Intelligenza Artificiale II

Logica del Primo Ordine

Parte 2

Marco Piastra

Logica del Primo Ordine, Parte 2

3. Risoluzione, unificazione

4. Metodi a tableau

3

Risoluzione, unificazione

Decidibilità ed automazione di LPO

- Indecidibilità di LPO (Church)
  - non esiste un algoritmo *deterministico* (e di valore generale) in grado di stabilire se una fbf è un teorema
  - non si ha, come nel caso proposizionale, la possibilità di verificare direttamente tutte le possibili interpretazioni
- Qual è quindi la speranza di avere un calcolo automatico?
- In realtà, LPO 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 $\Gamma \models \varphi$
    - divergono (i.e. girano all'infinito) se $\Gamma \not\models \varphi$

Universo e base di Herbrand

- **Termini** e **atomi** di Herbrand
  - Dato un linguaggio $\mathscr{L}_{PO}$
  - Un **termine** di Herbrand è un termine 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* 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 (di $\mathscr{L}_{PO}$) è l'insieme di tutti i termini di Herbrand

    $\mathbf{U}_H \equiv$ {f(a), g(a,b), g(f(a),b), g(f(a),g(b,c)), g(f(a),g(f(b),c)) ... }
  - 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))), ... }

- (La base di Herbrand B_H serve definire interpretazioni di $\mathscr{L}_{PO}$ usando gli oggetti del linguaggio stesso ...)

Modelli di Herbrand

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

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

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

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

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

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

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 <u>tutte</u> le possibili combinazioni $< t_1, t_2 \ldots t_n >$, $t_i \in \mathbf{U}_H$

- **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 di Skolem

- Qualsiasi enunciato di LPO può essere trasformato in un enunciato *universale* equivalente
- Forma normale **prenessa** FNP (i.e. tutti i quantificatori all'inizio)
  - Un enunciato φ qualsiasi può essere trasformata in un enunciato equivalente

    $Q_1x_1\ Q_2x_2\ \ldots\ Q_nx_n\psi$ (ψ è anche detta ***matrice***)
  - dove Q_i è $\forall$ oppure $\exists$ e ψ non contiene quantificatori
  - si ottiene usando le definizioni $\neg\forall x\ \varphi \leftrightarrow \exists x\ \neg\varphi$ e $\neg\exists x\ \varphi \leftrightarrow \forall x\ \neg\varphi$

  – 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 (2)

- Forma normale di Skolem
  - Da una formula in forma prenessa $Q_1x_1\ Q_2x_2\ \ldots\ Q_nx_n\psi$ si eliminano i quantificatori esistenziali
  - Ogni variabile esistenzialmente quantificata viene sostituita da una (nuova) costante o una (nuova) funzione:
  - se Q_1 è $\exists$, allora x_1 viene sostituito con k_1 (nuova costante di Skolem)
  - se Q_i è il primo $\exists$ da sinistra, allora x_1 viene sostituito con $k_i(x_1, \ldots, x_{i-1})$ (nuova funzione di Skolem)
  - si itera il procedimento sulla sequenza $Q_1x_1\ Q_2x_2\ \ldots\ Q_nx_n$
  - Esempi:
    - $\exists y\ \forall x\ (P(y) \rightarrow P(x))$
      $\forall x\ (P(k(y)) \rightarrow P(x))$ (k funzione 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 $sko(\varphi)$ (forma normale di Skolem di φ) ha un modello

Semi-decidibilità effettiva di LPO

- 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 $H(sko(\Gamma \cup \{\neg\varphi\}))$ (sistema di Herbrand della forma normale di Skolem) che è **contraddittorio**
  - Quindi:
    - Una procedura che genera e controlla in sequenza *tutti* i sottoinsiemi finiti di $H(sko(\Gamma \cup \{\neg\varphi\}))$
    - prima o poi (in un tempo finito) ne trova uno contraddittorio (se $\Gamma \models \varphi$)
  - Esempio:
    - $\forall x\ (P(f(x)) \rightarrow P(g(x))),\ \neg P(g(a)) \models \exists y\ \neg P(g(y))$ (problema iniziale)
    - $\{\forall x\ (P(f(x)) \rightarrow P(g(x))),\ \neg P(g(a)),\ \neg\exists y\ \neg P(g(y))\}$ ($\Gamma \cup \{\neg\varphi\}$)
    - $\{\forall x\ (P(f(x)) \rightarrow P(g(x))),\ \neg P(g(a)),\ \forall y\ P(g(y))\}$ (definizione di $\exists$)
    - $\forall x\ \forall y\ (\neg P(f(x)) \vee P(g(x))) \wedge \neg P(g(a)) \wedge P(g(y))$ (FC 'all'indietro')
    - $(\neg P(f(b)) \vee P(g(b))) \wedge \neg P(g(a)) \wedge P(g(a))$ (appartiene a $H(sko(\Gamma \cup \{\neg\varphi\}))$, con [x/b, y/a], ed è contraddittoria)

Risoluzione

- Una procedura effettiva
  - per trovare un sottoinsieme finito e contraddittorio di H(*sko*($\Gamma \cup \{\neg\varphi\}$))
    - sperabilmente più efficiente dell'enumerazione ricorsiva
    - può non avere termine se $\Gamma \not\models \varphi$ (altrimenti LPO sarebbe decidibile)

