Logica

II parte

La base della risoluzione

La risoluzione è un processo iterativo che, ad ogni passo, confronta (*risolve*) due clausole *genitori* e permette di inferire una *nuova clausola*.

Esempio. Nel sistema ci siano due clausole:

$\text{inverno} \vee \text{estate}$ cioè $\neg\text{inverno} \rightarrow \text{estate}$

$\neg\text{inverno} \vee \text{freddo}$ cioè $\text{inverno} \rightarrow \text{freddo}$

Entrambe le clausole devono essere vere (sono le nostre osservazioni iniziali).

Ora solo una tra inverno e ¬inverno può essere vera in ogni punto.

Se inverno è vera, allora freddo deve essere vero (per avere la II clausola vera).

Se inverno è falso, allora estate deve essere vera (per la I clausola).

Dalle due clausole si può dedurre dunque:
estate $\vee$ freddo cioè $\neg$estate $\rightarrow$ freddo.

Conclusione: per la risoluzione si considerano due clausole che contengono la stessa formula atomica, una volta affermata e una volta negata.

La risolvente è ottenuta combinando tutte le formule atomiche delle clausole genitori eccetto quelle che cancella.

Se si produce la clausola vuota, si ha una contraddizione. Esempio:

inverno

¬inverno (risulta la clausola vuota)

Il tentativo sarà quello di trovare una contraddizione, se esiste.

Se non esiste, la procedura può non avere mai termine.

Risoluzione nella logica proposizionale

Procedimento:

1)Trasformare tutte le proposizioni in F in forma a clausole.
Negare S e trasformare il risultato in forma a clausole. Aggiungerlo all'insieme di clausole ottenute al passo 1).
2)Ripetere fino a quando viene trovata una contraddizione o non si può più andare avanti i passi seguenti:

a) Selezionare due clausole. Chiamarle clausole genitori.
b) Risolverle insieme. La clausola risultante, chiamata la *risolvente*, sarà la disgiunzione di tutte le formule atomiche di entrambe le clausole genitori con la seguente eccezione: se vi sono coppie di formule atomiche L e ¬L, tali che una delle clausole genitori contenga L e l'altra contenga ¬L, allora eliminare sia L che ¬L dalla risolvente.
c) Se la risolvente è la clausola vuota, allora è stata trovata una contraddizione. In caso contrario, aggiungerla all'insieme di clausole a disposizione della procedura.

Esempio: siano dati i seguenti assiomi:

Assiomi:

p

$(p \wedge q) \rightarrow r$

$(s \vee t) \rightarrow q$

t

Convertiti in forma a clausole:

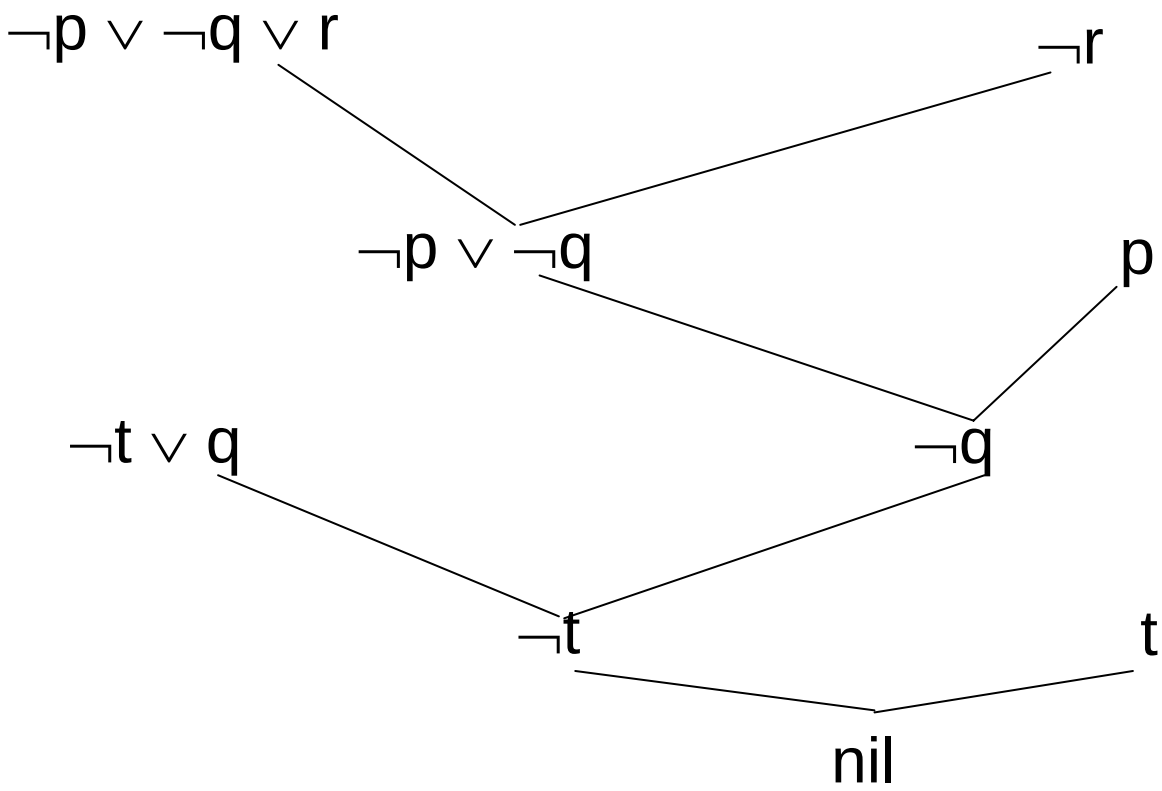
p	1.
$\neg p \vee \neg q \vee r$	2.
$\neg s \vee q$	3.
$\neg t \vee q$	4.
t	5.

Si voglia dimostrare r.

Dopo aver trasformato gli assiomi in forma a clausola, si introduce nella lista ¬r (già in forma a clausola).

Poi si selezionano le clausole a 2 a 2 e si risolvono (conviene scegliere clausole che contengono la stessa forma atomica una volta affermata e una volta negata).

Si ottiene, ad esempio:



Nota:
la proposizione 2 è vera se sono vere

¬p, ¬q, r

Al primo passo si assume che ¬r sia vera, insieme alla preposizione 2. Ciò può accadere solo se è vera ¬p oppure ¬q.

È quello che afferma la risolvente!

Conclusione:

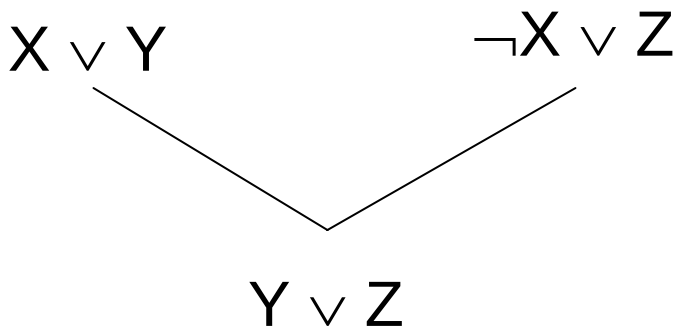
per provare r, si prova che ¬r crea contraddizione. Si inserisce quindi ¬r nella lista di forme a clausola e si cerca, mediante risoluzione, se esiste questa contraddizione (*metodo della refutazione*).

Fondamenti giustificativi delle risoluzione:

Il teorema m) duale:

$$(X + Y)(\overline{X} + Z)(Y + Z) = (X + Y)(\overline{X} + Z)$$

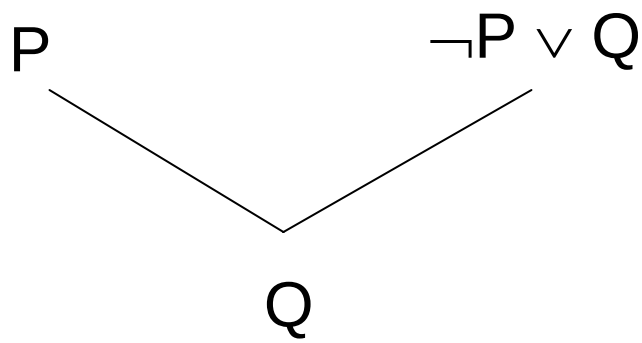
letto da destra a sinistra, porta a giustificare la risoluzione:



È una generalizzazione del *modus ponens*, in cui:

$$\{P, P \rightarrow Q\} \vdash Q$$

Infatti, in clausole:



L’universo di Herbrand

Prima si è vista la risoluzione nella logica proporzionale.

Nel caso della logica dei predicati, ci si pone il problema di dimostrare che un insieme finito S di clausole è insoddisfacibile.

Cioè bisogna dimostrare che non esiste *nessuna* interpretazione che lo soddisfi.

Per specificare un'interpretazione per S, si deve scegliere un dominio D, per poi associare a ogni costante in S un elemento di D ecc: ci sono *infiniti domini* e *infinite associazioni possibili*.

Herbrand ha dimostrato che è possibile enumerare un'adeguata lista di nomi per gli elementi del domino, tale che se mostriamo che non esiste un'interpretazione soddisfacente su domini i cui elementi possono essere nominati dai nomi di quella lista, ciò equivale a mostrare che non esiste nessuna interpretazione soddisfacente.

Si dice *universo di Herbrand* la lista di nomi adeguati a un insieme S di clausole.

Si definisce ricorsivamente così:

1. L'insieme di tutte le lettere costanti $\{f_i^0\}$ nominate in S è contenuto in H(S). Se $\{f_i^0\}$ è vuoto, ammettiamo che H(S) contenga una lettera costante arbitraria, diciamo *a*.
2. Supponiamo che i termini $t_1, \ldots, t_n$ si trovino in H(S). Allora anche le espressioni $f_i^n(t_1, t_2, \ldots, t_n)$ si trovano in H(s), dove le f_i^n sono le lettere funzionali nominate in H(S).
3. Nessun altro termine si trova in S.

In pratica, H(S) è il dominio più generale: se si dimostra che S è insoddisfacibile sul dominio H(S), si è sicuri che è insoddisfa-cibile su tutti i domini.

L'universo di Herbrand è infinito ma numerabile: i suoi elementi possono essere quindi ordinati secondo qualche criterio.

Esempio. Sia dato l'insieme di clausole:

$\{P(x) \vee Q(a) \vee \neg P(f(x)),\ \neg Q(b) \vee P(g(x,y))\}$

I termini costanti sono {a, b} e le funzioni {f, g}.
H(S) è quindi l'insieme infinito (numerabile) di espressioni {a, b, f(a), f(b), g(a,a), g(a,b), g(b,a), g(b,b), f(f(a)), f(f(b)), g(a, f(a)), …}.

Concludendo:

- Per vedere se un insieme di clausole S è insoddisfacibile, è necessario considerare le interpretazioni solo su un insieme particolare, chiamato *universo di Herbrand* di S.
- Un insieme di clausole S è insoddisfacibile se e solo se un sottoinsieme finito di esemplari di base (in cui tutte le variabili legate sono state sostituite con un valore) di S è insoddisfacibile.

La seconda limita la ricerca

La prima suggerisce di cercare sistematicamente le sostituzioni possibili per vedere se qualcuna produce contraddizione.

Applicato brutalmente, è inefficiente.

Robinson (1965) dà un algoritmo di risolu-zione più efficiente, basato sull'idea di mantenere le clausole nella loro forma più generale il più a lungo possibile e di introdurre le sostituzioni solo quando richieste.

Base di Herbrand ed alberi semantici

Si definisce *base di Herbrand* di S l'insieme degli esempi base di tutte le formule atomiche di S, ottenuti usando l'*universo* di Herbrand per nominare gli elementi del dominio (gli elementi della base di Herbrand sono detti *atomi*).

Un'interpretazione su H(S) è completa per tutte le clausole di S quando ad ogni atomo della base si assegna un valore di verità.

Si osservi che anche la base di Herband è un insieme numerabile, i cui elementi possono essere ordinati in qualche modo (per esempio, possono essere disposti in una successione ordinata $\{p_1, p_2, p_3, ...\}$).

Si può costruire pertanto un albero semantico, come illustrato nel seguente esempio.

L'insieme S sia costituito dall'insieme insoddisfacibile di clausole

$P(x) \vee Q(y)$

$\sim P(a)$

$\sim Q(b)$

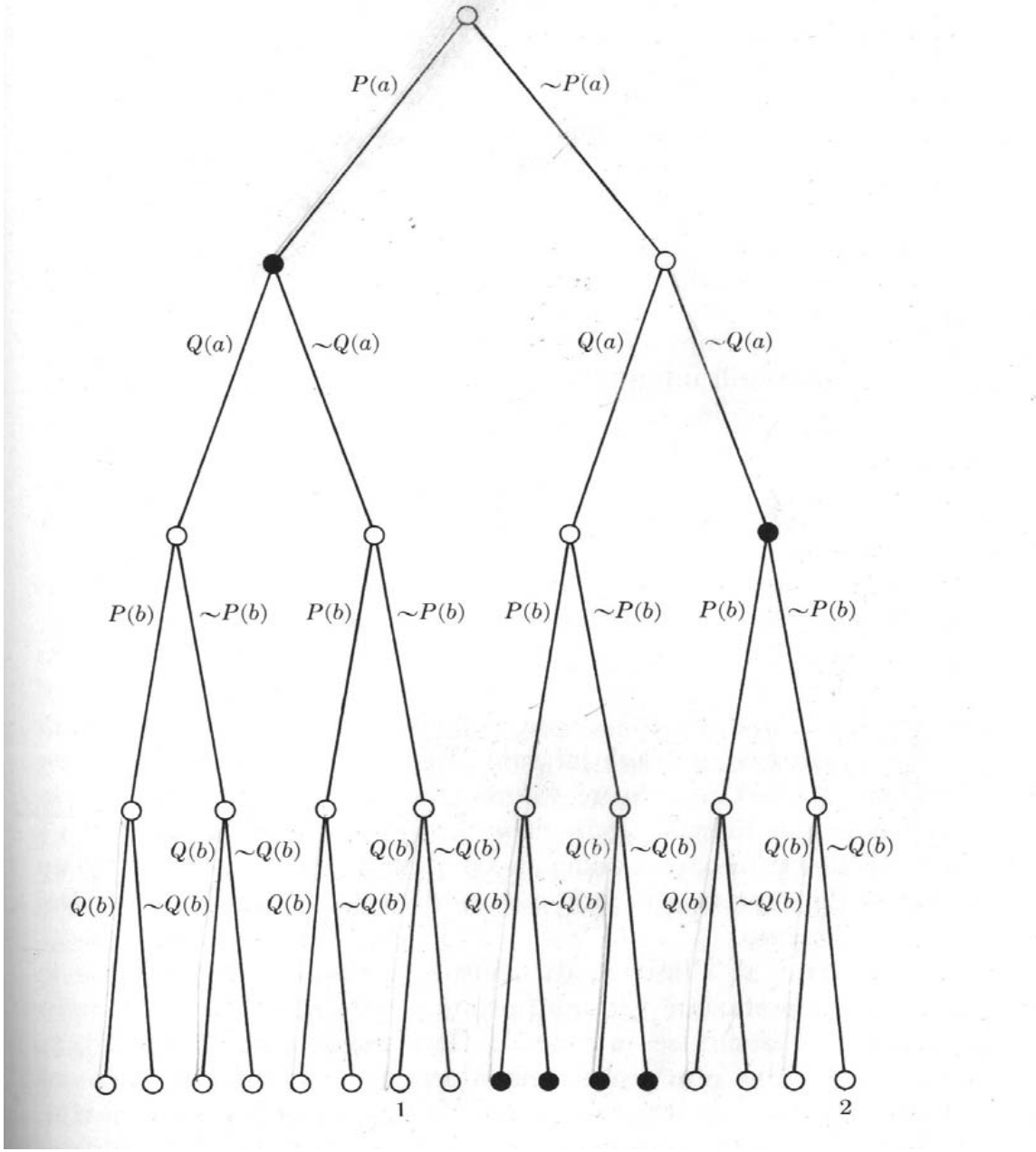
In questo caso l'insieme di Herbrand è l'insieme $H(S) = \{a, b\}$ (non ci sono funzioni!)

