\forall x_1 \forall x_2 \dots \forall x_n (\neg\alpha_1 \vee \neg\alpha_2 \vee \dots \vee \neg\alpha_m)$

che equivale a $\neg\exists x_1 \exists x_2 \dots \exists x_n (\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m)$

Un sistema di programmazione logica genera tutte le sostituzioni $[x_1/t_1, x_2/t_2, \dots, x_n/t_n]$ tali per cui $\Pi \cup \{\neg(\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m)[x_1/t_1, x_2/t_2, \dots, x_n/t_n]\}$ è insoddisfacibile

(vale a dire $\Pi \models (\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m)[x_1/t_1, x_2/t_2, \dots, x_n/t_n]$)

(vale a dire $(\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m)[x_1/t_1, x_2/t_2, \dots, x_n/t_n] \in M_\Pi$)

Il goal agisce da filtro, caratterizzando il sottoinsieme di M_Π

Goal diverso, sottoinsieme diverso

Esempio

- Un programma logico Π:

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

$$\phi \equiv \exists x\ Mortale(x)$$

$$\neg\phi \equiv \neg\exists x\ Mortale(x)$$

$$\equiv \forall x\ \neg Mortale(x)$$

$$\equiv \{\neg Mortale(x)\}$$ (goal in forma di clausola di Horn)

Applicando la procedura di risoluzione in modo esaustivo

Si ottengono le sostituzioni:

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

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*, clausole di Horn con esattamente un letterale positivo

- Descrizione

  Programma (*definite clauses*: regole + fatti): Π

  Regole: $\beta \vee \neg\gamma_1 \vee \neg\gamma_2 \vee \ldots \vee \neg\gamma_n$

  Fatti: δ

  Goal: $\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_k$

  Caratteristiche della 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 letterale positivo compatibile (*esplorazione delle alternative*)
  - Le risposte sono le assegnazioni che permettono di derivare la clausola vuota
  - L'insieme delle risposte è un sottoinsieme di M_Π

Alberi SLD

- Una traccia del metodo di risoluzione SLD

  Esempio:

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

  $goal \equiv \{\neg Mortale(x), \neg Umano(x)\}$ "Chi è mortale ed umano?"

$goal\ 1$: $\neg Mortale(x)$ []

$\{\neg Mortale(x)\}, \{Mortale(y_1), \neg Umano(y_1),\}$ []

$goal\ 2$: $\{\neg Umano(y_1)\}$ $[x/y_1]$

$\{\neg Umano(y_1)\}, \{Umano(x_1), \neg Filosofo(x_1)\}$ $[x/y_1]$

$\{\neg Filosofo(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Filosofo(x_1)\}$ $\{Filosofo(socrate)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Filosofo(x_1)\}$ $\{Filosofo(platone)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Filosofo(x_1)\}$ $\{Filosofo(aristotele)\}$ $[x/y_1]$ $[y_1/x_1]$

$\{\}$ $[x/y_1][y_1/x_1][x_1/socrate]$ $\{\}$ $[x/y_1][y_1/x_1][x_1/platone]$ $\{\}$ $[x/y_1][y_1/x_1][x_1/aristotele]$

Esempio

- Non tutti i rami SLD si chiudono con successo

$$\Pi \equiv \{\{Umano(x), \neg Filosofo(x)\}, \{Mortale(y), \neg Umano(y)\},$$
$$\{Filosofo(socrate)\}, \{Filosofo(platone)\}, \{Mortale(felix)\}\}$$
$$goal \equiv \{\neg Mortale(x), \neg Umano(x)\}$$

"Chi è mortale ed umano?"

goal 1: $\neg Mortale(x)$ []

$\{\neg Mortale(x)\}, \{Mortale(y_1), \neg Umano(y_1),\}$ []

goal 2: $\neg Umano(y_1)$ $[x/y_1]$

$\{\neg Umano(y_1)\}, \{Umano(x_1), \neg Filosofo(x_1)\}$ $[x/y_1]$

$\{\neg Filosofo(x_1)\}$ $[x/y_1][y_1/x_1]$

$\{\neg Filosofo(x_1)\}$ $\{Filosofo(socrate)\}$ $[x/y_1][y_1/x_1]$

$\{\}$ $[x/y_1][y_1/x_1][x_1/socrate]$

$\{\neg Filosofo(x_1)\}$ $\{Filosofo(platone)\}$ $[x/y_1][y_1/x_1]$

$\{\}$ $[x/y_1][y_1/x_1][x_1/platone]$

$\{\neg Mortale(x)\}, \{Mortale(felix)\}$ []