- Risoluzione **proposizionale** (per refutazione)
  - 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 α o $\neg\alpha$

    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

    $\{\psi, a\}, \{\neg a, \chi\} \vdash \{\psi, \chi\}$ dove ψ e χ sono disgiunzioni qualsiasi
  - fino a derivare la clausola vuota (successo) o al termine (fallimento)

Forma a clausole in LPO

- Molto simile al caso proposizionale ...
  - Si parte da $sko(\Gamma \cup \{\neg\varphi\})$
    - cioè tutti gli enunciati di $sko(\Gamma \cup \{\neg\varphi\})$ 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 regola di 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)
    - $\forall x \exists y\ (P(g(x, f(a)), a) \vee Q(x, y))$
      $\forall w \exists z\ (\neg P(g(b, f(w)), w) \vee \neg R(z))$ (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 a cui applicare la risoluzione è
      $P(g(x, f(a)), a)$, $\neg P(g(b, f(w)), w)$
      ma i due atomi sono diversi

      Tuttavia, applicando la sostituzione σ = [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 letterali 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, non esiste sempre: P(g(x, f(a)), a), ¬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)

Risoluzione per refutazione in LPO

- Problema: $\Gamma \models \varphi$?
  - Procedura completa:
    1) Insieme di enunciati $\Gamma \cup \{\neg\varphi\}$
    2) Forma normale prenessa + skolemizzazione *sko*$(\Gamma \cup \{\neg\varphi\})$
    3) Traduzione di *sko*$(\Gamma \cup \{\neg\varphi\})$ in forma a clausole (FC)
    4) Standardizzazione delle variabili
       (ogni quantificatore opera su una variabile diversa)
    5) Applicazione iterativa della regola di risoluzione:
    6) Selezione di due clausole da risolvere
    7) Costruzione del MGU μ (deve essere il MGU, cioè la sostituzione minima)
    8) Applicazione di μ e generazione del risolvente
    9) Torna a 6)
  - Terminazione della procedura:
    - Quando si produce la clausola vuota {} (successo)
      vale a dire si trovano due clausole della forma $\{\alpha\}, \{\neg\alpha\} \vdash \{\}$
    - In caso contrario, la procedura potrebbe continuare all'infinito

Esempio 8

- Problema: $\Gamma \models \varphi$?
  - $\Gamma \equiv$ {∀x (Filosofo(x) → Umano(x)), ∀x (Umano(x) → Mortale(x)), Filosofo(Socrate)}
  - $\varphi \equiv$ Mortale(Socrate)
  - Procedura (risoluzione per refutazione):
    1: {∀x (Filosofo(x) → Umano(x)), ∀x (Umano(x) → Mortale(x)), Filosofo(Socrate), ¬Mortale(Socrate)}

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

    2: {{¬Filosofo(x), Umano(x)}, {¬Umano(y), Mortale(y)}, {Filosofo(Socrate)}, {¬Mortale(Socrate)}}

    (standardizzazione e forma a clausole)

    3: Applicazione iterativa della regola di risoluzione:

    4: {Filosofo(Socrate)}, {¬Filosofo(x), Umano(x)}, [x/Socrate] ⊢ {Umano(Socrate)}

    5: {Umano(Socrate)}, {¬Umano(y), Mortale(y)}, [y/Socrate] ⊢ {Mortale(Socrate)}

    6: {Mortale(Socrate)}, {¬Mortale(Socrate)}, [] ⊢ {}
    - Successo

Clausole di Horn in LPO

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

      {Umano(Socrate)}, {Piramide(a)}, {Sorella(Alba, madreDi(Paolo))}
    - **Regole**: clausola di due o più letterali, uno in forma positiva

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

      {¬Donna(x), ¬Genitore(k(x), x), ¬Genitore(k(y), y), Sorella(x, y)}
      $\forall x\ \forall y\ ((\text{Donna}(x) \wedge \exists z\ (\text{Genitore}(z, x) \wedge \text{Genitore}(z, y))) \rightarrow \text{Sorella}(x, y))$

      {¬Sovrasta(x, y), Sopra(x, k(x))}, {¬Sovrasta(x, y), Sopra(j(y), y)}
      $\forall x\ \forall y\ (\text{Sovrasta}(x, y) \rightarrow (\exists z\ \text{Sopra}(x, z) \wedge \exists v\ \text{Sopra}(v, y)))$
    - **Goal**: clausola di letterali in forma negativa

      {¬Umano(Socrate)}

      {¬Sorella(Alba, Amelia), ¬Sorella(Amelia, Alba)}

Clausole di Horn e modelli di Herbrand

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

- **Modello minimo** di Herbrand
  - Il modello minimo M_P è l'intersezione di tutti i modelli di Herbrand M_i di P:
    $M_P \equiv \cap_{\forall i} M_i$
- Teorema (van Emden e Kovalski, 1976)
  - Siano P ed A insiemi di clausole di Horn, le seguenti affermazioni sono equivalenti:
    - $P \models A$
    - $A \in M_P$
    - L'unione di tutte le clausole A che sono conseguenza logica di P coincide con M_P