La base di Herbrand è finita e può essere ordinata così $\{P(a), Q(a), P(b), Q(b)\}$

L’albero semantico è un albero binario che parte da un nodo radice ed ha i rami etichettati con ciascun atomo, assunto una volta con il valore Vero e una volta con il valore Falso.

Nell'esempio, l'albero semantico di S è finito ed ha la seguente forma:

Fig. 6.1 – L'albero semantico dell'insieme di clausole $\{P(x) \vee Q(y), \sim P(a), \sim Q(b)\}$.



Se si percorre un cammino dalla radice ad una foglia si ottiene un'interpretazione per l'insieme S.

Ad esempio, per il cammino che porta alla foglia 1 si ha

$$M_1 = \{P(a), \sim Q(a), \sim P(b), Q(b)\}$$

(insieme che è detto *modello* dell'insieme di clausole)

Ora si può vedere che non soddisfa né la clausola $\sim P(a)$ né la clausola $\sim Q(b)$, quindi fornisce un valore *F*.

Similmente

$$M_2 = \{\sim P(a), \sim Q(a), \sim P(b), \sim Q(b)\}$$

non soddisfa la clausola : infatti l'esempio base avrebbe valore *F*.

Tutte e 16 le interpretazioni forniscono valore *F* (l'insieme è insoddisfacibile).

In generale, un albero semantico ha cammini infiniti.

Un risultato interessante è che non è indispensabile percorrere cammini infiniti per determinare che certe interpretazioni non soddisfano l'insieme di clausole.

Nell'esempio precedente, il ramo di sinistra in cui P(a) vale *V* poteva non essere sviluppato perché P(a) posto a quel valore non può soddisfare l'insieme di clausole. Si dice che P(a) è un *nodo di fallimento*.

Man mano che si costruisce l'albero semantico, si effettuano delle verifiche e ci si ferma appena si incontra un nodo di fallimento.

Se tutti i cammini sono interrotti, l'albero si dice *chiuso*.

Pertanto l'insoddisfacibilità di un insieme di clausole è riportato alla verifica che l'albero semantico è chiuso:

un albero semantico di un insieme insoddisfacibile S di clausole è chiuso per S e contiene un numero finito di nodi al di sopra dei nodi di fallimento (corrisponde al teorema di Herbrand).

Altro esempio, in cui la base di Herbrand è infinito:

Insieme S di clausole insoddisfacibile:

$\sim P(x) \vee Q(x)$

$P(f(y))$

$\sim Q(f(y))$

Universo di Herbrand:

$$H(S) = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$$

(non essendoci costanti nelle clausole, si suppone che esista almeno una, *a*)

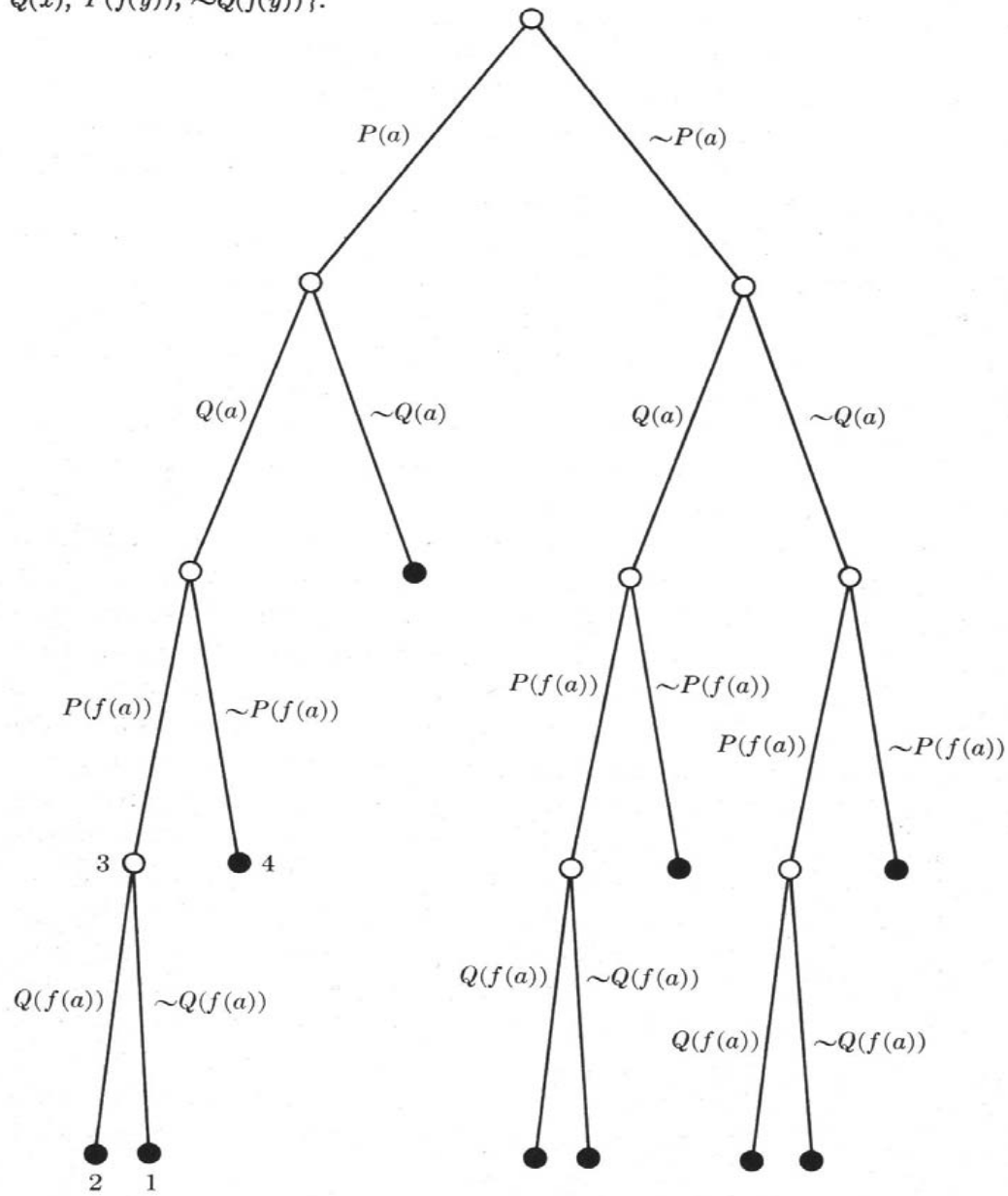
La base di Herbrand può essere così ordinata:

$$\{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), \ldots\}$$

La figura seguente mostra l'albero semantico sviluppato fino ai nodi di fallimento.

Provare l'insoddisfacibilità mediante lo sviluppo dell'albero semantico non è pratico.

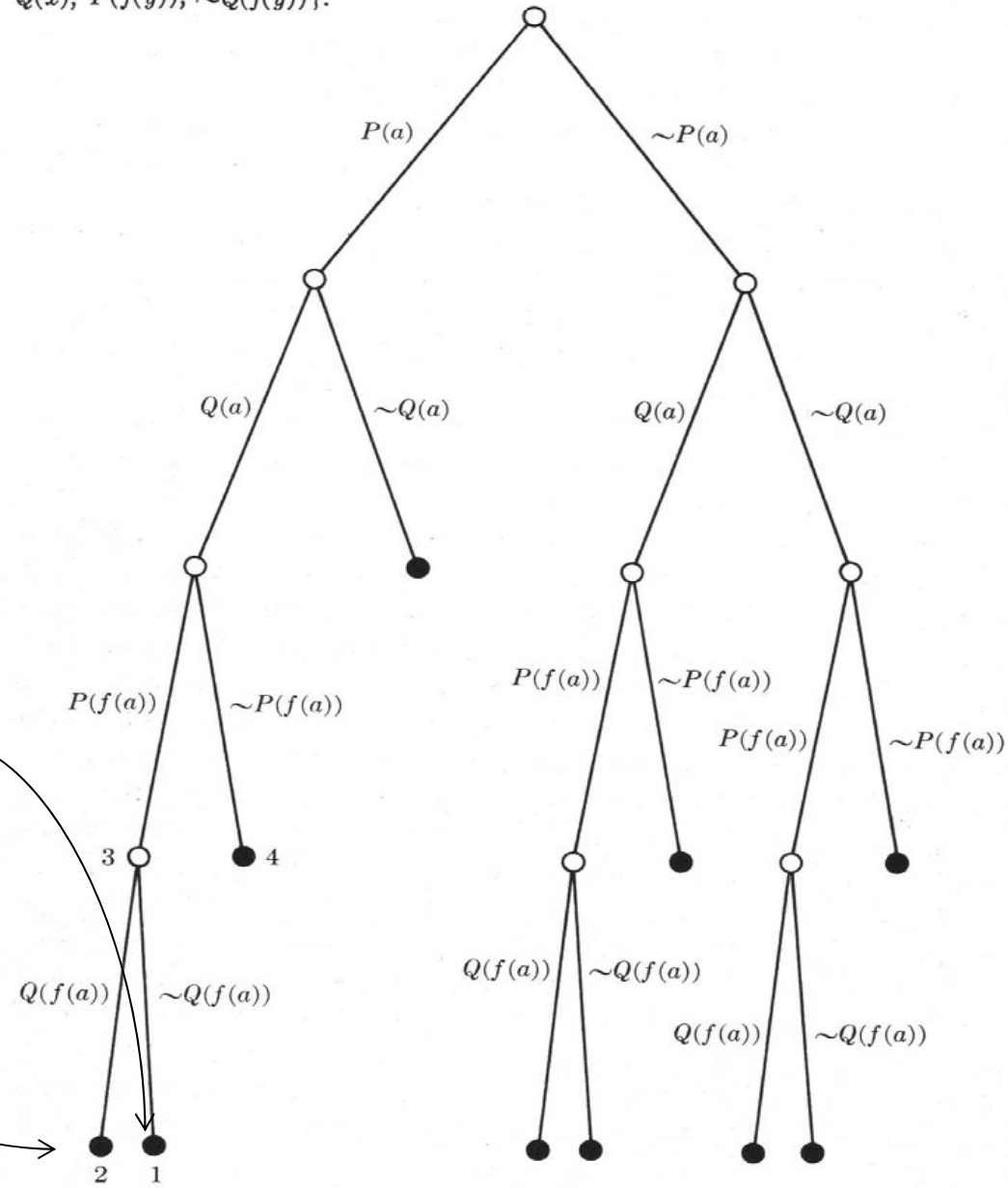
Fig. 6.2 – Un albero semantico chiuso dell'insieme insoddisfacibile $\{\sim P(x) \vee Q(x),\ P(f(y)),\ \sim Q(f(y))\}$.



Un altro risultato molto importante è che se alle clausole di S si aggiunge una clausola C ottenuta come risolvente di due clausole di S, l'insieme unione $S' = S \cup \{C\}$ è ancora insoddisfacibile e *l'albero semantico di S' ha un numero di nodi sopra i nodi di fallimento che è strettamente minore del numero corrispondente nell'albero di S.*

Esempio: si consideri il caso precedente:

Fig. 6.2 – Un albero semantico chiuso dell'insieme insoddisfacibile $\{\sim P(x) \vee Q(x),\ P(f(y)),\ \sim Q(f(y))\}$.



fallisce per ~P(x)∨Q(x)

fallisce per ~Q(f(y))

Da queste due posso inferire (segue logicamente) ~P(f(y)), che fallisce al nodo precedente.

Questo processo può essere ripetuto, pervenendo all'insieme vuoto: l'albero semantico dell'insieme vuoto ha zero nodi sopra quelli di fallimento, per cui è subito dimostrata l'insoddisfacibilità.

Praticamente l'applicazione ripetuta della risoluzione, se porta all'insieme vuoto, fornisce immediatamente la verifica dell'insoddisfacibilità, e quindi evita la costruzione dell'albero semantico.

Si dimostra che il principio di risoluzione è *corretto* e *completo*.

Corretto perché se la clausola vuota viene prodotta, l'insieme originario è sicuramente insoddisfacibile;

completo perché se l'insieme originario è insoddisfacibile, la clausola vuota viene prima o poi prodotta.

Algoritmo di unificazione

Nella logica proposizionale è facile verificare la contraddizione di due formule atomiche: L e ¬L. Nella logica dei predicati entrano in gioco i legami delle variabili.

Esempio:

uomo(Giovanni) ¬uomo(Giovanni)

Contraddizione

uomo(Giovanni) ¬uomo (Felix)

no-contraddizione

L'unificazione confronta due formule atomiche per determinare se esiste un insieme di sostituzioni che le rende identiche.

L’idea su cui si basa l’algoritmo di unificazione è di immaginare che ogni formula atomica sia rappresentata da una lista: in prima posizione c’è il predicato, seguito poi dagli argomenti (questi possono essere singoli elementi (cioè atomi in LIPS) o altre liste).

Esempio:
(cercadiassassinare Marco Cesare)
(cercadiassassinare Marco (capodi Roma)

Si controllano innanzitutto i primi elementi. Se uguali, si procede. Se diversi, non sono unificabili, come ad esempio

(cercadiassassinare Marco Cesare)
(odia Marco Cesare)

Se ok, si continua il confronto a coppie degli elemento delle liste (l'algoritmo è ricorsivo).

Le regole sono:

- costanti, funzione e predicati si possono unificare solo se identici
- una variabile può unificarsi:
  - con un'altra variabile
  - con una qualunque costante
  - con una funzione (purché non contenga alcun esemplare della variabile che si sta unificando)

Complicazione: la stessa sostituzione deve valere per l'intera formula atomica.

Se si trova una sostituzione, bisogna applicarla al resto della formula atomica prima di continuare con l'unificazione.

Esempio. Siano date le espressioni

(p x x)

(p y z)

I predicati si unificano. Si confrontano x e y. Si decide di sostituire x con y.

Si scrive la sostituzione come y/x (leggere: y sostituisce x).

Al passo successivo, non si può avere il confronto x e z e la sostituzione z/x (cioè x con z): le due sostituzioni y/x e z/x insieme non sono coerenti.

Occorre prima sostituire la seconda x con y, poi si può fare l'ulteriore sostituzione z/y.

Si avrà unica composizione di sostituzioni

(z/y) (y/x)

In generale, la sostituzione

$$(a_1/a_2 , a_3/a_4 \ldots) \quad (b_1/b_2 , b_3/b_4 \ldots)$$

significa applicare tutte le sostituzioni della lista più a destra; prendere il risultato e applicare quelle della lista successiva, e così via.

Osservazione. L'obiettivo è una sostituzione che causi l'unificazione di due parti letterali. Si possono avere più sostituzioni.

Esempio: le parti laterali

odia(x, y)

odia(Marco, z)

potrebbero essere unificate con una qualunque delle seguenti sostituzioni:

(Marco/x, z/y)
(Marco/x, y/z)
(Marco/x, Cesare/y, Cesare/z)
(Marco/x, Polonio/y, Polonio/z)

(le prime due sono più generale, le seconde due più restrittive).

È preferibile generare l'unificazione più generale possibile.

Logica dei predicati: la sostituzione

Chiamiamo *espressione* genericamente un termine, un atomo, una funzione, un predicato, una formula.