goal 2: $\neg Umano(y_1)$ $[x/felix]$

goal 2 non è risolvibile
con $[x/felix]$

Esempio

- Non tutti gli alberi SLD sono finiti

$$\Pi \equiv \{\{Loop(x), \neg Loop(x)\}\}$$
$$goal \equiv \{\neg Loop(x)\}$$

$$\text{goal: } \neg Loop(x)\ []$$
$$|$$
$$\{\neg Loop(x)\}, \{Loop(x_1), \neg Loop(x_1),\}\ []$$
$$|$$
$$\{\neg Loop(x_1)\}\ [x/x_1]$$
$$|$$
$$\{\neg Loop(x_1)\}, \{Loop(x_2), \neg Loop(x_2),\}\ [x/x_1]$$
$$|$$
$$\{\neg Loop(x_2)\}\ [x/x_1]\ [x_1/x_2]$$
$$|$$
$$\ldots$$

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)

- Metodo effettivo (*semantica procedurale*)

  S*election function* delle clausole

  Si usa (quasi) sempre la ***leftmost sub-goal first***, con sostituzione del ***sub-goal***

  S*trategia di esplorazione* delle alternative

  - 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* $\Pi \models \phi$, con Π e ϕ a clausole)

  In pratica si utilizza la selezione in ***profondità***

  (*Il metodo SLD non è completo - può divergere anche quando* $\Pi \models \phi$)

Risoluzione SLD in Prolog

- Metodo effettivo

  Selection function: leftmost sub-goal first

  Esplorazione *depth-first* delle alternative

  Si esplora una sola alternativa alla volta, e si risparmia memoria (*backtracking*)

  E` una strategia **incompleta**:

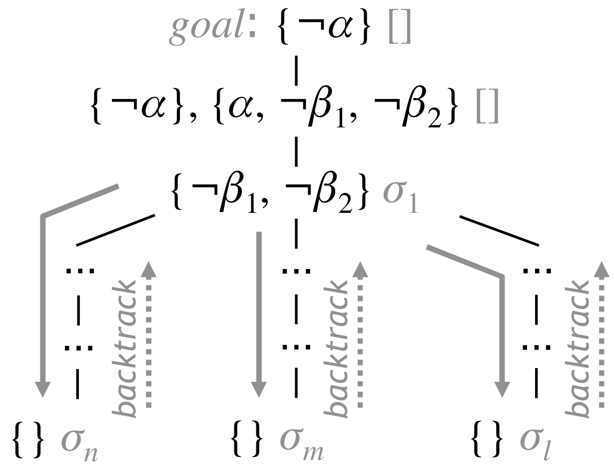
  Un ramo divergente impedisce di trovare tutte le risposte dei rami 'alla destra'

  Scelta tra risoluzioni alternative

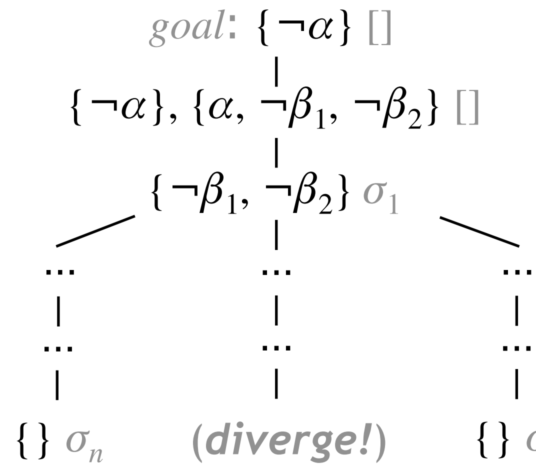
  (= *ordine di esplorazione dei sotto-alberi*)

  Ordine di definizione della clausola applicata

  (≈ quella che compare prima nel file)



Risposte diverse: σ_n σ_m σ_l



Il metodo SLD *depth-first* non troverà la risposta σ_l

Controllo del *backtracking*

- ***cut*** (!)

