Agenti logici: sistemi a regole

Regole all'indietro e programmazione logica
Regole in avanti e basi di dati deduttive

Maria Simi
a.a. 2011-2012

Risoluzione efficiente

- Il *metodo di risoluzione* per il FOL
  - KB in forma a clausole
  - Unificazione e regola di risoluzione (strategia di "lifting" rispetto a quella per PROP)
- Come si può rendere più efficiente?
  - Strategie di risoluzione: tecniche per esplorare in maniera efficiente il grafo di risoluzione, possibilmente senza perdere completezza

Strategie di risoluzione

- Si distingue tra [Genesereth-Nilsson]:
  - Strategie di cancellazione
  - Strategie di restrizione
  - Strategie di ordinamento

Strategie di cancellazione

- Si tratta di rimuovere dalla KB (ai fini della dimostrazione) certe clausole che non potranno essere utili nel processo di risoluzione
  1. Clausole con *letterali puri*
  2. *Tautologie*
  3. Clausole *sussunte*

Cancellazione di letterali puri

- Clausole con *letterali puri*: quelli che non hanno il loro negato nella KB

  Es. {¬P, ¬Q, R} {¬P, S} {¬Q, S} {P} {Q} {¬R}

  Le clausole con letterali puri non potranno mai essere risolte con altre clausole per ottenere { }

Cancellazione di tautologie

- *Tautologie*: clausole che contengono due letterali identici e complementari

  Es. {P(A), ¬P(A), …} {P(x), Q(y), ¬Q(y)}

  La loro rimozione non influenza la soddisfacibilità.
- *Nota*: non basta che siano unificabili e di segno opposto

  Es. {¬P(A), P(x)} {P(A)} {¬P(B)} è insoddisfacibile
  {P(A)} {¬P(B)} non lo è
- Le tautologie possono essere generate ⇒ controllo da fare ad ogni passo

Cancellazione di clausole sussunte

3. Eliminazione di *clausole sussunte* (implicate)
   - Es. P(x) *sussume* P(A), P(A) *sussume* P(A) ∨ P(B)
   - In generale: α *sussume* β sse $\exists\sigma\ \alpha\sigma \subseteq \beta$
     se un'istanza di α (con la sost. σ) è un sottoinsieme di β

   Es. {P(x), Q(y)} *sussume* {P(A), Q(v), R(w)} infatti
   {P(x), Q(y)}{x/A, y/v}={P(A), Q(v)}

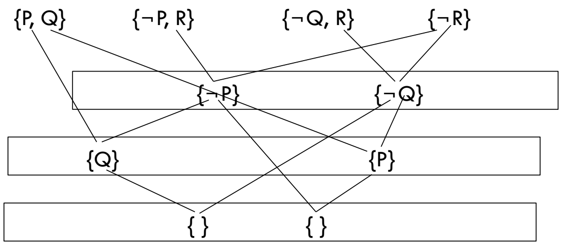
- β può essere ricavata da α. Quindi β può essere eliminata senza perdere soluzioni.
- Le clausole sussunte possono essere generate.

Strategie di restrizione

- Ad ogni passo si sceglie tra un sottoinsieme delle possibili clausole
- Tra le strategie di restrizione possibili:
  1. Risoluzione unitaria
  2. Risoluzione da input
  3. Risoluzione lineare
  4. Risoluzione lineare da input
  5. Risoluzione guidata dal goal

Risoluzione unitaria

- Risoluzione unitaria: almeno una delle due clausole è *unitaria* (contiene un solo letterale)



Risoluzione unitaria: completa?

- Facile da implementare, si converge rapidamente
- *Problema*: la strategia non è completa

  Esempio. {P, Q} {¬P, Q} {P, ¬Q} {¬P, ¬Q} $\vdash_{RES}$ { }

  ma non con risoluzione unitaria
- La strategia è completa per clausole Horn.

  Clausole Horn: clausole con al più un letterale positivo
- Nota: {P, Q} non è una clausola Horn

Risoluzione da input

- Una delle clausole appartiene alla KB iniziale

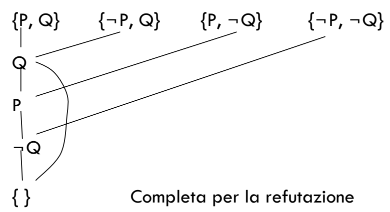
  Teorema: c'è una risoluzione da input sse ce n'è una unitaria (metodi diversi ma equivalenti)
- *Corollario*: risoluzione da input non completa, ma completa per clausole Horn.