Costituisce una *sostituzione* (substitution) la serie i coppie ordinate

$$\alpha=\{t_1/y_1, t_2/y_2, \ldots, t_n/y_n\}$$

dove t sono termini, le y sono variabili e $t_i \neq y_i$ per ogni i = 1 …n

Quando una sostituzione α è applicata ad un'espressione K, ogni occorrenza di y_i in K è sostituita da t_i.

L'espressione risultante Kα è detta *istanza* di K.

Esempio

α={a/x, g(u)/z} applicata a

P(w, f(x), z) fornisce

P (w, f(a), g(u))

Esiste la sostituzione vuota ε:

Kε = K

Composizione di sostituzioni

Siano α e β due sostituzioni.

Si dice composizione di α e β ($\alpha \circ \beta$) la sostituzione tale che

$$K(\alpha \circ \beta) = (K\alpha)\beta$$

per ogni espressione K.

In pratica, si applica prima α e poi β.

$\alpha \circ \beta$ si può derivare da α e β con la seguente procedura:

Siano:

$\alpha = \{t_1/y_1, t_2/y_2 \ldots t_n/y_n\}$

$\beta = \{s_1/x_1, s_2/x_2 \ldots s_m/x_m\}$

Si eseguono i seguenti due passi:

passo 1

Da α e β si costituiscono i tre set λ_1 , λ_2 e λ_3 così:

$\lambda_1 = \{t_1\beta/y_1, t_2\beta/y_2, \ldots, t_n\beta/y_n, s_1/x_1 , s_2/x_2 , \ldots, s_m/x_m\}$

$\lambda_2 = \{t_i\beta/y_i \mid t_i\beta/y_i \in \lambda_1 \;\&\; t_i\beta = y_i \quad \text{per } 1 \leq i \leq n\}$
(cioè se $t_i\beta/y_i$ è contenuta in λ_i AND
$t_i\beta = y_i$: in pratica identifica in λ_1 le sostituzioni y_i/y_i)

$\lambda_3 = \{s_i/x_i \mid s_i/x_i \in \lambda_1 \ \& \ x_i \in \{y_1 , y_2 \ldots y_n\}\}$
(in pratica identifica in β sostituzioni che già compaiono in λ_1 e l'elemento sostituito era già in α)

passo 2

Si deriva la composizione $\alpha \circ \beta$ come

$\alpha \circ \beta = \lambda_1 - \lambda_2 - \lambda_3$

Esempio: dati

$\alpha = \{z/u, h(u)/w\}$

$\beta = \{a/u, z/w, u/z\}$

derivare la composizione $\alpha \circ \beta$ e applicarla a

$K = P(u, w, f(z))$

Dimostrare inoltre che

$K(\alpha \circ \beta) = (K\alpha)\beta$

Passo 1

$\lambda_1 = \{z\beta/u, h(u)\beta/w, \quad a/u, z/w, u/z\}$ $\quad \alpha = \{z/u, h(u)/w\}$

$\quad = \{u/u, h(a)/w, a/u, z/w, u/z\}$ $\quad \beta = \{a/u, z/w, u/z\}$

$\lambda_2 = \{u/u\}$

$\lambda_3 = \{a/u , z/w\}$

Passo 2

$\alpha \circ \beta = \lambda_1 - \lambda_2 - \lambda_3 = \{h(a)/w, u/z\}$ $\quad K = P(u, w, f(z))$

$K(\alpha \circ \beta) = P(u, h(a), f(u))$

Altrimenti:

$K\alpha = P(z, h(u), f(z))$

$(K\alpha)\beta = P(u, h(a), f(u))$ $\quad$ q.d.e.

Proprietà delle composizioni di sostituzioni:

1. $(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$
   (associatività)
2. $\varepsilon \circ \alpha = \alpha$
   (ε sostituzione vuota)
3. $\alpha \circ \varepsilon = \alpha$
   (identità a sinistra e a destra)
4. $\alpha \circ \beta \neq \beta \circ \alpha$
   (solitamente non commutativa)

Unificazione

Le espressioni K_1 e K_2 sono unificabili se e solo se esiste una sostituzione γ tale per cui $K_1\gamma = K_2\gamma$.

La sostituzione γ è detto *unificatore*.

$K_1\gamma$ e $K_2\gamma$ sono *istanze comuni* delle due espressioni.

Unificatori più generali

Un unificatore δ delle espressioni K_1 e K_2 è detto mgu (*most general unifier*) se e solo se, per ogni unificatore γ di K_1 e K_2, l'istanza comune di $K_i\delta$ è più generale che l'istanza comune di $K_i\gamma$.

In altre parole $K_i\gamma$ è una istanza di $K_i\delta$, ovvero esiste una sostituzione θ tale per cui
$K_i\gamma = (K_i\delta)\theta = K_i(\delta \circ \theta)$.

Esempio: se una variabile può essere sostituita da una variabile o una costante, la prima è da preferire (è più generale).

Un algoritmo per l'unificazione è fornito dalla procedura che segue.

La procedura UNIFICA(L1, L2) fornisce una lista che rappresenta la composizione delle sostituzioni attuate durante l'unificazione. Se la lista contiene NIL (è vuota) l'unificazione è avvenuta senza sostituzioni. Se la lista contiene F, l'unificazione è impossibile (fail).

UNIFICA(L1, L2)

1. Se L1 o L2 è un atomo allora si comporta nel modo seguente:
   a) se L1 e L2 sono identici allora ritorna NIL
   b) altrimenti se L1 è una variabile allora fa quanto segue:
      i. se L1 compare in L2 allora dà F, altrimenti ritorna (L2/L1)
   c) altrimenti se L2 è una variabile allora fa quanto segue:
   d) se L2 compare in L1 allora dà F, altrimenti ritorna (L1/L2)
   e) altrimenti ritorna F.

2. Se lunghezza (L1) non è uguale a lunghezza (L2) allora ritorna F.
3. Dà a SOST il valore NIL (alla fine di questa procedura, SOST conterrà tutte le sostituzioni usate per unificare L1 e L2).
4. Per i := 1 fino al numero di elementi di L1 fà quanto segue:
   a) chiama unifica con l'iesimo elemento di L1 e l'iesimo elemento di L2, mettendo il risultato in S
   b) se S = F allora ritorna F
   c) se S non è uguale a NIL allora fà così:
      i. applica S a ciò che rimane sia di L1 che di L2
      ii. SOST = CONCATENA(S, SOST)

Occorre un controllo di occorrenza (inserito nella procedura precedente).

Infatti si supponga di dover unificare le espressioni

(f, x, x)

(f, g(x), g(x))

Se si assume di sostituire x con g(x) e poi si cerca di applicare la sostituzione al resto dell'espressione, si ottiene una ricorsione infinita, poiché non sarà mai possibile eliminare x!

Esempio di unificazione:

Trovare la *mgu* delle espressioni

$K_1 = P(g(u), z, f(z))$

$K_2 = P(x, y, f(b))$

Sviluppando la mgu:

a) $\{g(u)/x\}$ unifica la prima sotto-espressione differente di K_1 e K_2

$$K_1\{g(u)/x\} = P(g(u), z, f(z))$$

$$K_2\{g(u)/x\} = P(g(u), y, f(b))$$

b) {y/z} unifica la successiva sotto-espressione differente

Occorre applicare la composizione:

$$\{g(u)/x\} \circ \{y/z\} = \{g(u)/x, y/z\}$$

$$K_1\{g(u)/x, y/z\} = P(g(u), y, f(y))$$
$$K_2\{g(u)/x, y/z\} = P(g(u), y, f(b))$$

c) {b/y} unifica la successiva sotto-espressione differente

Componendo le sostituzioni:

$\{g(u)/x, y/z\} \circ \{b/y\} =$

$\{g(u)/x, b/z, b/y\} = \delta$

che è la mgu richiesta.

Applicando δ a K_1 e K_2 si ottiene:

$K_i\delta = P(g(u), b, f(b))$ per $i = 1, 2$

Esempi di espressioni che non possono essere unificati.

Examples of expressions K_1 and K_2 that cannot be unified; procedure MGUNIFIER will return failure.		Cause of the failure in unification.	The step number of procedure MGUNIFIER in Figure 3.7 at which the procedure returns failure.
K_1	K_2		
P(x)	P(f(x))	To try to unify these, we need substitution {f(x)/x}. But that can cause indefinite recursion, as the expressions become P(f(x)) and P(f(f(x))).	2.2.1
P(f(x))	P(x)	ditto	2.3.1
P(x)	Q(x)	One predicate symbol P cannot be replaced by another predicate symbol Q.	2.4
P(f(x))	P(g(x))	One function symbol f cannot be replaced by another function symbol g.	2.4

Figure 3.9 Examples of expressions that cannot be unified. P is a predicate; f and g are functions; x is a variable.

Ancora sulla mgu.

Si è visto che in

$K_1 = P(g(u), z, f(z))$
$K_2 = P(x, y, f(b))$

la mgu $\delta = \{g(u)/x, b/z, b/y\}$

produce l'istanza comune generale:

$K_i\delta = P(g(u), b, f(b))$

Si poteva trovare come unificatore (generico)

$\gamma = \{g(c)/x, c/u, b/z, b/y\}$

Si può mostrare che $K_i\delta$ è più generale di $K_i\gamma$.

Infatti:

$K_i\gamma = P(g(c), b, f(b))$

Ora si può osservare che esiste una sostituzione

$\theta = \{c/u\}$ tale che

$K_i\gamma = (K_i\delta)\theta$

Pertanto $K_i\gamma$ è un'istanza di $K_i\delta$, e quindi $K_i\delta$ è più generale di $K_i\gamma$.

Risoluzione nella logica dei predicati

Due formule atomiche sono contraddittorie se l'una può essere unificata con la negazione dell'altra.

Esempio:

uomo(x) e ¬uomo(Fido)

sono contraddittorie, poiché uomo(x) e uomo(Fido) possono essere unificate.

(NB: si è formalizzato quanto conosciamo intuitivamente: uomo(x) non è sempre vero per tutti gli x. Infatti se x vale Fido, uomo(x) è falso!)

Si applica quindi l'unificatore prodotto dall'algoritmo di unificazione per generare la clausola risolvente.

Si rammenti che le variabili usate durante il processo di trasformazione delle espressioni in forme a clausole devono essere *differenziate*.

Esempio: si debbano risolvere le due clausole

1. uomo(Marco) [Marco è un uomo]

2. $\neg$uomo(x_1) $\vee$ mortale(x_1)
   [uomo(x_1) $\rightarrow$ mortale(x_1), ogni uomo è mortale]

uomo(Marco) e uomo(x_1) possono essere unificate con la sostituzione Marco/x_1

Se uomo(Marco) è vera, ¬uomo(Marco), nella 2ª formula, è falsa. Affinché la 2ª formula resti vera, deve essere vera mortale(Marco), che è la risolvente (il processo di risoluzione dovrà andare avanti per scoprire se quest'ultima affermazione è contrad-dittoria).

Nota. Si osservi che la 2ª clausola afferma che, per un dato x_1, è vera o $\neg$uomo(x_1) o mortale(x_1) e che per un x_1 per cui è vera $\neg$uomo(x_1), mortale(x_1) è irrilevante per la verità della clausola completa.

Si ribadisce dunque che se compaiono due esemplari della stessa variabile, occorre sottoporle a sostituzioni identiche.

Algoritmo di risoluzione per la logica dei predicati

Dato un insieme F di enunciati ed un enunciato S da provare:

1) Trasformare tutti gli enunciati di F in forma a clausole.
2) Negare S e trasformare il risultato in forma a clausole. Aggiungerlo all'in-sieme di clausole ottenuto al passo 1.

3) Ripetere i passi seguenti, fino a quando non viene trovata una contraddizione, oppure non si può più procedere, oppure è già stata utilizzata una predeterminata quantità di risorse:
   a) Selezionare due clausole, e chiamarle clausole genitori.

b) Risolvere insieme: la risolvente sarà la disgiunzione di tutte le parti letterali di entrambe le clausole genitori compiute le sostituzioni appropriate e con la seguente eccezione: se vi è una coppia di formule atomiche T1 e ¬T2 tali che una delle clausole genitori contiene T1 e l'altra contiene T2 e se T1 e T2 sono unificabili, allora né T1 né T2 dovrebbero apparire nella risolvente. Chiameremo T1 e T2 *formule atomiche complementari*. Usare la sostituzione prodotta dall'unificazione per creare la risolvente.

c) Se la risolvente è la clausola vuota, allora è stata trovata una contraddizione. In caso contrario, aggiungerla all'insieme di clausole a disposizione della procedura.

Strategie da adottare nella scelta delle clausole da risolvere insieme per accelerare il processo:

- Selezionare coppie di clausole che contengono formule atomiche complementari.
  Si possono, ad esempio, indicizzare le clausole in base ai predicati che contengono, con l'aggiunta di una indicazione che dice se il predicato è affermato o negato (memorie associative?!?)

- eliminare subito le clausole generate che non possono partecipare a risoluzioni successive e in particolare:
  - le tautologie (sempre soddisfacibili)
  - le clausole sussunte da altre clausole (esempio: $p \vee q$ è sussunta da p)

- adoperare la *strategia dell'insieme di supporto*: quando possibile, risolvere con una delle clausole che è parte dell'enunciato che si sta tentando di refutare oppure con una clausola generata da una risoluzione con una clausola di questo tipo
- adoperare la *strategia di preferenza delle clausole unitarie*: quando possibile, risolvere con clausole con una sola formula atomica (la risolvente avrà meno formule atomiche dei genitori, e quindi avvicina di più alla meta).

Esempio.