Programmazione logica

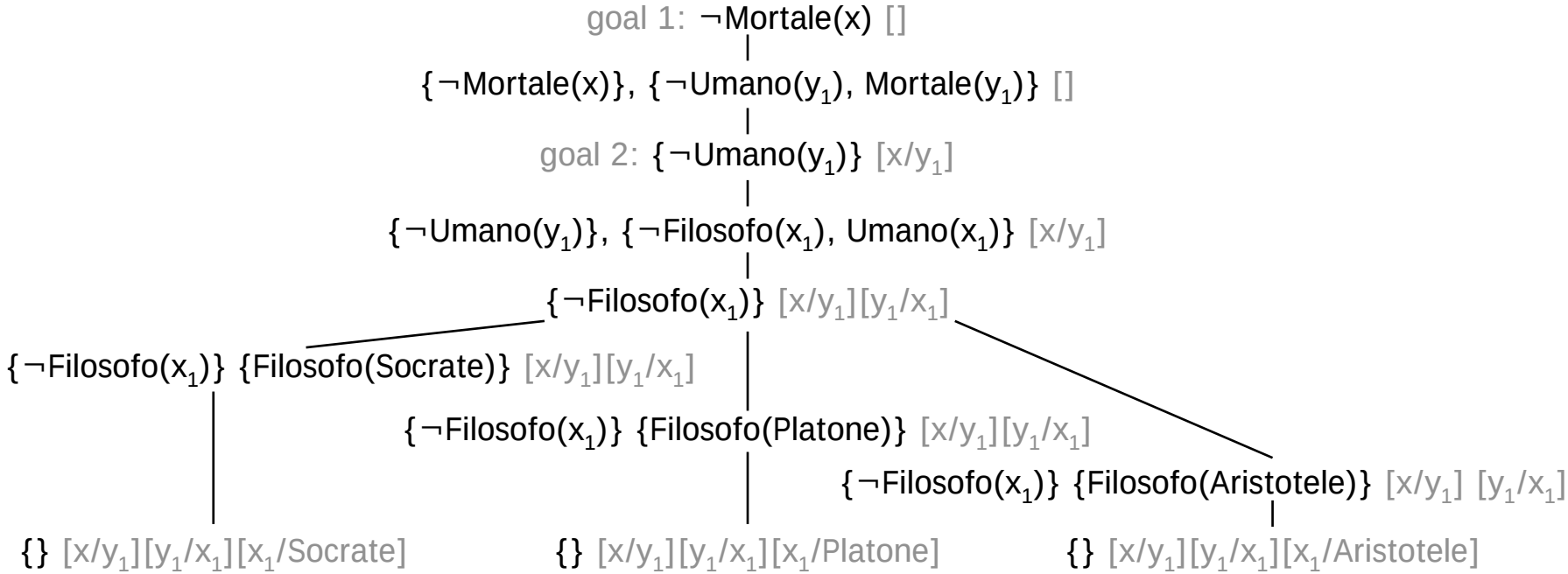
- (Un dimostratore di teoremi risponde alle domande "$\Gamma \models \varphi$?")
  - Si rammenti che se $\Gamma \models \varphi$ allora $\Gamma \cup \{\neg\varphi\}$ è insoddisfacibile
- Teorema (Apt e van Emden, 1982)
  - Sia P un **programma** in clausole di Horn.
    Applicata a P, la procedura di risoluzione genera esattamente M_P
    - Si usa P al posto di $\Gamma \cup \{\neg\varphi\}$
    - In generale, P è soddisfacibile
    - La procedura termina se M_P è finito
  - Esempio:
    - P ≡ {{¬Filosofo(x), Umano(x)}, {¬Umano(y), Mortale(y)}, {Filosofo(Socrate)}, {Filosofo(Platone)}, {Filosofo(Aristotele)}}
    - Applicando la procedura di risoluzione in modo esaustivo, si ottiene:
    - M_P ≡ {{Filosofo(Socrate)}, {Filosofo(Platone)}, {Filosofo(Aristotele)}, {Umano(Socrate)}, {Umano(Platone)}, {Umano(Aristotele)}, {Mortale(Socrate)}, {Mortale(Platone)}, {Mortale(Aristotele)}}
    - (assomiglia ad una query su un database)

Risoluzione SLD

- Una strategia effettiva per la risoluzione di programmi
- SLD
  - S: *selection rule*, selezione dei **goal** da sinistra a destra
    L: *linear*, si procede in sequenza (secondo l'ordinamento testuale)
    D: *definite clause*, un altro nome per le clausole di Horn
- Descrizione
  - Goal: $\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_k$ (uno solo, per semplicità)
  - Regole (o fatti): $\alpha_i \vee \neg\beta_1 \vee \neg\beta_2 \vee \ldots \vee \neg\beta_n$ (per i fatti, $n = 0$)
  - Procedura:
    - I goal vengono considerati in ordine, da 1 a k
    - Per ciascun goal $\neg\alpha_i$ viene tentata la risoluzione (con unificazione) di tutte le regole (o fatti) che hanno α_i come letterale positivo
    - Prima di tentare l'unificazione, vengono sempre ridenominate le variabili della regola (o fatto) $\alpha_i \vee \neg\beta_1 \wedge \neg\beta_2 \wedge \ldots \wedge \neg\beta_n$
    - Le risposte sono le assegnazioni che permettono di derivare la clausola vuota
    - L'insieme delle risposte (in teoria) è M_P

Alberi SLD

- Una rappresentazione della *traccia* di esecuzione della risoluzione SLD
  - Esempio:
    - P ≡ {{¬Filosofo(x), Umano(x)}, {¬Umano(y), Mortale(y)}, {Filosofo(Socrate)}, {Filosofo(Platone)}, {Filosofo(Aristotele)}}
    - goal ≡ {¬Mortale(x) ∨ ¬Umano(x)} "Chi è mortale ed umano?"