  Es. {P, Q} {¬P, Q} {P, ¬Q} {¬P, ¬Q} non Horn

  … e la clausola vuota non può essere generata da input.

Risoluzione lineare

- Ultima clausola generata con una clausola da input oppure una clausola antenata.
- Generalizzazione della risoluzione da input, con in più il vincolo di linearità



Completa per la refutazione

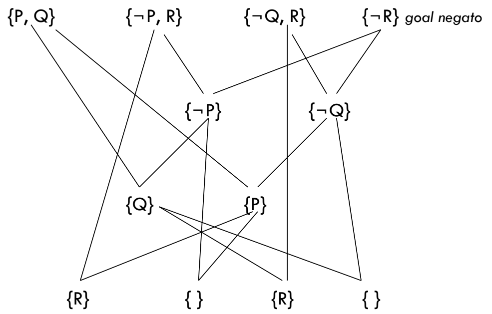
Risoluzione lineare da input

- Ultima clausola generata più una da input
- Completa, ma solo per clausole Horn

Risoluzione guidata dal goal

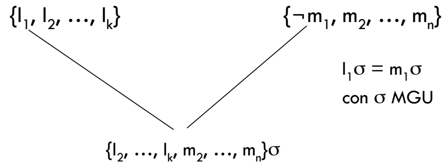
- Insieme di supporto: un sotto-insieme Γ della KB responsabile dell'insoddisfacibilità
- Almeno una delle due clausole appartiene a questo insieme o a suoi discendenti
- Tipicamente, assumendo la KB iniziale consistente, si sceglie come insieme di supporto iniziale il negato della clausola goal
- … è come procedere all'indietro dal goal

Risoluzione all'indietro dal goal: esempio

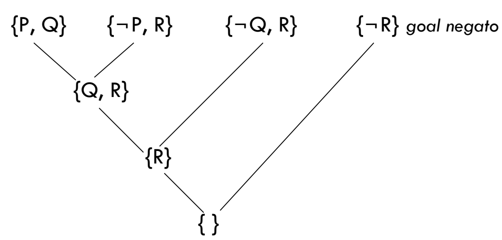


Risoluzione ordinata

- Ogni clausola è un insieme ordinato di letterali e si possono unificare solo i letterali di testa delle clausole
- L'ordinamento deve essere rispettato nel risolvente



Risoluzione ordinata: esempio



La risoluzione ordinata è completa per clausole Horn

Il sottoinsieme "a regole" del FOL

- *Clausole Horn definite: esattamente un letterale positivo*
- Possono essere riscritte come fatti e regole:

$\neg P_1 \vee \ldots \vee \neg P_k \vee Q$

$\neg(P_1 \wedge \ldots \wedge P_k) \vee Q$

$P_1 \wedge \ldots \wedge P_k \Rightarrow Q$ *regola*

Q *fatto*

Sistemi a regole logici

- KB a regole
  - Fatti: letterali positivi. Es. *p*
  - Regole: $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$
- Se la KB contiene solo clausole Horn *definite* i meccanismi inferenziali sono molto più semplici, il processo molto più "guidato" senza rinunciare alla completezza.
- Nota: è restrittivo. Non coincide con FOL.

Uso delle regole in avanti e all'indietro

- Concatenazione all'indietro (*Backward Chaining*): un'istanza di ragionamento guidato dall'obiettivo
  - Le regole sono applicate alla rovescia
  - Programmazione logica (PROLOG)
- Concatenazione in avanti (*Forward Chaining*): un'istanza di ragionamento|ricerca guidato dai dati
  - Le regole sono applicate nel senso "antecedente-conseguente"
  - Basi di dati deduttive e sistemi di produzione

Programmazione logica

- I programmi logici sono KB costituiti di clausole Horn definite espressi come fatti e regole, con una sintassi alternativa

  {A}

  $\{A, \neg B_1, \neg B_2, \ldots, \neg B_n\}$ $[B_1 \wedge B_2 \wedge \ldots \wedge B_n \Rightarrow A]$

  diventano

  A. fatto

  A :- B_1, B_2, … , B_n. regola, con *testa* A, il conseguente
- *Altre convenzioni*: in PL le variabili sono indicate con lettere maiuscole, le costanti con lettere minuscole

Programmi logici

- *Interpretazione dichiarativa* di una regola

  A :- B_1, B_2, … , B_n (A *testa*, B_1, B_2, … B_n *corpo*)