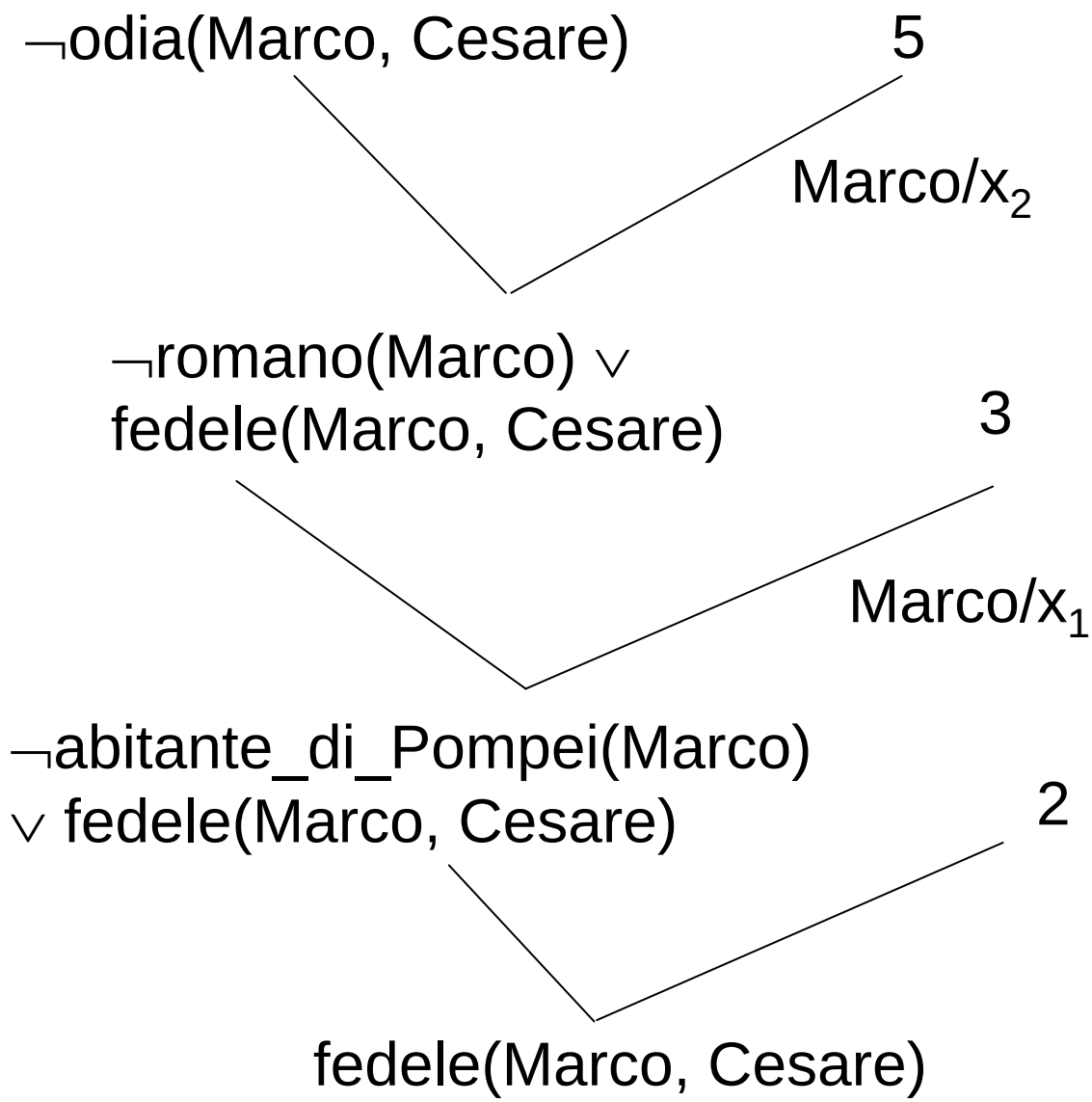
Dati i seguenti assiomi in forma a clausole:

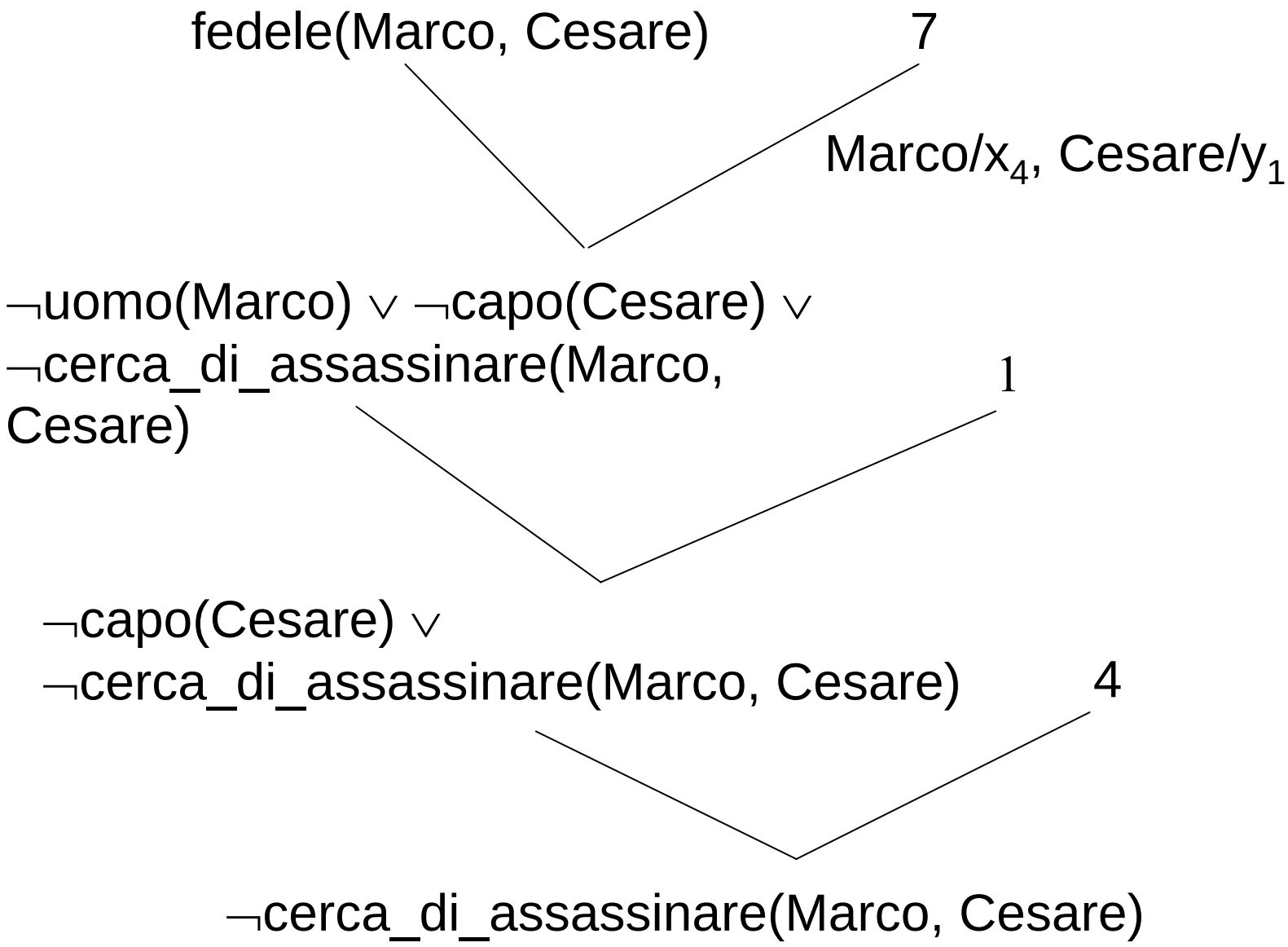
1. uomo (Marco)
2. abitante_di_Pompei (Marco)
3. $\neg$abitante_di_Pompei(x_1) $\vee$ romano(x_1)
4. Capo (Cesare)
5. $\neg$romano(x_2) $\vee$ fedele(x_2, Cesare)
   $\vee$ odia(x_2, Cesare)

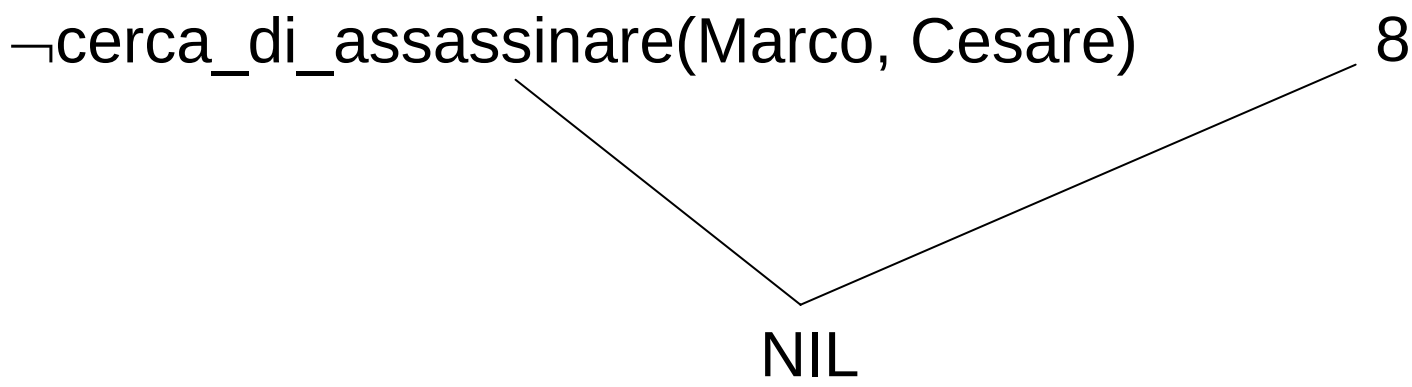
6. fedele(x_3, $f_1(x_3)$)
7. $\neg$uomo(x_4) $\vee$ $\neg$capo(y_1) $\vee$
   $\neg$cerca_di_assassinare(x_4, y_1) $\vee$
   $\neg$fedele(x_4, y_1)
8. cerca_di_assassinare (Marco, Cesare)

dimostrare l'enunciato:

odia(Marco, Cesare)


¬odia(Marco, Cesare)
5
Marco/x2
¬romano(Marco) ∨
fedele(Marco, Cesare)
3
Marco/x1
¬abitante_di_Pompei(Marco)
∨ fedele(Marco, Cesare)
2
fedele(Marco, Cesare)


fedele(Marco, Cesare)
7
Marco/x_4, Cesare/y_1
¬uomo(Marco) ∨ ¬capo(Cesare) ∨
¬cerca_di_assassinare(Marco, Cesare)
1
¬capo(Cesare) ∨
¬cerca_di_assassinare(Marco, Cesare)
4
¬cerca_di_assassinare(Marco, Cesare)



Si osservi che se avessimo dovuto rispondere alla domanda:
Marco odiava Cesare?

e avessimo posto il problema di dimostrare

¬odia(Marco, Cesare)

avremmo dovuto aggiungere la clausola

odia(Marco, Cesare)

così che il processo di dimostrazione non avrebbe funzionato (non vi sono clausole che contengono una parte letterale che comprenda ¬odia!).

In pratica

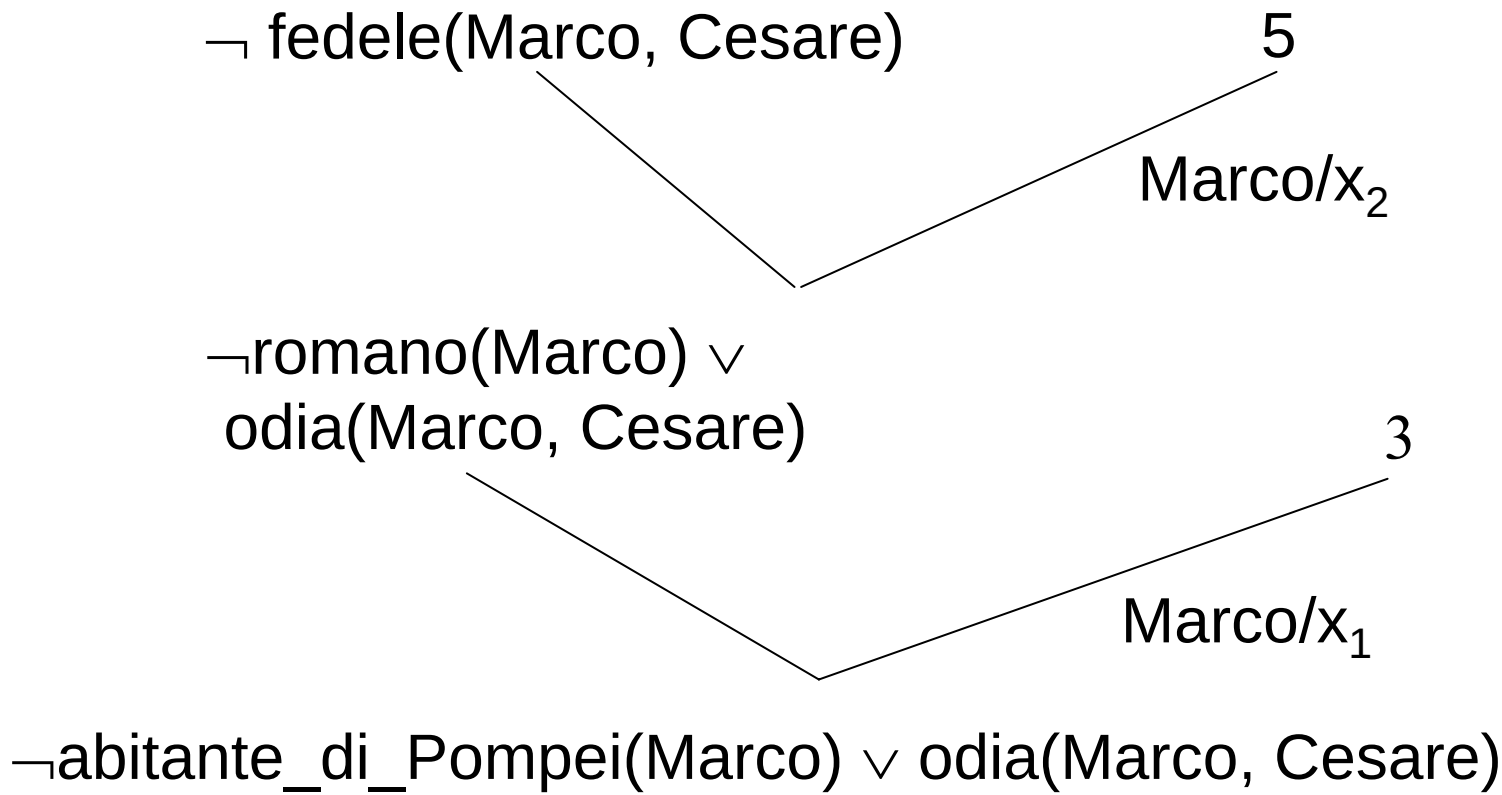
odia(Marco, Cesare)

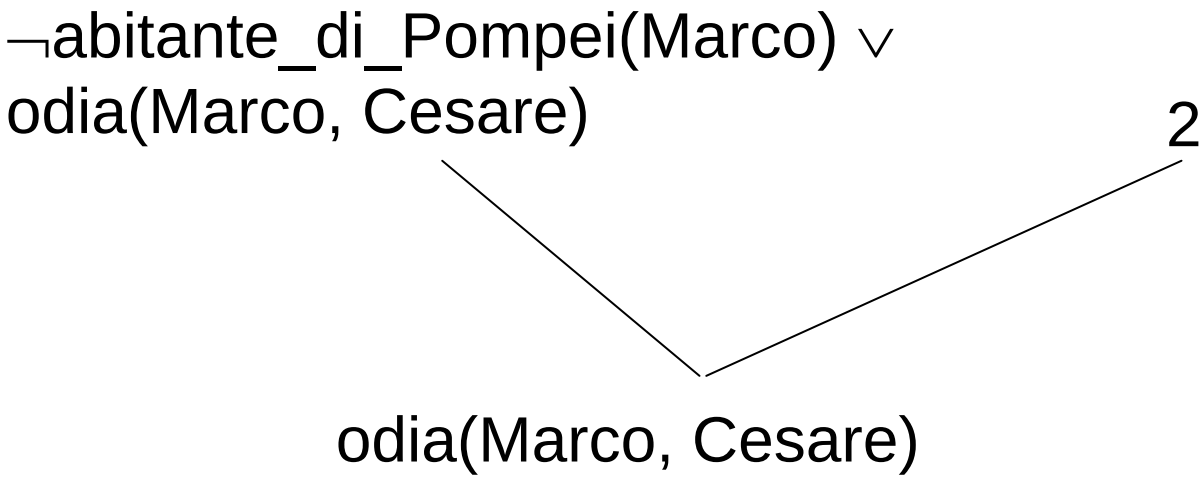
non produce contraddizione rispetto agli enunciati noti.

A volte ci si accorge di questa situazione non all'inizio, ma nel corso della prova.