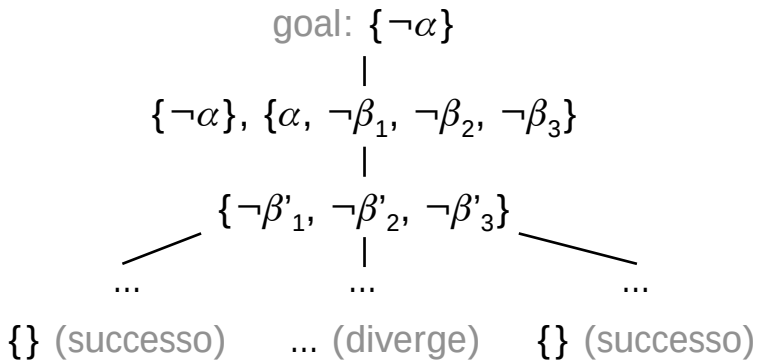
SLD e programmazione logica

- Insieme delle risposte
  - Ogni ramo dell'albero SLD che termina con una clausola vuota (conseguenza)
  - Insieme $M_P \equiv$ i goal (in forma positiva) a cui sono applicati tutti gli MGU + i fatti del programma

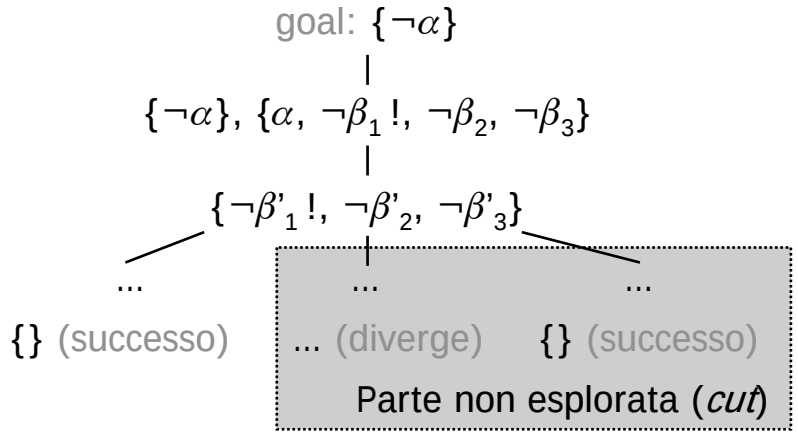
    $M_P \equiv$ Mortale(Socrate) $\wedge$ Umano(Socrate) $\wedge$ Filosofo(Socrate) $\wedge$
    Mortale(Platone) $\wedge$ Umano(Platone) $\wedge$ Filosofo(Platone) $\wedge$
    Mortale(Aristotele) $\wedge$ Umano(Aristotele) $\wedge$ Filosofo(Aristotele)
- Esecuzione
  - L'ordine testuale del programma (in teoria) è irrilevante (però ...)
  - L'albero SLD può essere generato in due modi
    - in ampiezza (*depth first*)
    - in profondità (*breadth first*)
    - solo il modo *in ampiezza* è teoricamente corretto
    - in pratica si utilizza il modo in profondità

Prolog

- Un linguaggio effettivo di programmazione logica
  - Caratteristiche specifiche
    - Risoluzione SLD *depth first*
    - Più efficiente della *breadth first*
    - Ma incompleta: un ramo divergente impedisce di trovare tutte le risposte dei rami 'a destra'
    - L'insieme di risposta dipende dall'ordine testuale



    - Uso del taglio (*cut*)
    - Interrompe l'esplorazione dell'albero al primo successo
    - Non ha una semantica logica
    - E` un 'controllo di flusso'



3

Metodi a tableau

Metodo a tableau

- E` un metodo basato sulla refutazione:
  - per stabilire che $\Gamma \models \varphi$ si tenta di mostrare che $\Gamma \cup \{\neg\varphi\}$ è inconsistente

- Caratteristiche generali:
  - l'insieme $\Gamma \cup \{\neg\varphi\}$ viene completamente espanso in forma di *albero*
  - si usano diverse regole di inferenza per l'espansione
  - ogni nodo dell'albero rappresenta una **congiunzione** di fbf
  - ogni biforcazione rappresenta una **disgiunzione**
  - un ramo è chiuso non appena si incontra una contraddizione $\{\varphi, \neg\varphi\}$
  - $\Gamma \cup \{\neg\varphi\}$ è inconsistente (o insoddisfacibile) se tutti i rami dell'albero sono **chiusi**
  - $\Gamma \cup \{\neg\varphi\}$ non è inconsistente se un ramo non si chiude
  - non è necessario che $\Gamma \cup \{\neg\varphi\}$ sia espresso in una forma normale