  A è vero se sono veri B_1, B_2, … B_n
- *Interpretazione procedurale*: la testa può essere vista come una chiamata di procedura e il corpo come una serie di procedure da eseguire in sequenza
- *Clausole goal*

Se $B_1 \wedge B_2 \wedge \ldots \wedge B_n$	è il goal
$\neg(B_1 \wedge B_2 \wedge \ldots B_n) \vee False$	è il goal negato, ovvero
$B_1 \wedge B_2 \wedge \ldots B_n \Rightarrow False$	che viene scritto
:- B_1, B_2, … , B_n	omettendo il conseguente

Esempio di KB come programma logico

1. Genitore(X, Y) :- Padre(X, Y).
2. Genitore(X, Y) :- Madre(X, Y).
3. Antenato(X, Y) :- Genitore(X, Y).
4. Antenato(X, Y) :- Genitore(X, Z), Antenato(Z, Y).
5. Padre(gio, mark).
6. Padre(gio, luc).
7. Madre(lia, gio).

8. :- Antenato(lia, mark) *goal negato*

Risoluzione SLD

- La risoluzione SLD (Selection Linear Definite-clauses) è una strategia *ordinata*, basata su un *insieme di supporto* (la clausola goal), *lineare da input*.
- La risoluzione SLD è completa per clausole Horn.

Alberi di risoluzione SLD

- Dato un programma logico P, l'albero SLD per un goal G è definito come segue:
  - ogni nodo dell'albero corrisponde a un goal [congiuntivo]
  - la radice è :-G, il nostro goal
  - sia :- $G_1, G_2, \dots, G_k$ un nodo dell'albero; il nodo ha tanti discendenti quanti sono i fatti e le regole in P la cui testa è unificabile con G_1

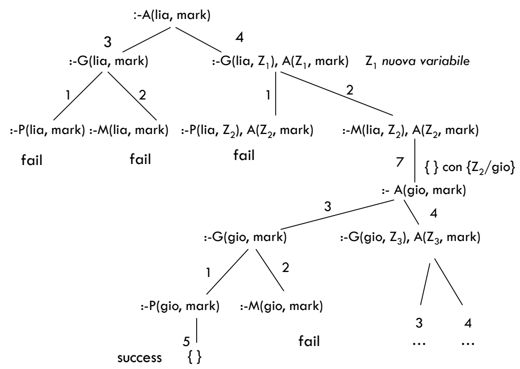
    Se A :- $B_1, \dots, B_k$ e A è unificabile con G_1 il discendente è il goal :- $(B_1, \dots, B_k, G_2, \dots, G_k)\gamma$ con γ = MGU(A, G_1)
  - i nodi che sono clausole vuote sono successi
  - i nodi che non hanno successori sono fallimenti

Esempio di albero SLD: il programma

1. Genitore(X, Y) :- Padre(X, Y).
2. Genitore(X, Y) :- Madre(X, Y).
3. Antenato(X, Y) :- Genitore(X, Y).
4. Antenato(X, Y) :- Genitore(X, Z), Antenato(Z, Y).
5. Padre(gio, mark).
6. Padre(gio, luc).
7. Madre(lia, gio).

8. :- Antenato(lia, mark). *goal negato*

Albero SLD per :- Antenato(lia, mark)



Risoluzione SLD

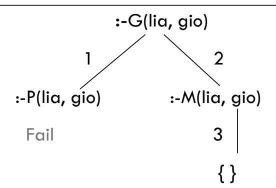
- La strategia è completa per clausole Horn definite e quindi, se P $\cup\{\neg G\}$ è insoddisfacibile, allora una delle foglie deve essere la clausola vuota (successo)
- Non è restrittivo andare in ordine nel risolvere i sottogoal in and.
- La sostituzione corrispondente è la *risposta calcolata*

Strategia di visita dell'albero SLD e PROLOG

- A seconda di come visito l'albero potrei anche non trovare la clausola vuota. La strategia di ricerca può essere responsabile dell'incompletezza.
- In PROLOG, il più famoso linguaggio di programmazione logica, la visita dell'albero di risoluzione avviene con una ricerca in profondità, con *backtracking* in caso di fallimento
- Su richiesta si trovano tutte le soluzioni.
- Quindi la strategia di PROLOG non è completa
- PROLOG omette l'*occur check* per motivi di efficienza
- Le regole vengono applicate nell'ordine in cui sono immesse

PROLOG e domande del tipo "si-no"

I numeri corrispondono all'ordine di visita



:- G(lia, gio) → SI

:- G(lia, pete) → NO

Assunzione di mondo chiuso

PROLOG con domande del tipo "trova"

:- P(X, mark)
chi è il padre di Mark?
X=gio

:- P(X, mark)
1
{ } con {X/gio}

:- P(gio, X)
chi sono i figli di Gio?
X=mark;
X=luc.