Esempio. Con gli stessi assiomi precedenti, si vuole dimostrare l'enunciato:

fedele(Marco, Cesare)




¬abitante_di_Pompei(Marco) ∨
odia(Marco, Cesare)
2
odia(Marco, Cesare)

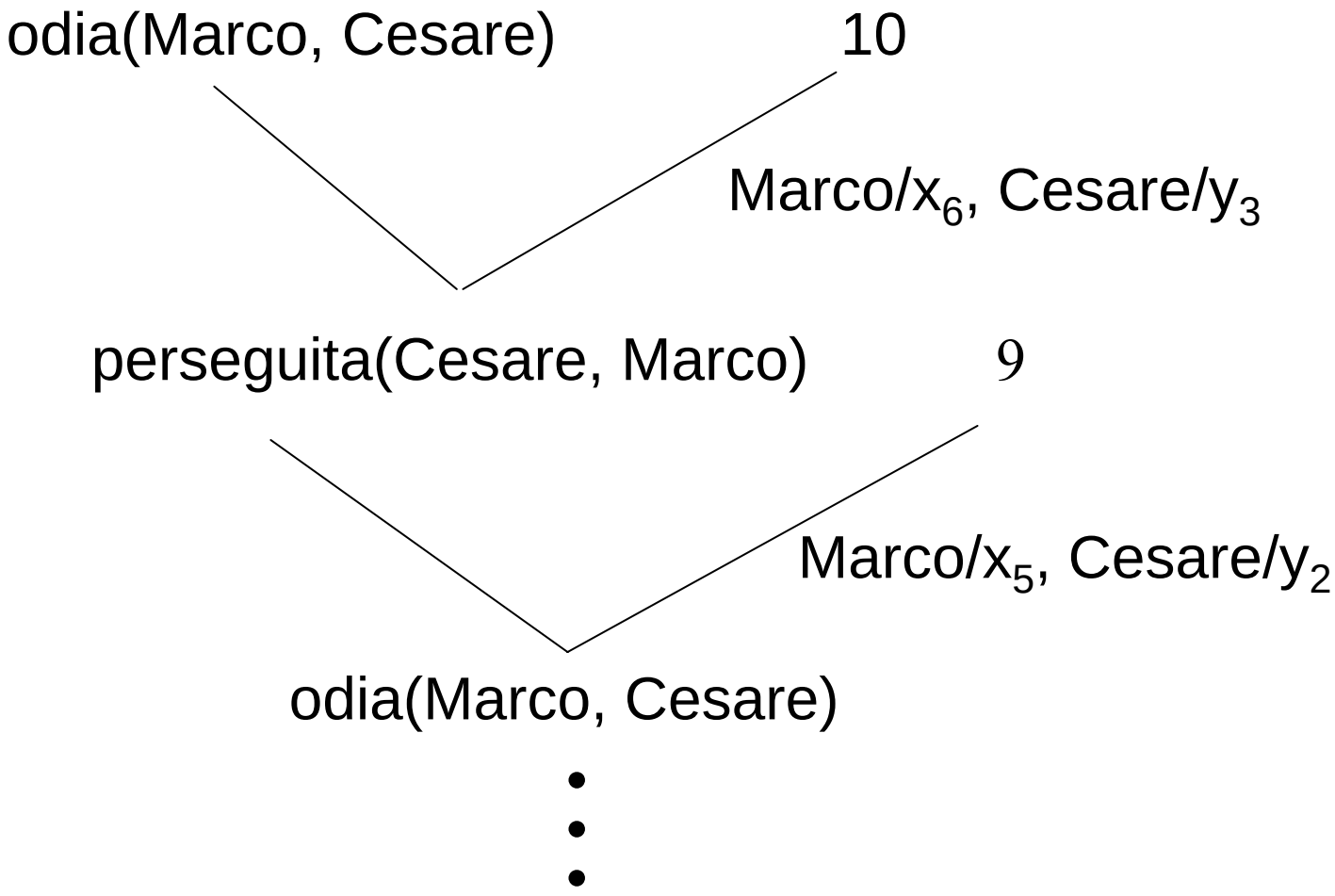
Può venire il sospetto che la base di conoscenza sia incompleta. Si supponga di aggiungere altri enunciati:

9. perseguita(x, y) → odia(y, x)
10. odia(x, y) → perseguita(y, x)

Trasformandoli in forma di clausole, si ottiene:

9. $\neg$perseguita(x_5, y_2) $\vee$ odia(y_2, x_5)
10. $\neg$odia(x_6, y_3) $\vee$ perseguita(y_3, x_6)

La prova precedente potrebbe continuare così:



Poiché le sole risolventi che è possibile generare coincidono con quelle generate prima, si desume che non può essere dimostrata la contraddizione.

Conclusione: il processo di prova dovrebbe effettuare la verifica che le risolventi non siano state già generate!

Ancora sulla necessità di standardizzare le variabili (differenziandole). Si consideri il seguente esempio.

Siano date le seguenti asserzioni:

1. $padre(x, y) \rightarrow \neg donna(x)$
   $\neg padre(x, y) \vee \neg donna(x)$
2. $madre(x, y) \rightarrow donna(x)$
   $\neg madre(x, y) \vee donna(x)$
3. madre(Chris, Mary)

Si voglia dimostrare

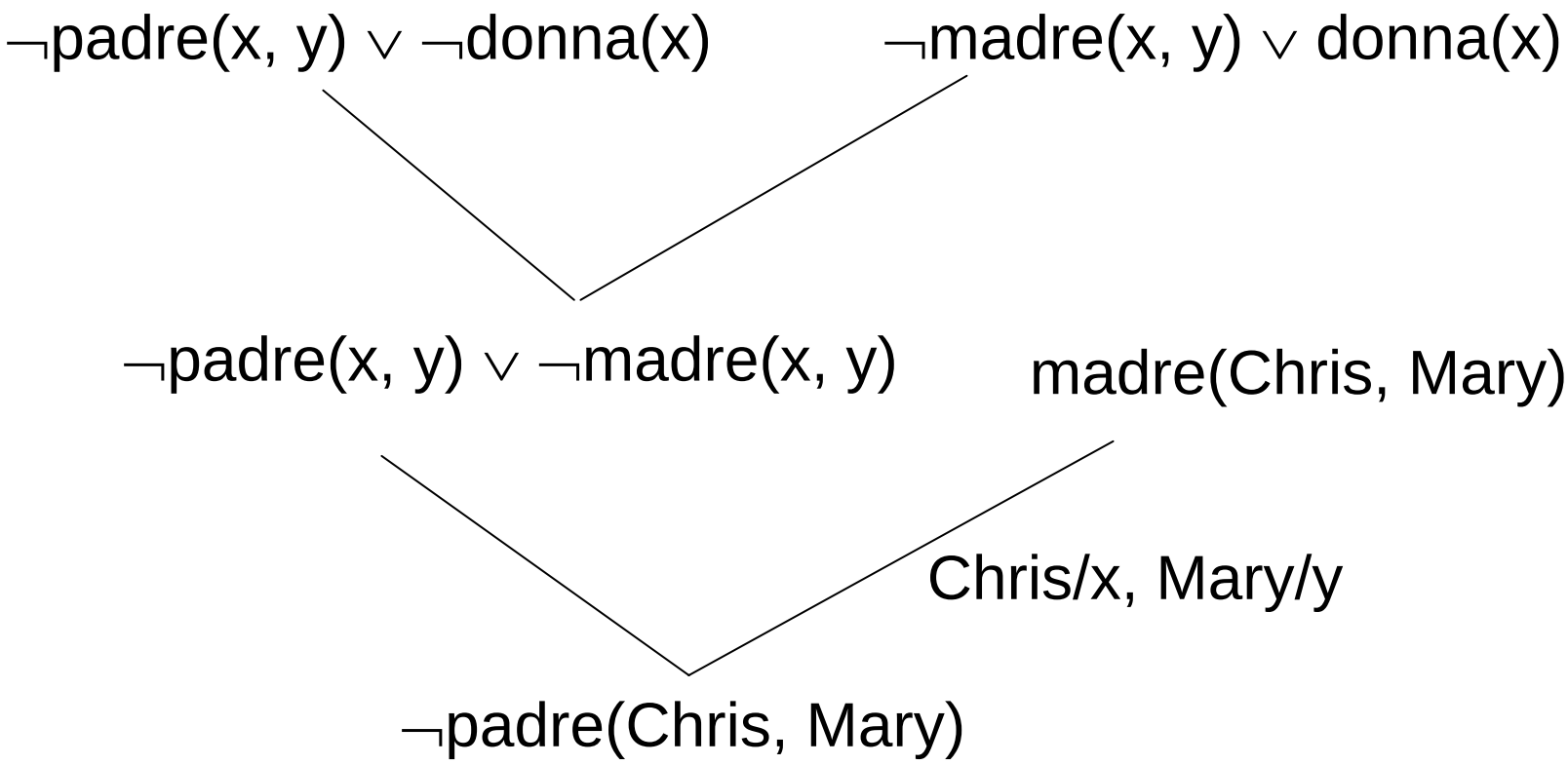
$\neg padre(Chris, Bill)$

Occorre inserire nella base di conoscenza:

4. padre(Chris, Bill)

(contraddizione intuitiva: Chris non può essere madre e padre!)

Se però proviamo a dimostrare la contraddittorietà della base di conoscenza:



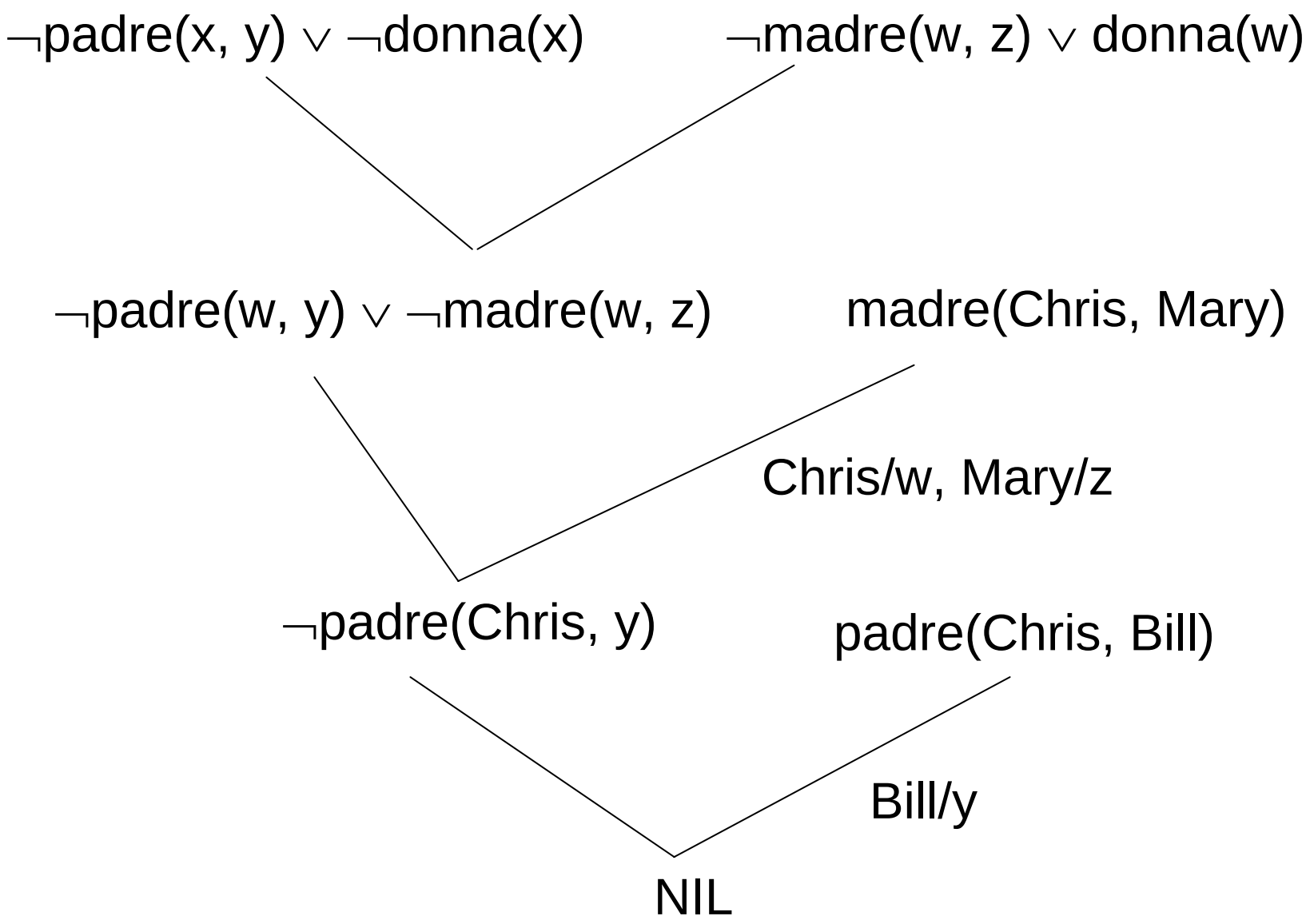
(NOTA: padre(Chris, Bill), presente nella base di conoscenza, non unifica con padre(Chris, Mary)!)

Dov'è l'errore? La clausola 1 e la clausola 2 contengono la stessa variabile y: la sostituzione al II passo produce una clausola troppo restrittiva che non evidenzia la contraddizione nella base dati.

Riscrivendo invece la clausola 2) come

$\neg$madre(w, z) $\vee$ donna(w)

si ha:


¬padre(x, y) ∨ ¬donna(x)
¬madre(w, z) ∨ donna(w)
¬padre(w, y) ∨ ¬madre(w, z)
madre(Chris, Mary)
Chris/w, Mary/z
¬padre(Chris, y)
padre(Chris, Bill)
Bill/y
NIL

Funzioni computabili, predicati computabili e relazioni di uguaglianza possono migliorare l'efficienza della risoluzione.

In questi casi, alla regola di risoluzione si possono aggiungere due modi di generare nuove clausole:

- sostituire un valore con un altro uguale

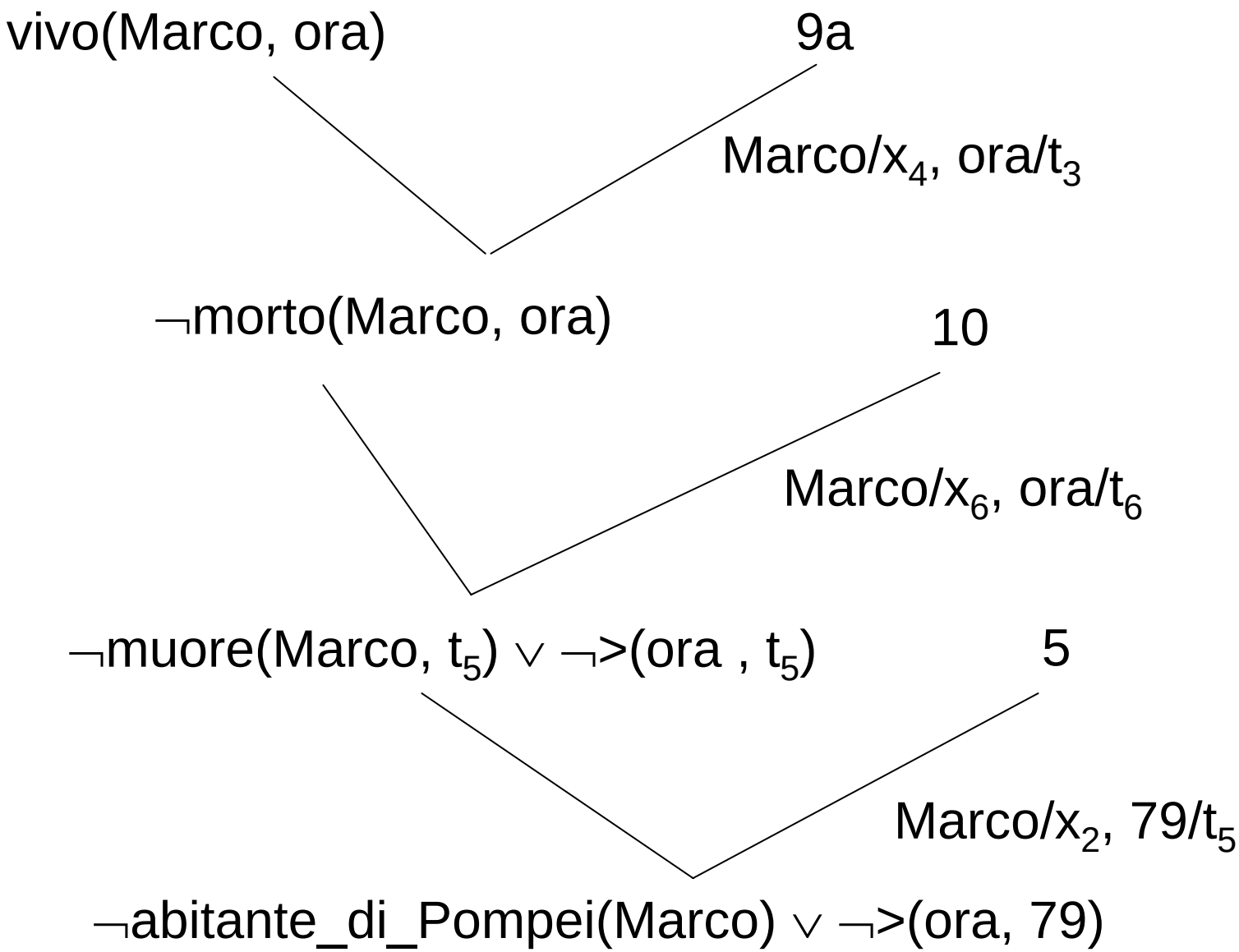
- ridurre i predicati computabili: se la valutazione porta al valore False, il predicato può essere omesso (aggiungere $\vee$ F non modifica il valore di verità di una disgiunzione); se invece la valutazione porta a True, allora la clausola generata è una tautologia che non può portare ad una contraddizione.