  Interrompe l'esplorazione dell'albero al primo successo

  Di fatto, 'taglia' il *backtracking*
  - Prima del cut: backtracking libero
  - Dopo il cut: backtracking libero solo fino al cut

- ***fail***

  Forza il fallimento del ramo

  cut e *fail* non hanno una semantica logico-dichiarativa: sono un 'controllo di flusso'

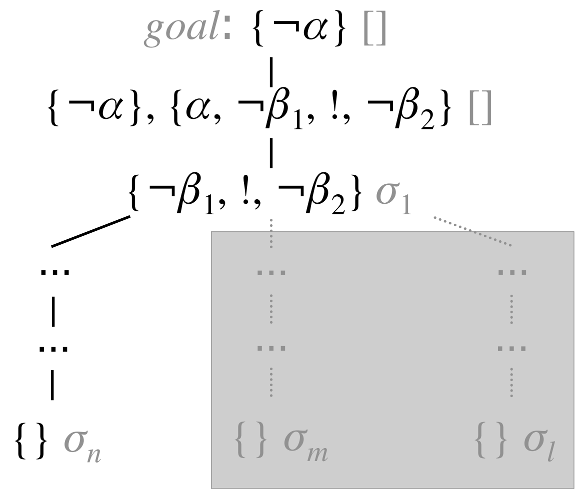
$\alpha, \neg\beta_1, \neg\beta_2, \neg\beta_3, \neg\beta_4$ **senza *cut***

backtrack

$\alpha, \neg\beta_1, \neg\beta_2, !, \neg\beta_3, \neg\beta_4$ **con *cut***

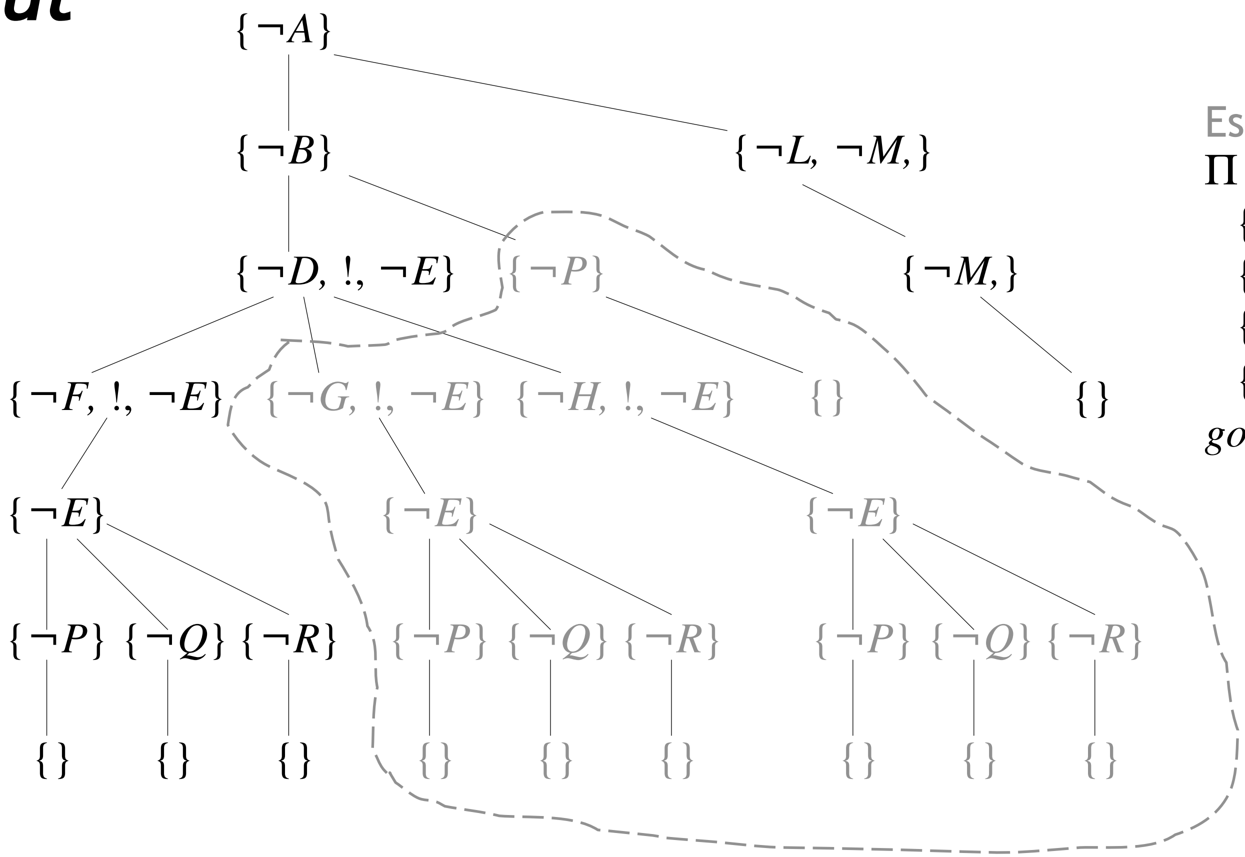
$\alpha, \neg\beta_1, \neg\beta_2, !, \neg\beta_3, \neg\beta_4$

Dal punto di *cut*, il *backtrack* torna alla radice dell'albero SLD (e si arresta).