P(gio, X)
1 2
{ } con {X/mark} { } con {X/luc}

Altre domande ...

- Chi è figlio di chi?
  :- G(X, Y).
- Quali sono i fratelli (coloro che hanno lo stesso genitore)?
  :- G(X, Y), G(X, Z).
- Chi sono i nipoti di Lia (in quanto nonna)?
  :- G(lia, X), G(X, Y).

Incompletezza

Supponiamo di avere un programma leggermente diverso:

1. G(X, Y) :- P(X, Y)
2. G(X, Y) :- M(X, Y)
4. A(X, Y) :- A(Z, Y), G(X, Z)
3. A(X, Y) :- G(X, Y)
5. P(gio, mark)
6. P(gio, luc)
7. M(lia, gio)

Nota. Abbiamo scambiato la regola 3 con la 4 e i due letterali nel corpo della 4 tra di loro

Goal:
:- A(lia, mark)
:- A(Z_1, mark), G(lia, Z_1)
:- A(Z_2, mark), G(Z_1, Z_2)
:- A(Z_3, mark), G(Z_2, Z_3)
...

Si finisce in un cammino infinito e non si trova mai la soluzione

Estensioni: le liste

- Prolog ammette anche le liste come strutture dati.
  - [E|L] indica una lista il cui primo elemento è E e il resto è L; [ ] lista vuota.
- Concatenazione di liste:
  concatena ([ ], Y, Y).
  concatena([A|X], Y, [A|Z]) :- concatena(X, Y, Z).

Negazione come fallimento finito

- Orfano(X) :- not Padre(Y, X)
- Se :- Padre(Y, X) fallisce (non si trovano padri), la risposta è SI
- Non coincide con la negazione logica:
  - KB $\nvdash$ Padre(joe, mark) piuttosto che
    KB $\vdash \neg$Padre(joe, mark)
- È una forma di ragionamento non monotòno e fa uso della assunzione di mondo chiuso.

Estensioni: semplice aritmetica

- Operatori infissi predefiniti: +, -, *, /, //, ** ...
- Espressioni numeriche: il predicato "A is 2*3" è vero se A ha un valore e il valore di A è 6
- Operatori di confronto: >, <, >=, =<, =:=, =\= forzano la valutazione, variabili ok purché instanziate
  Nota: 2+1 = 1+2 unificazione fallisce; 2+1 =:= 1+2 ok
- Esempio:
  max(X, Y, Y) :- X =< Y.
  max(X, Y, X) :- X>Y.
  Molto elegante, ma presuppone che i primi due argomenti nel goal, X e Y, siano numeri

Per provare …

- SWI Prolog
  http://www.swi-prolog.org/

Sistemi a regole in avanti

- *Modus ponens generalizzato*

$$\frac{p_1'\ p_2' \ldots p_n' \quad (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{(q)\ \theta}$$

  dove θ=MGU(p_i', p_i), per ogni I
- Regola corretta:
  - Si istanziano gli universali
  - Si istanziano le regole
  - Si applica il Modus Ponens classico

Esempio di MP generalizzato

- Supponiamo
  King(John)
  Greedy(y)
  King(x) ∧ Greedy(x) ⇒ Evil(x)

$$\frac{King(John),\ Greedy(John),\ King(John) \wedge Greedy(John) \Rightarrow Evil(John)}{Evil(John)}$$

  con θ={x/John, y/John}

Esempio di concatenazione in avanti

È un crimine per un Americano vendere armi a una nazione ostile. Il paese Nono, un nemico dell'America, ha dei missili, e tutti i missili gli sono stati venduti dal colonnello West, un Americano.