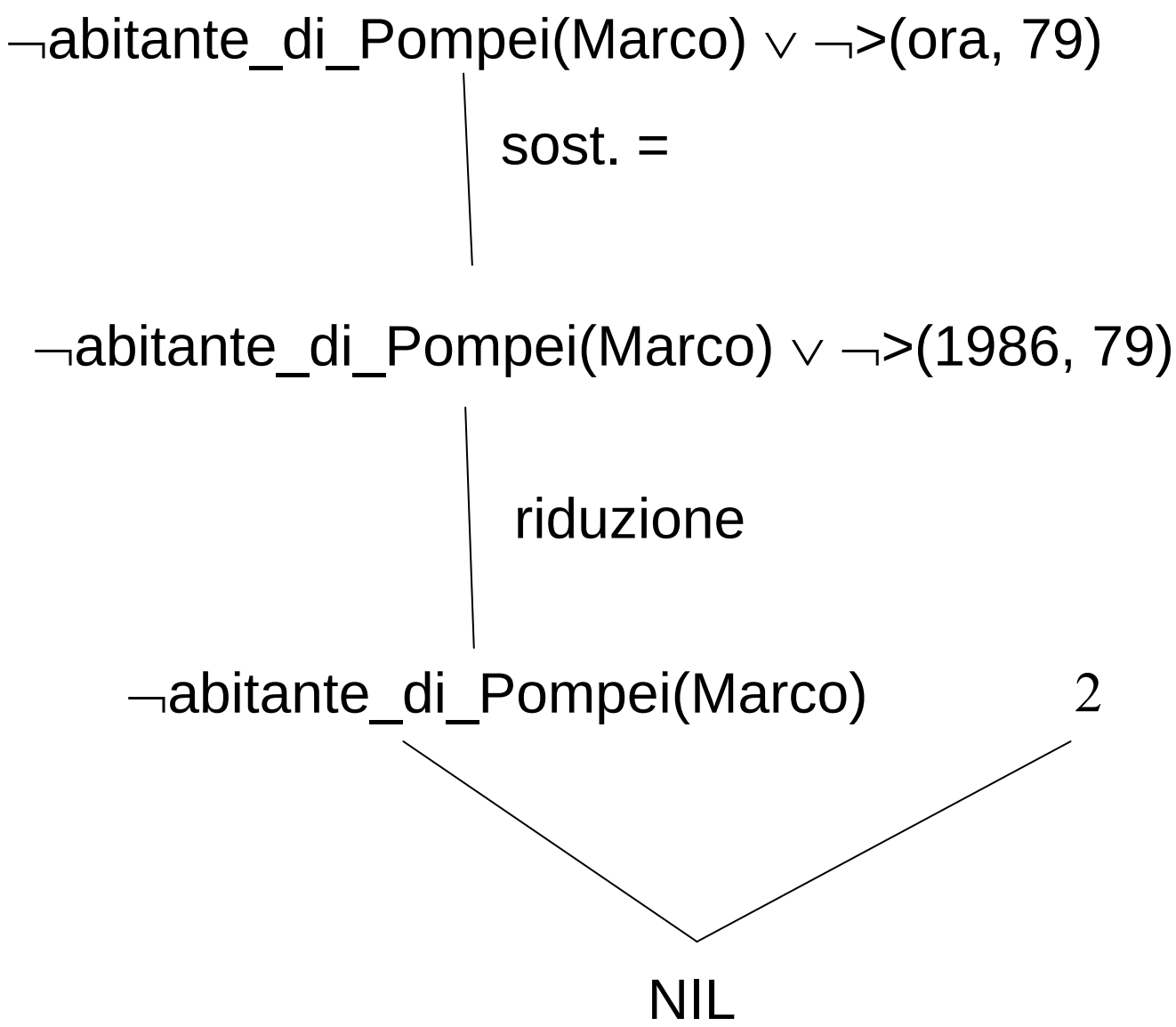
Esempio. Dati gli assiomi:

1. uomo(Marco)
2. abitante_di_Pompei(Marco)
3. nato(Marco, 40)
4. $\neg$uomo(x_1) $\vee$ mortale(x_1)
5. $\neg$ abitante_di_Pompei(x_2) $\vee$ muore(x_2, 79)
6. eruzione (vulcano, 79)

7. $\neg mortale(x_3) \vee \neg nato(x_3, t_1) \vee \neg >(t_2\text{-}t_1, 150) \vee morto(x_3, t_2)$
8. Ora = 1986
9.
   a. $\neg vivo(x_4, t_3) \vee \neg morto(x_4, t_3)$
   b. $vivo(x_5, t_4) \vee morto(x_5, t_4)$
10. $\neg muore(x_6, t_5) \vee \neg >(t_6 , t_5) \vee morto(x_6, t_6)$

dimostrare: $\neg vivo(Marco, ora)$


vivo(Marco, ora)
9a
Marco/x_4, ora/t_3
¬morto(Marco, ora)
10
Marco/x_6, ora/t_6
¬muore(Marco, t_5) ∨ ¬>(ora , t_5)
5
Marco/x_2, 79/t_5
¬abitante_di_Pompei(Marco) ∨ ¬>(ora, 79)


¬abitante_di_Pompei(Marco) ∨ ¬>(ora, 79)
sost. =
¬abitante_di_Pompei(Marco) ∨ ¬>(1986, 79)
riduzione
¬abitante_di_Pompei(Marco)
2
NIL

Osservazione:

il metodo di risoluzione riduce il numero di sostituzioni rispetto a quelle date dal teorema di Herbrand, ma non elimina la necessità di tentare più di una sostituzione.

Se, ad esempio, agli assiomi

1. uomo (Marco)
2. abitante_di_Pompei (Marco)
3. $\neg$abitante_di_Pompei(x_1) $\vee$ romano(x_1)
4. Capo (Cesare)
5. $\neg$romano(x_2) $\vee$ fedele(x_2, Cesare) $\vee$ odia(x_2, Cesare)
6. fedele(x_3, $f_1(x_3)$)

7. $\neg$uomo(x_4) $\vee$ $\neg$capo(y_1) $\vee$
$\neg$cerca_di_assassinare(x_4, y_1) $\vee$
$\neg$fedele(x_4, y_1)

8. cerca_di_assassinare (Marco, Cesare)

si aggiungono gli assiomi

odia(Marco, Paolo)

odia(Marco, Giulio)

e si voglia provare che Marco odia qualche capo, occorre inserire tra gli assiomi:

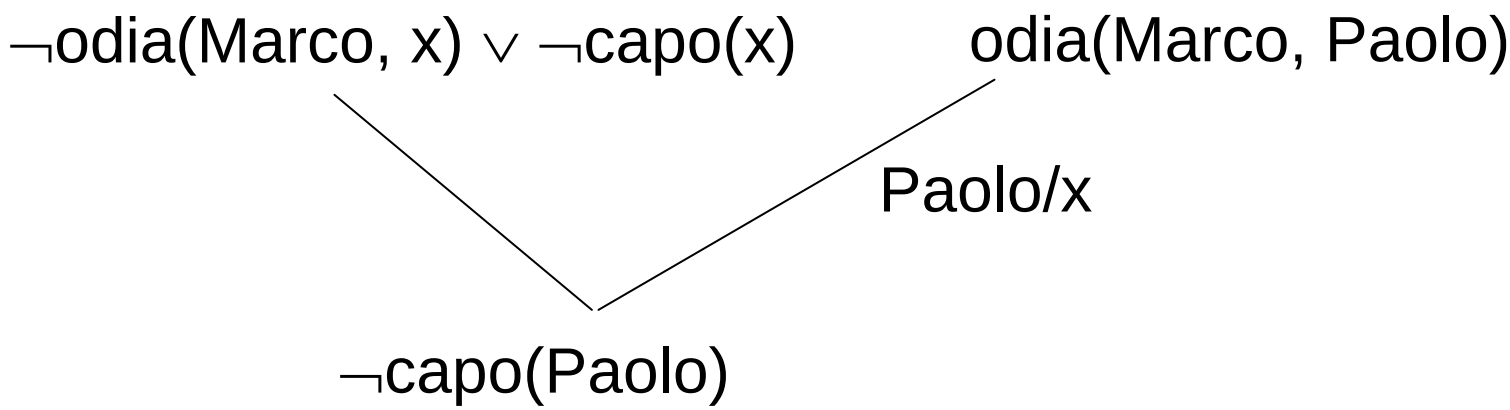
$\neg\exists x\ (odia(Marco, x) \wedge capo(x))$

che, ridotto in forma a clausola, diventa:

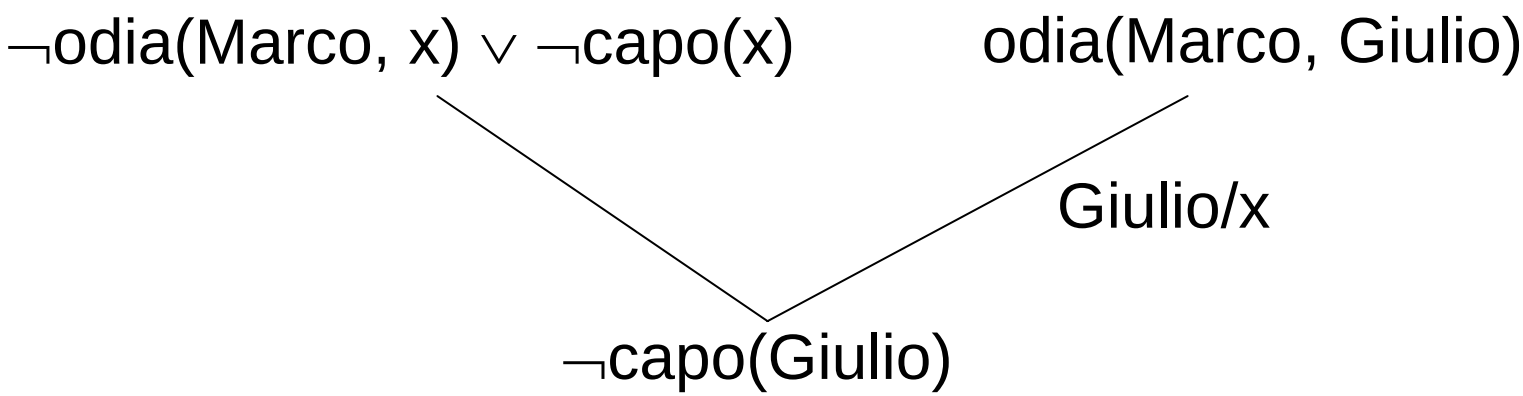
$\neg odia(Marco, x) \vee \neg capo(x)$

e quindi si possono avere i seguenti tentativi:

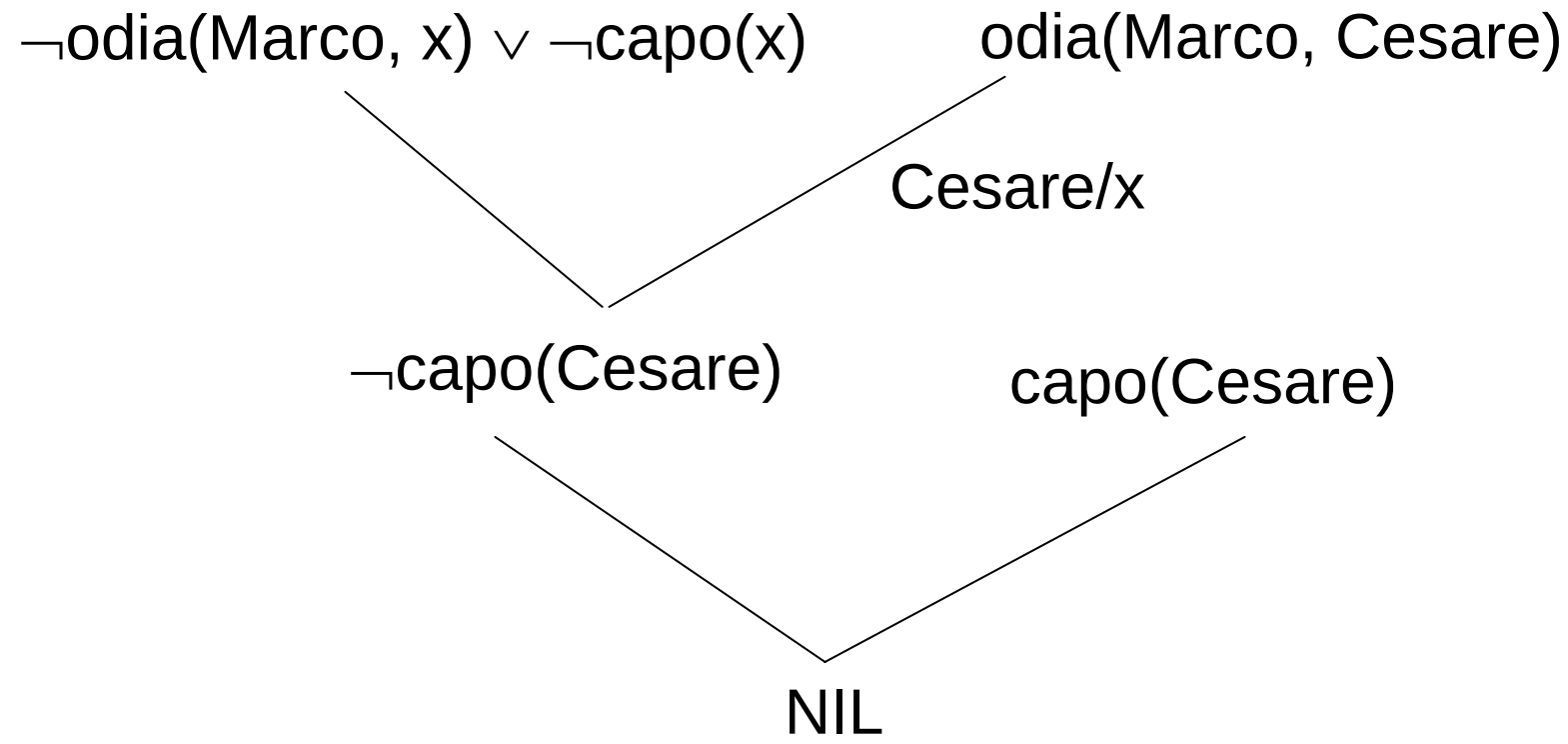
a)


¬odia(Marco, x) ∨ ¬capo(x)
odia(Marco, Paolo)
Paolo/x
¬capo(Paolo)

b)


¬odia(Marco, x) ∨ ¬capo(x)
odia(Marco, Giulio)
Giulio/x
¬capo(Giulio)

c)


¬odia(Marco, x) ∨ ¬capo(x)
odia(Marco, Cesare)
Cesare/x
¬capo(Cesare)
capo(Cesare)
NIL

Risposta alle domande

Non è sufficiente una risposta SI/NO, ma occorre "riempire caselle vuote".