Regole di inferenza per i tableaux

- Regole α (congiunzione = espansione)

(a1) $\varphi \wedge \psi$ | φ, ψ

(a2) $\neg(\varphi \vee \psi)$ | $\neg\varphi, \neg\psi$

(a3) $\neg(\neg\varphi)$ | φ

$\neg(\varphi \rightarrow \psi)$ | $\varphi, \neg\psi$

- Regole β (disgiunzione = biforcazione)

(b1) $\varphi \vee \psi$ → φ | ψ

(b2) $\neg(\varphi \wedge \psi)$ → $\neg\varphi$ | $\neg\psi$

$\varphi \rightarrow \psi$ → $\neg\varphi$ | ψ

$\varphi \leftrightarrow \psi$ → $\neg\varphi, \neg\psi$ | φ, ψ

$\neg(\varphi \leftrightarrow \psi)$ → $\neg\varphi, \psi$ | $\varphi, \neg\psi$

Algoritmo del metodo a tableau

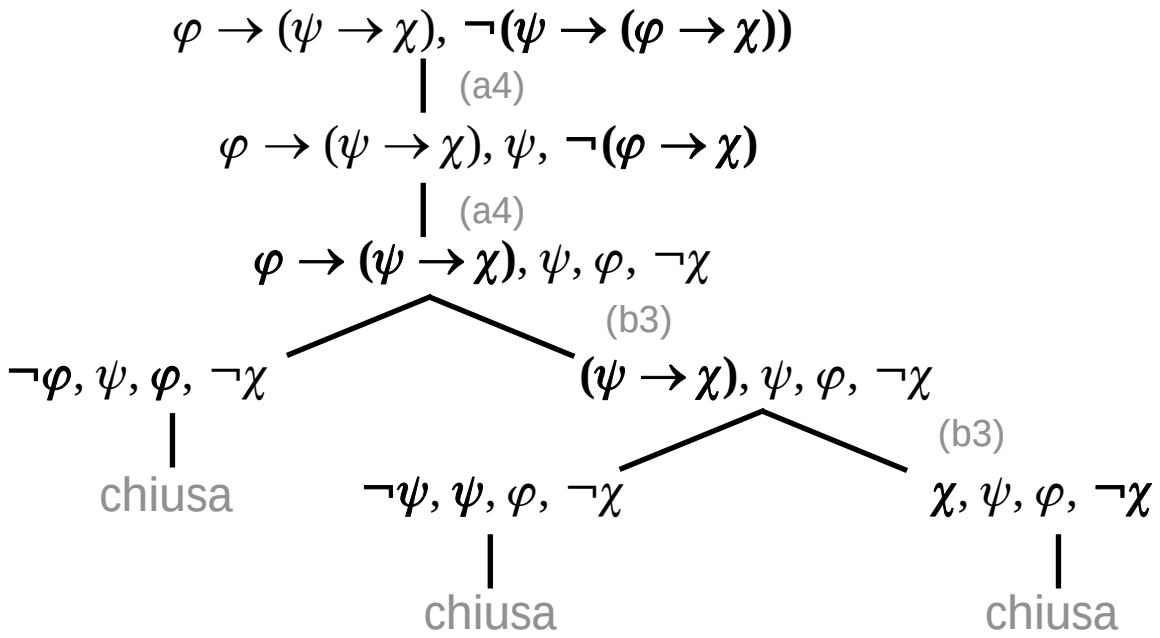
- Procedura per $\Gamma \models \varphi$

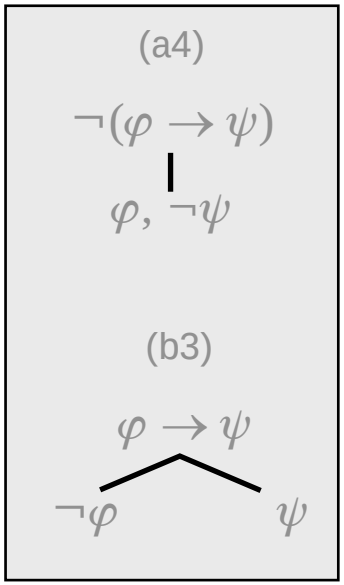
  1) Definire il nodo iniziale $\Gamma \cup \{\neg\varphi\}$
  2) Per ogni ramo, in modalità *depth first*
  3) Se il nodo contiene solo letterali,
     allora se il nodo contiene una contraddizione, chiudere
     altrimenti il procedimento è fallito, uscire
  4) Se il nodo contiene formule
     applicare le regole α
     applicare le regole β

- Notare che le regole α vengono applicate prima delle regole β

Esempio 9

- Obiettivo
  - stabilire che $\varphi \to (\psi \to \chi) \vdash \psi \to (\varphi \to \chi)$
- Espansione