Dimostrare: che *West* è *un criminale*

Formalizzazione

1. Americano(x) ∧ Arma(y) ∧ Vende(x, y, z) ∧ Ostile(z) ⇒ Criminale(x)
2. ∃x Possiede(Nono,x) ∧ Missile(x)
   Possiede(Nono, M_1) ∧ Missile(M_1)
3. Missile(x)∧Possiede(Nono,x)⇒Vende(West, x, Nono)
4. Missile(x) ⇒ Arma(x)
5. Nemico(x, America) ⇒ Ostile(x)
6. Americano(West)
7. Nemico(Nono, America)

Concatenazione in avanti

- Un semplice processo inferenziale applica ripetutamente il Modus Ponens generalizzato per ottenere nuovi fatti fino a che
  - si dimostra quello che si desidera
  - nessun fatto nuovo può essere aggiunto
- Una strategia di ricerca sistematica in ampiezza

Concatenazione in avanti: esempio

I iterazione:

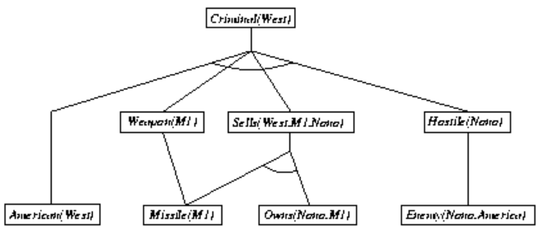
2. Possiede(Nono, M_1) ∧ Missile(M_1)
3. Missile(x)∧Possiede(Nono,x) ⇒ Vende(West,x,Nono)
   - La regola 3 è soddisfatta con {x/M_1} e viene aggiunto
   - Vende(West, M_1, Nono)
4. Missile(x) ⇒ Arma(x)
   - La regola 4 è soddisfatta con {x/M_1} e viene aggiunto
   - Arma(M_1)
5. Nemico(x, America) ⇒ Ostile(x)
6. Nemico(Nono, America)
   - La regola 5 è soddisfatta con {x/Nono} e viene aggiunto
   - Ostile(Nono)

Concatenazione in avanti: esempio

II iterazione

1. Americano(x) ∧ Arma(y) ∧ Vende(x, y, z) ∧ Ostile(z) ⇒ Criminale(x)
   - La regola 1 è soddisfatta con {x/West, y/M_1, z/Nono)}
   - Criminale(West) viene aggiunto.

La dimostrazione in avanti



Analisi di FOL-FC-Ask

- Corretta perché il MP generalizzato è corretto
- Completa per KB di clausole Horn definite
  - Completa e convergente per calcolo proposizionale e per KB di tipo DATALOG (senza funzioni) perché la chiusura deduttiva è un insieme finito
  - Completa anche con funzioni ma il processo potrebbe non terminare (semidecidibile)
- Il metodo descritto è sistematico ma non troppo efficiente

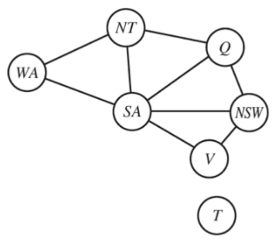
FC efficiente

- Ordinamento dei congiunti:
  conviene soddisfare prima i congiunti con meno istanze nella KB (come per i CSP)

Missile(x) ∧ Possiede(Nono,x) ⇒Vende(West,x,Nono)

Tipi di missile << cose possedute

Relazione con CSP



$$\begin{aligned}&Diff(wa,nt) \wedge Diff(wa,sa) \wedge \\ &\quad Diff(nt,q) \wedge Diff(nt,sa) \wedge \\ &\quad Diff(q,nsw) \wedge Diff(q,sa) \wedge \\ &\quad Diff(nsw,v) \wedge Diff(nsw,sa) \wedge \\ &\quad Diff(v,sa) \Rightarrow Colorable()\end{aligned}$$

$Diff(Red,Blue)$ $Diff(Red,Green)$
$Diff(Green,Red)$ $Diff(Green,Blue)$
$Diff(Blue,Red)$ $Diff(Blue,Green)$

FC incrementale

- ogni nuovo fatto inferito al tempo t deve essere dedotto usando almeno un fatto dedotto al tempo t-1
- si possono guardare solo le regole che hanno come premesse unificabili con fatti aggiunti nell'ultima iterazione
- indicizzare le regole sui fatti
- altre ottimizzazioni presenti nell'algoritmo RETE …

FC efficiente: ridurre deduzioni irrilevanti

- Un modo per evitare di ricavare fatti irrilevanti
- Lavorando all'indietro dal goal, non c'è questo problema
- Si fa una specie di pre-processing per individuare le regole che servono, procedendo all'indietro dal goal

FC efficiente: l'idea del *magic set*

- Goal: Criminal(West) $KB \leftarrow KB \cup \{Magic(West)\}$
- Riscrittura regole:
  - Magic(x) ∧ Americano(x) ∧ Arma(y) ∧ Vende(x, y, z) ∧ Ostile(z) ⇒ Criminale(x)
- Procedendo poi in avanti saranno utilizzate solo le "regole magiche" in modo mirato.
- Combina BC e FC