Esempio
la domanda: Quando morì Marco?

si può porre come

muore(Marco, ??)

occorre sostituire ?? Con anno(79)

Come porre il problema?

Occorre innanzi tutto provare che Marco morì, cioè che è vero

$$\exists t \text{ muore}(\text{Marco}, t)$$

Usando la risoluzione, si può dimostrare che

$$\neg\exists t \text{ muore}(\text{Marco}, t)$$

produce una contraddizione. Ma rispetto a quale enunciato?

Si possono avere più scelte.

I scelta

$\forall$t muore(Marco, t)

(si verifica che ci sono molti t in cui Marco morì)

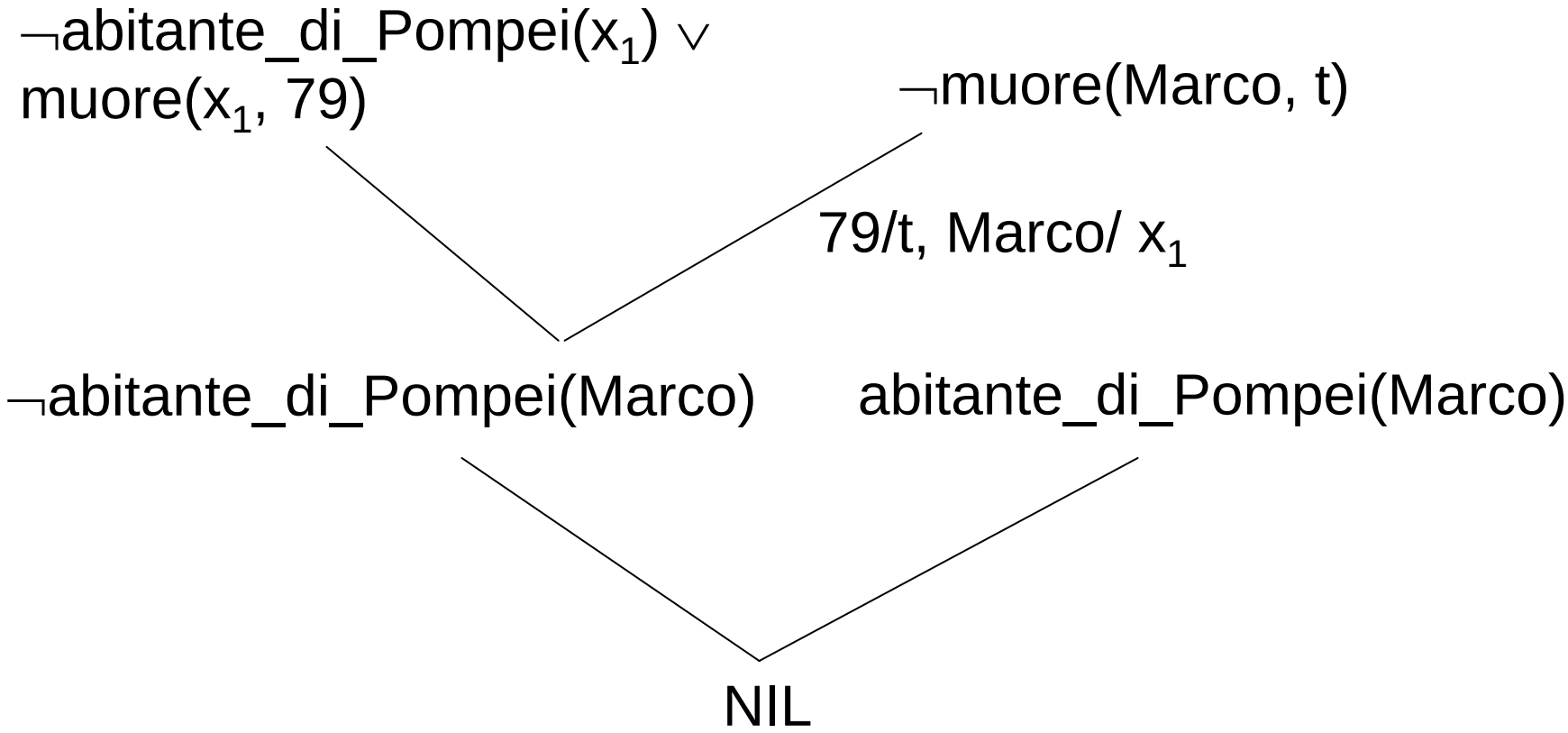
II scelta

Muore(Marco, data)

(la data che dimostra la contraddizione è la risposta)

Nell'esempio di risoluzione che segue la risposta si trova percorrendo all'indietro l'albero risolutivo (la si ricava dal processo di unificazione):

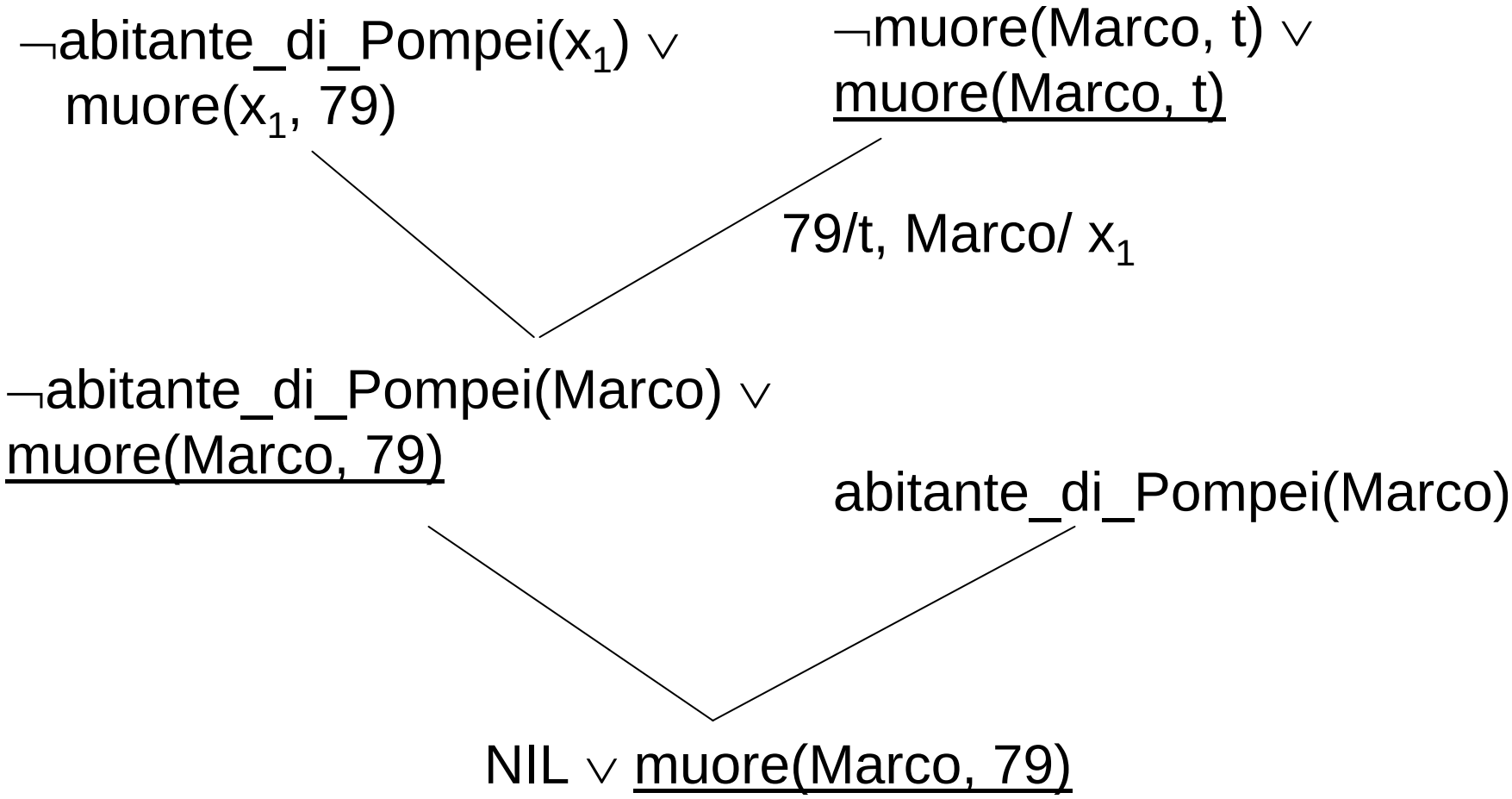
$\neg\exists t$ muore(Marco, t) $\equiv$ $\forall t \neg$muore(Marco, t)


¬abitante_di_Pompei(x1) ∨
muore(x1, 79)
¬muore(Marco, t)
79/t, Marco/ x1
¬abitante_di_Pompei(Marco)
abitante_di_Pompei(Marco)
NIL

Per le risposte alle domande, si può usare il metodo di aggiungere all'espressione utilizzata per trovare la contraddizione un'espressione in più, proprio quella che si sta cercando di provare.

Questa parte non viene mai usata nel processo di risoluzione (è quella sottolineata), ma solo nelle unificazioni.

Il processo di risoluzione diventa:


¬abitante_di_Pompei(x_1) ∨
muore(x_1, 79)
¬muore(Marco, t) ∨
muore(Marco, t)
79/t, Marco/ x_1
¬abitante_di_Pompei(Marco) ∨
muore(Marco, 79)
abitante_di_Pompei(Marco)
NIL ∨ muore(Marco, 79)

È ancora importante rilevare come la conformazione della base di conoscenza permette o meno di rispondere a delle domande.

Esempio:

domanda: Cosa successe nel 79 d.C.?

Formalmente:

$$\exists x \text{ evento}(x, 79)$$

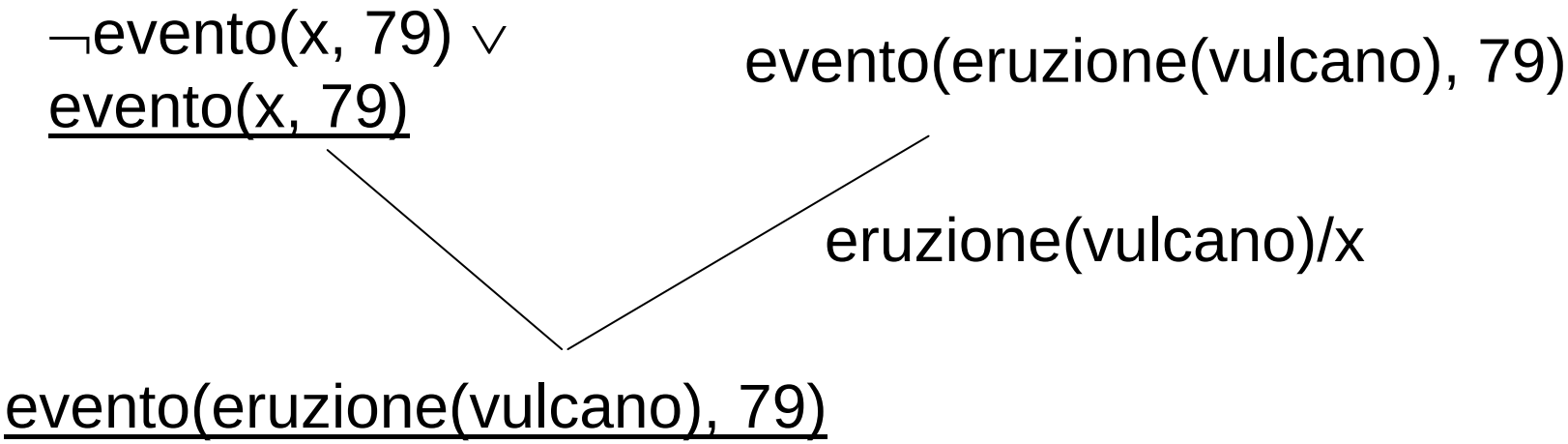
Ma non c'è nessun enunciato nella forma evento(x, y).

Se invece di

eruzione(vulcano, 79)

si avesse

evento(eruzione(vulcano), 79)



ESERCIZIO

Dimostrare che

se nessun cetaceo è un pesce e tutti i delfini sono cetacei

allora

nessun pesce è un delfino.

Soluzione:

1. nessun cetaceo è un pesce

   $\neg(\exists x)(C(x) \wedge P(x))$

   (non esiste un x che sia insieme cetaceo e pesce)

   per ridurlo alla forma a clausole:

   - spostamento della negazione:

     $(\forall x) \neg(C(x) \wedge P(x))$

   - applicazione della negazione alle singole lettere:

     $(\forall x) (\neg C(x) \vee \neg P(x))$ (clausola)

2. tutti i delfini sono cetacei:
   $(\forall x)(D(x) \rightarrow C(x))$
   (essere delfino implica essere cetaceo)

   – eliminazione dell'implicazione:
   $(\forall x)(\neg D(x) \vee C(x))$

   – differenziazione delle variabili:
   $(\forall y)(\neg D(y) \vee C(y))$ (clausola)

3. nessun pesce è un delfino
   $\neg(\exists x)(P(x) \wedge D(x))$
   (non esiste un x che sia insieme pesce e delfino)

   è l'asserzione da dimostrare, va quindi negata:
   $(\exists x)(P(x) \wedge D(x))$

   – skolemizzazione:
     $P(g) \wedge D(g)$ (poiché compare solo la variabile, la funzione di Skolem è la costante g che si suppone che "esista")
   – eliminazione dell'$\wedge$
     P(g) (clausole)
     D(g)

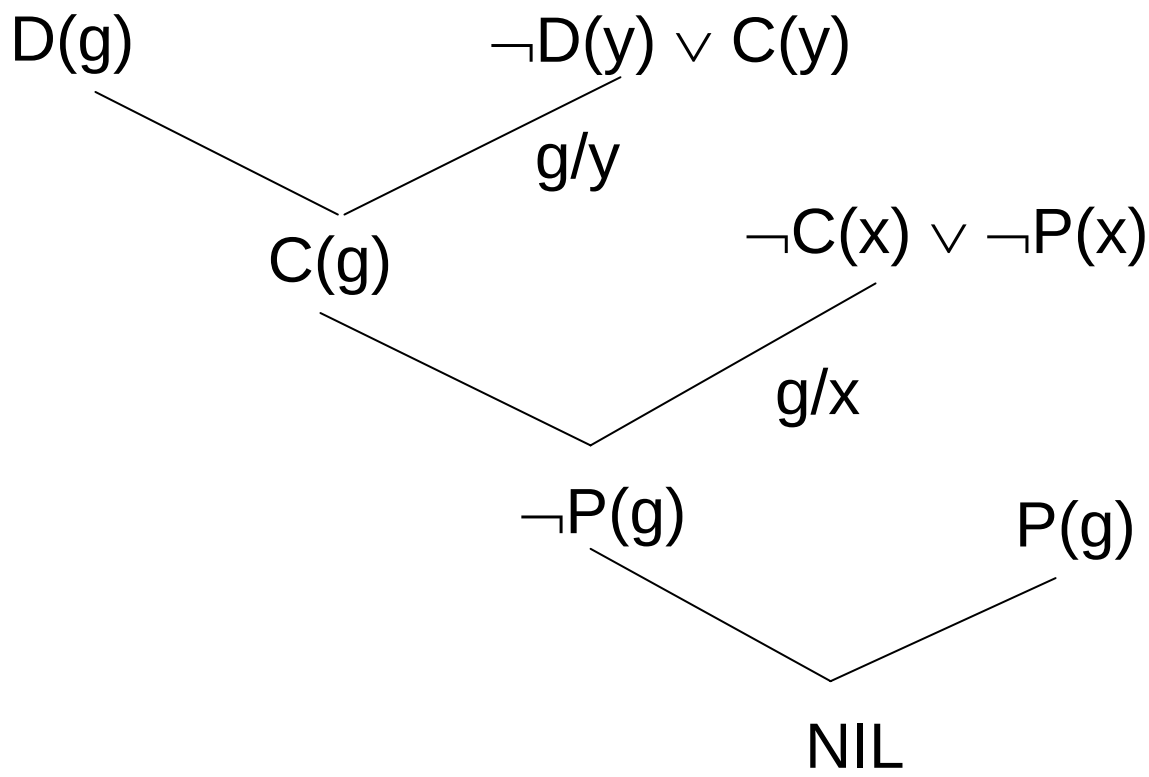
Lista delle clausole:

a) $\neg C(x) \vee \neg P(x)$

b) $\neg D(y) \vee C(y)$

c) $P(g)$

d) $D(g)$



LA NEGAZIONE

La frase *nessun cetaceo è un pesce* (cioè *nessun A è B*) può essere interpretata come:

a) $(\forall x)\neg(cetaceo(x) \wedge pesce(x))$

a parole: per tutti gli x, vale l'affermazione negata di essere cetaceo e insieme pesce.