Per effetto del *cut*, la parte in grigio dell'albero SLD non viene esplorata.

Cut



Questa parte dell'albero SLD
non viene espansa a causa del *cut*

Esempio:

$\Pi \equiv \{\{A, \neg B\}, \{A, \neg L, \neg M,\}$
$\{B, \neg D, !, \neg E\}, \{B, \neg P\},$
$\{D, \neg F\}, \{D, \neg G\}, \{D, \neg H\},$
$\{E, \neg P\}, \{E, \neg Q\}, \{E, \neg R\}, \{L\},$
$\{M\}, \{F\}, \{G\}, \{H\}, \{P\}, \{Q\}, \{R\}\}$
$goal \equiv \{\neg A\}$

Il cut inibisce il *backtracking*
a partire dal goal genitore
(= che attiva la regola che
contiene il *cut*)

Negazione come fallimento

- Clausole in forma negata (\+)

  In generale, nelle clausole di Horn, le premesse di una regola devono essere in forma positiva

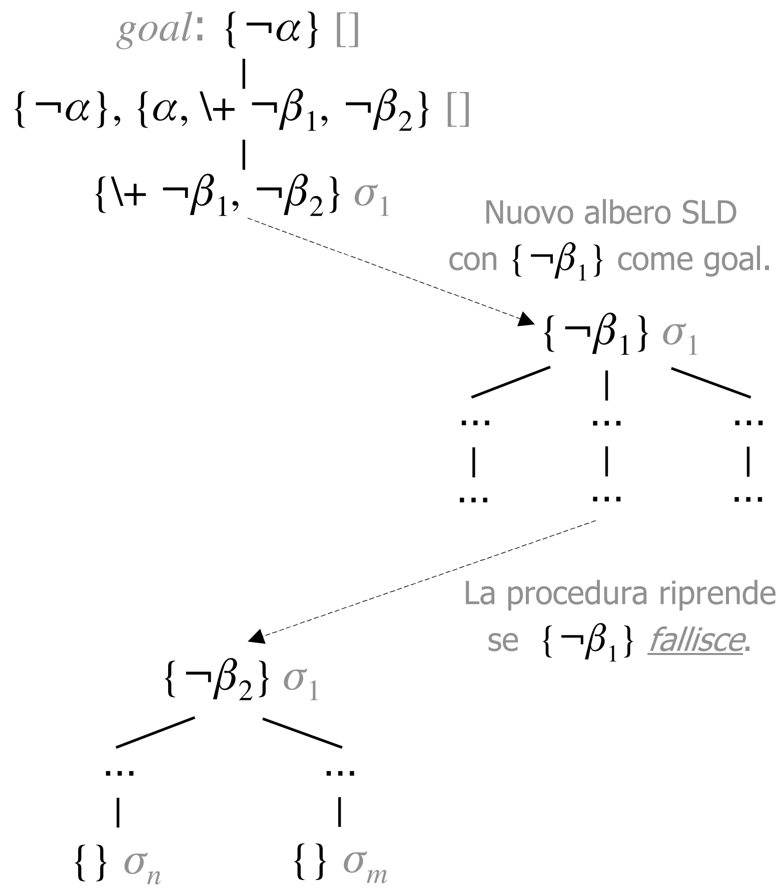
  In Prolog le premesse in forma negativa sono interpretate come negazione per fallimento (*Negation as Failure – NAF, vedi oltre*)

  Per il goal \+ $\neg\beta_1$

  si apre una nuova procedura SLD:
  il goal ha successo se il goal $\neg\beta_1$ fallisce (*senza divergere*)

  Risoluzione SLDNF
  (*Negation as Failure*)

  (Vedere esempio "library.pl")



Identità

- 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 **`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 ***equivalenza simbolica***
    - t_1 **==** t_2 sse t_1 e t_2 sono lessicalmente identici

Esempio

- 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]
```

Unificazione e *occur check*

- Un altra particolarità del Prolog: 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 la regola* (5) *con* **[Y/f(Y)]**)