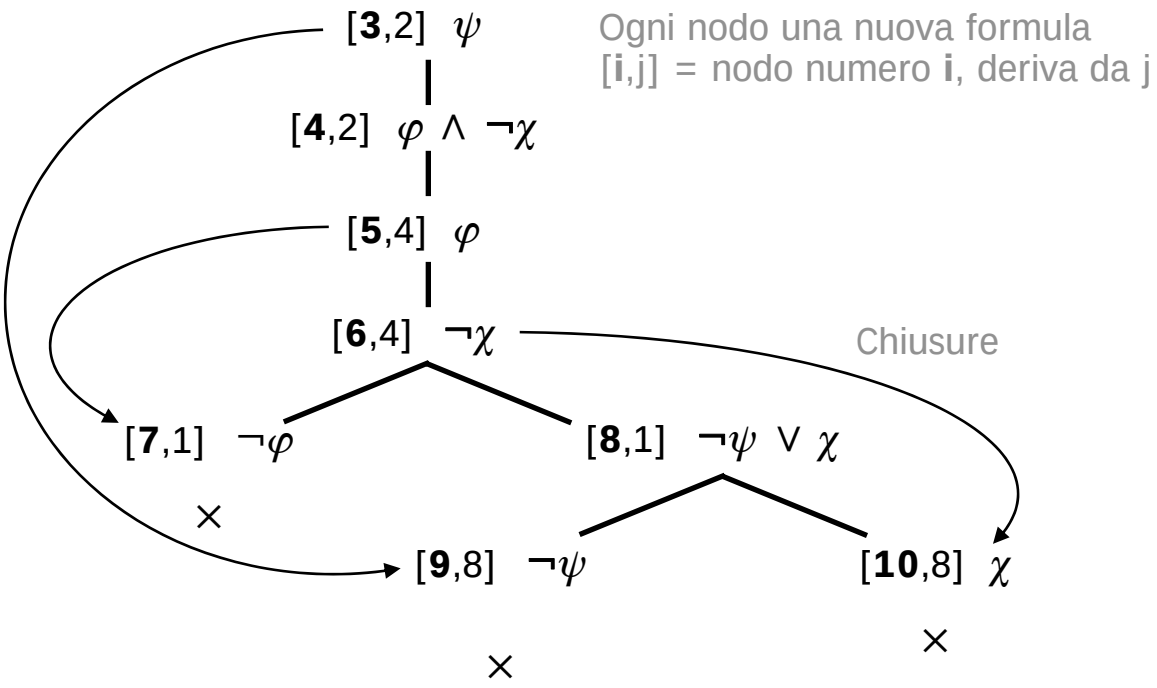
Esempio 10

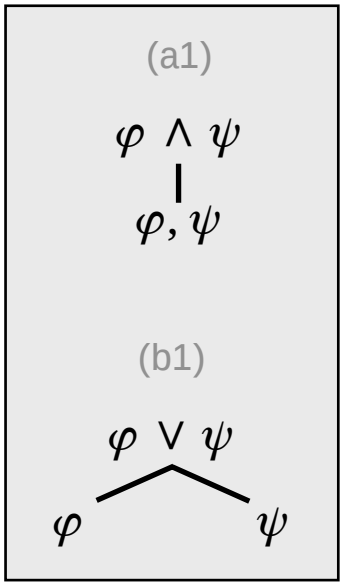
- Formule di partenza (non sono in forma normale)

  [**1**] $\neg\varphi \vee (\neg\psi \vee \chi)$

  [**2**] $\psi \wedge (\varphi \wedge \neg\chi)$

- Espansione





Estensione delle regole

- Tableaux per la logica del primo ordine
  - Le regole α e β rimangono valide per gli enunciati privi di quantificatori
  - Si aggiungono le regole γ e δ per le formule con quantificatori
  - Si assume la forma normale **prenessa (FNP)**

- Schema generale delle regole α, β, γ e δ

α	$\alpha_1, \ldots, \alpha_n$
$\phi_1 \wedge \ldots \wedge \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \vee \ldots \vee \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$
$\neg\neg\phi$	ϕ

β	$\beta_1, \ldots, \beta_n$
$\phi_1 \vee \ldots \vee \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \wedge \ldots \wedge \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$

γ	γ_1
$(\forall x)(\phi(x))$	$\phi(x)$
$\neg(\exists x)(\phi(x))$	$\neg\phi(x)$

δ	δ_1
$\neg(\forall x)(\phi(x))$	$\neg\phi(x)$
$(\exists x)(\phi(x))$	$\phi(x)$

Regole e unificazione

- Applicazione delle regole per le variabili

$$\frac{\alpha}{\begin{array}{l}\alpha_1\\ \vdots\\ \alpha_n\end{array}} \qquad \frac{\beta}{\beta_1 \mid \cdots \mid \beta_n} \qquad \frac{\gamma(x)}{\gamma_1(y)}$$

$y \in \mathrm{Var}$ is new to the tableau.

$$\frac{\delta(x)}{\delta_1(sko_\delta(x_1,\ldots,x_n))}$$

$x_1,\ldots,x_n$ are the free variables in δ.