Questa è del tutto equivalente a

$$\neg(\exists x)(cetaceo(x) \wedge pesce(x))$$

a parole: non esiste un x per cui vale essere cetaceo e insieme pesce.

(Notare infatti che: $\neg(\exists x)(...) \Rightarrow (\forall x)\neg(...)$)

Le due forme precedenti sono ancora equivalenti a $(\forall x)(\neg cetaceo(x) \vee \neg pesce(x))$

a parole: x o non è cetaceo o non è pesce.

b)

$$(\forall x)(cetaceo(x) \rightarrow \neg pesce(x))$$

a parole: per qualsiasi x, se x è cetaceo, allora non è pesce.

La forma equivalente

$$(\forall x)(\neg cetaceo(x) \vee \neg pesce(x))$$

coincide con quella di a)

c)

$$\neg(\exists x)(cetaceo(x) \rightarrow pesce(x))$$

a parole: non esiste nessun x per cui (se x è cetaceo, allora è pesce).

Trasformando la precedente si ha

$$(\forall x)\neg(cetaceo(x) \rightarrow pesce(x))$$

$$(\forall x)\neg(\neg cetaceo(x) \vee pesce(x))$$

$$(\forall x)(cetaceo(x) \wedge \neg pesce(x))$$

a parole: per tutti gli x, vale la relazione che x è cetaceo e insieme non pesce (dire che tutti gli x sono cetacei è esagerato).

Ora la formula

$$(\forall x)(\neg cetaceo(x) \vee \neg pesce(x))$$

è del tipo $\overline{A} + \overline{B} = \overline{A \cdot B}$ che ha la tavola di verità:

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	1	0
1	0	1

Invece la formula

$$(\forall x)(cetaceo(x) \wedge \neg pesce(x))$$

è del tipo $A \cdot \overline{B}$ che ha la tavola di verità:

A	B	$A \cdot \overline{B}$
0	0	0
0	1	0
1	1	0
1	0	1

che coincide con l'altra solo nelle ultime due righe: in generale, è più restrittiva.

Se, ad esempio, x=trota, dalla prima tavola di verità (seconda riga) si ottiene VERO (resta valida l'assunzione), dalla seconda si ottiene FALSO, che è assurdo.

Problema della scimmia e della banana

Molti problemi di pianificazione possono essere trattati mediante la logica.

Ad esempio, il problema della scimmia e della banana può essere formulato concependo gli operatori come funzioni che mandano uno stato in un altro stato.

Si consideri il problema semplificato adottando solo 3 operatori: grasp, climbbox e pushbox.

L'effetto degli operatori può essere così descritto:

1. Per tutti gli **x** ed s, se la scimmia non è sulla cassa nello stato s, la cassa si trova nella posizione **x** nello stato che si ottiene applicando l'operatore pushbox(**x**) allo stato s.

   In formula:

   $(\forall \mathbf{x}\ \forall s)\{\neg ONBOX(s) \rightarrow AT(box, \mathbf{x}, pushbox(\mathbf{x}, s))\}$

2. Per tutti gli s, la scimmia si trova sulla cassa nello stato che si ottiene applicando l'operatore climbox allo stato s.

   $(\forall s)\{ONBOX(climbbox(s))\}$

3. Per tutti gli s, se la scimmia è sulla cassa e la cassa è in **c** nello stato s, la scimmia prenderà le banane nello stato che si ottiene applicando l'operatore grasp allo stato s.

$$(\forall s)\{ONBOX(s) \wedge AT(box, \mathbf{c}, s) \rightarrow HB(grasp(s))\}$$

NB: HB è un predicato che è vero quando la scimmia ha preso le banane (hit bananas).

Oltre a questi assiomi, occorre enunciare esplicitamente altri effetti degli operatori (*frame problem*) come "la posizione della cassa non muta quando la scimmia sale sulla cassa" e "la scimmia non è ancora sulla cassa quando ha finito di spingerla", ecc.. Serve solo la prima:

4. La posizione della cassa non muta quando la scimmia sale sulla cassa.

   $(\forall \mathbf{x}\ \forall s)\{AT(box, \mathbf{x}, s) \rightarrow AT(box, \mathbf{x}, climbox(s))\}$

Infine serve descriverelo stato iniziale:

5. La scimmia non è sulla cassa in s_0.

   $\neg ONBOX(s_0)$

Si vuole dimostrare:
$(\exists s)HB(s)$
(esiste uno stato in cui è vera HB)

Passo 1) Si nega l'enunciato
$\neg(\exists s)HB(s)$

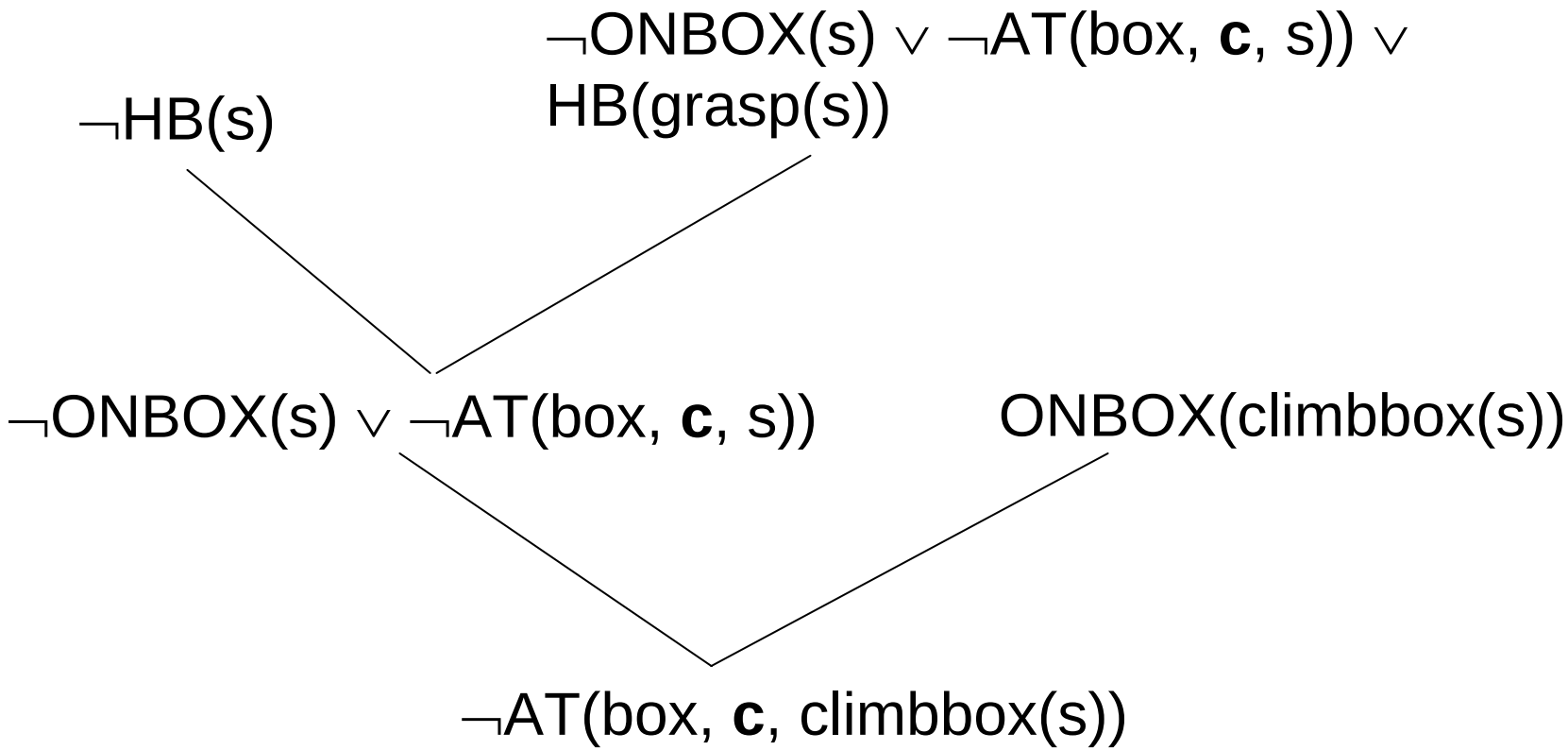
Passo 2) Si trasforma in clausola
$(\forall s)\neg HB(s)$ ovvero: $\neg HB(s)$

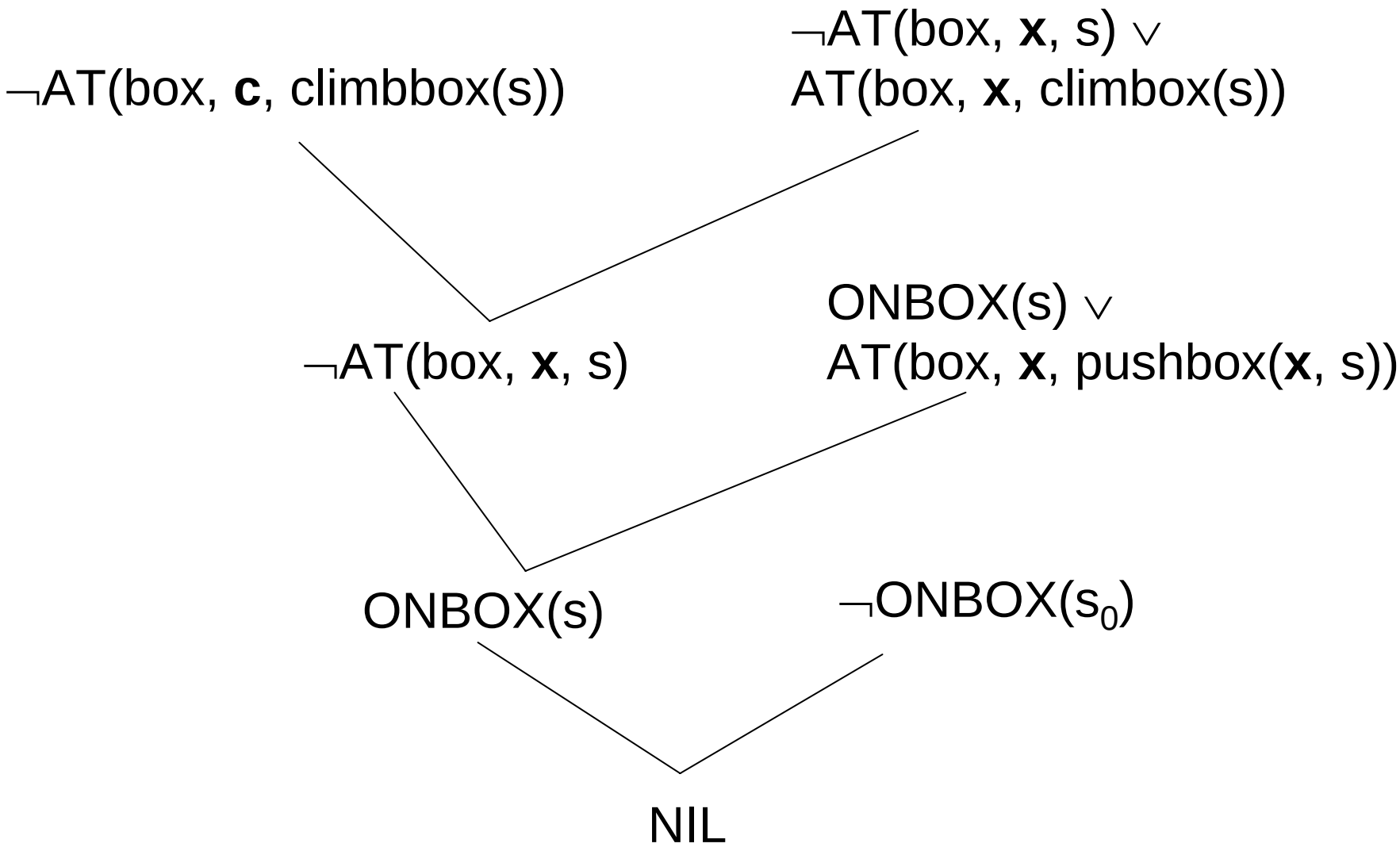
Passo 3) e successivi
Si inserisce nella base di conoscenza e si applica la dimostrazione.

La base di conoscenza, ridotta a clausole, è

1. $\text{ONBOX}(s) \vee \text{AT}(\text{box}, \mathbf{x}, \text{pushbox}(\mathbf{x}, s))$
2. $\text{ONBOX}(\text{climbbox}(s))$
3. $\neg(\text{ONBOX}(s) \wedge \text{AT}(\text{box}, \mathbf{c}, s)) \vee \text{HB}(\text{grasp}(s))$
   ovvero
   $\neg\text{ONBOX}(s) \vee \neg\text{AT}(\text{box}, \mathbf{c}, s)) \vee \text{HB}(\text{grasp}(s))$
4. $\neg\text{AT}(\text{box}, \mathbf{x}, s) \vee \text{AT}(\text{box}, \mathbf{x}, \text{climbox}(s))$
5. $\neg\text{ONBOX}(s_0)$

Dimostrazione:




¬AT(box, **c**, climbbox(s))
¬AT(box, **x**, s) ∨
AT(box, **x**, climbox(s))
¬AT(box, **x**, s)
ONBOX(s) ∨
AT(box, **x**, pushbox(**x**, s))
ONBOX(s)
¬ONBOX(s_0)
NIL

Nota: nella prima risolvente, si unifica con la sostituzione grasp(s)/s: questo è lecito perché non è la stessa s!

Idem per gli altri passi.