- La chiusura
  - Si effettua su coppie di fbf atomiche complementari
  - Trovando l'unificatore più generale **mgu**
    che va applicato a tutto il tableau

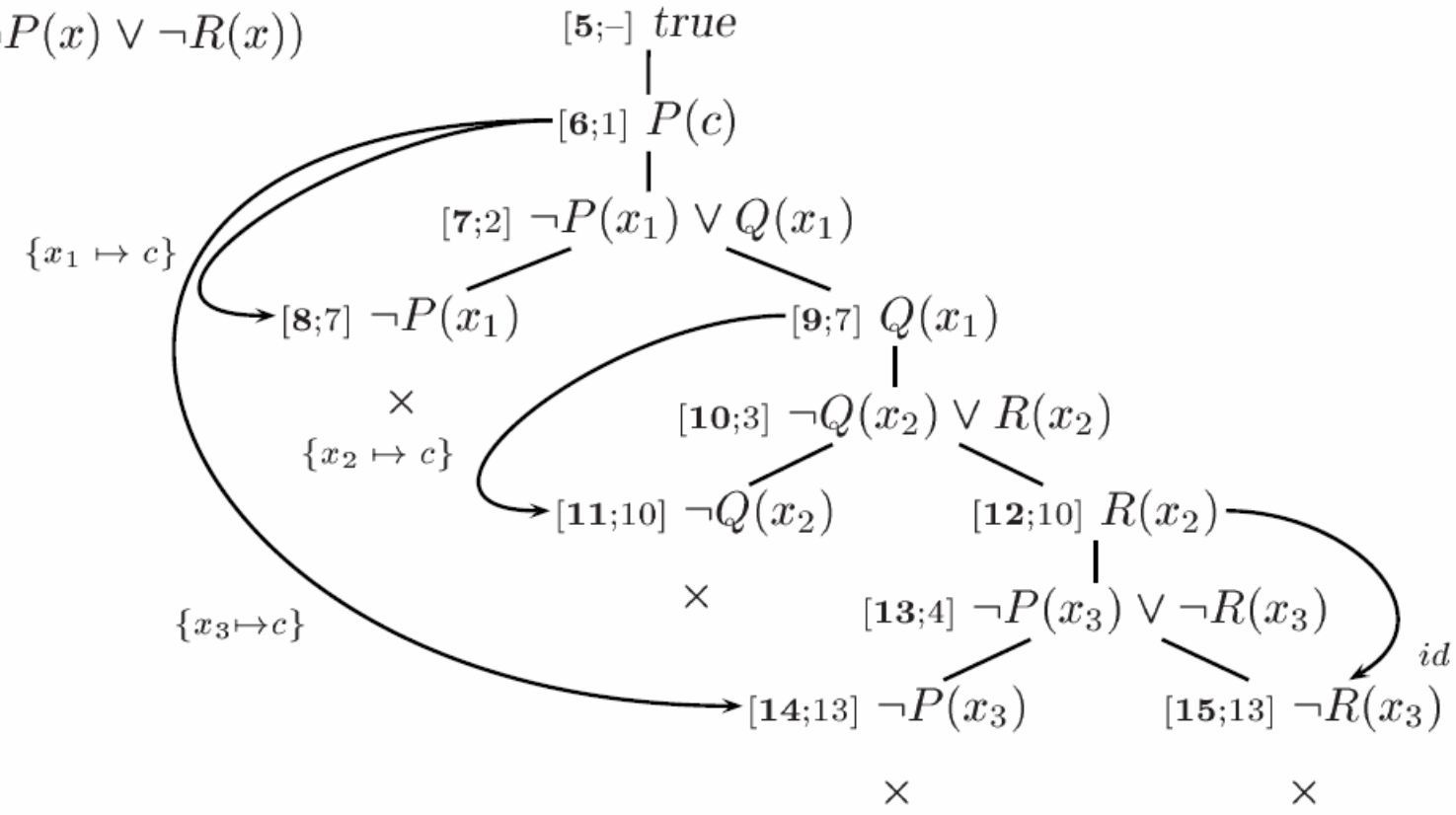
Esempio 11

(1) $(\exists x)P(x)$

(2) $(\forall x)(\neg P(x) \vee Q(x))$

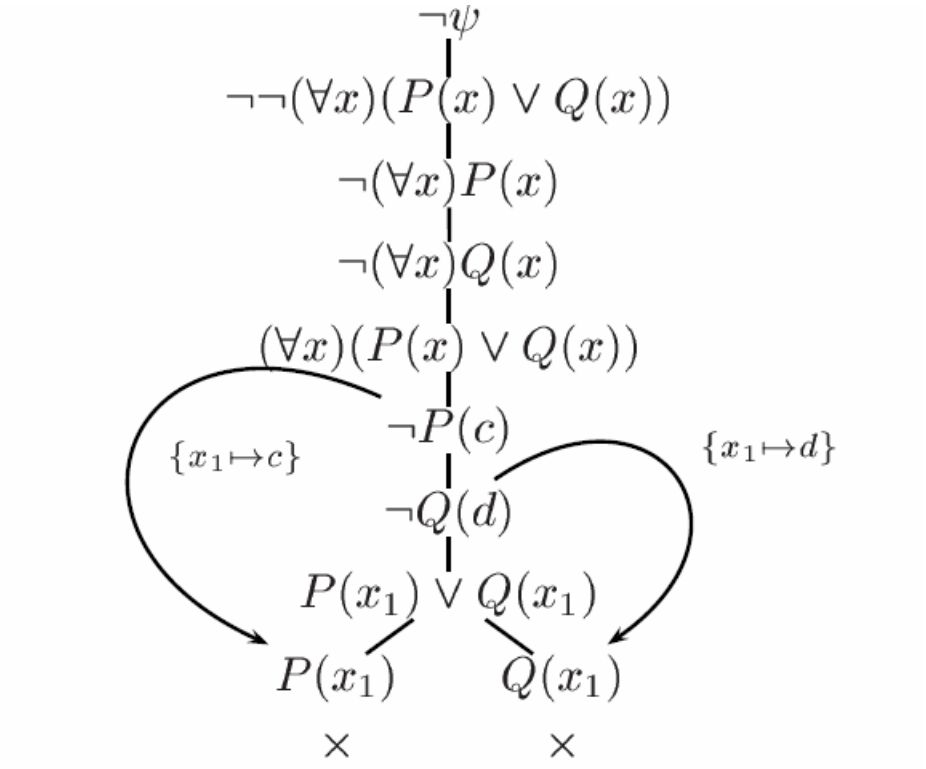
(3) $(\forall x)(\neg Q(x) \vee R(x))$

(4) $(\forall x)(\neg P(x) \vee \neg R(x))$



Esempio 12

- Non è corretto: le unificazioni vanno applicate a tutto il tableau



Algoritmo per i tableaux in LPO

- Problema della completezza

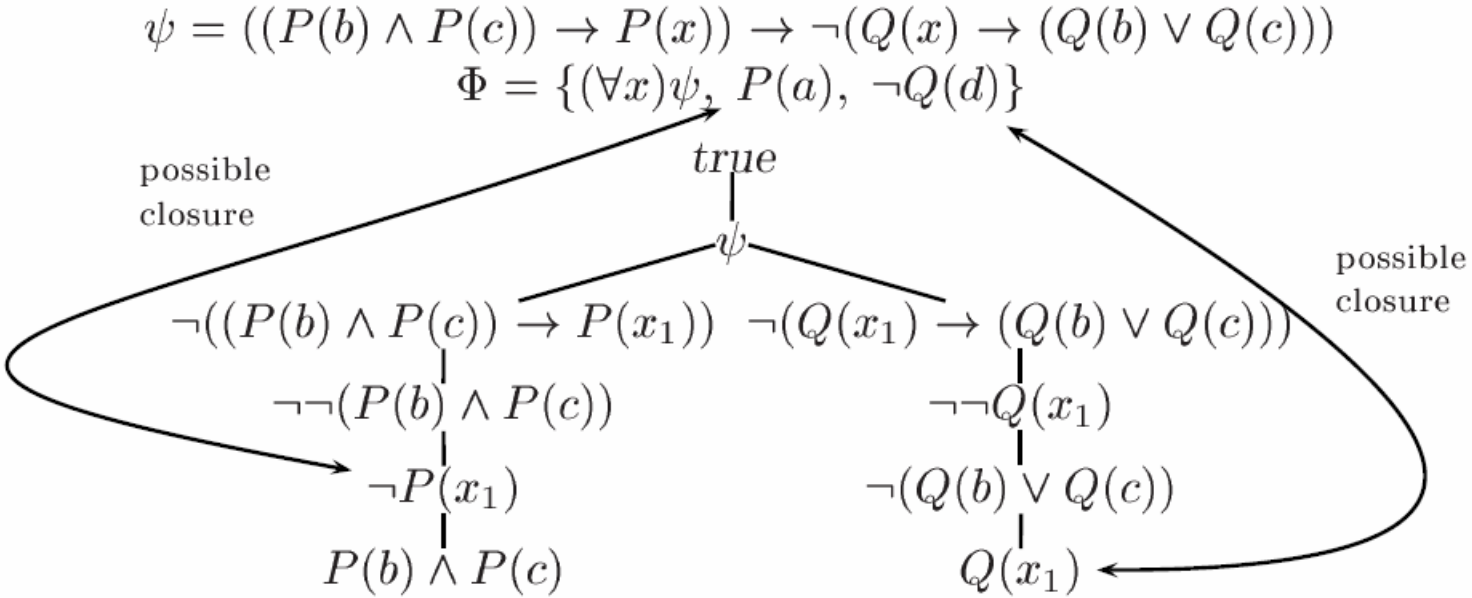


Figure 5: Incompleteness caused by unfair *select mode*.

Strategia

An obvious way to tableaux with unification into a strongly complete proof procedure is to separate application of expansion and closure rules. Under a fair computation rule, delay the application of the closure rule until all tableau branches can be closed simultaneously by a suitable substitution. This is the path chosen in Fitting's [1996] text book—and it has its inefficiencies: first, one cannot discard closed branches until the proof is essentially finished which might lead to storage problems (this can be partially remedied, see Section 3.5), and second, after each expansion rule application, the whole tableau must be tested for closure, which is very redundant. If more sophisticated data structures were used and the different MGUs available to close each branch were maintained as a tableau-wide constraint system that can be incrementally tested, then this approach might still be worth a try. There is experimental evidence to support this [Giese 2000].