Risoluzione

Eric Miotto
Corretto dal prof. Silvio Valentini

15 giugno 2005

1 Risoluzione

Introdurremo ora un metodo per capire se un insieme di formule è soddisfacibile o meno. Lo vedremo prima per insiemi finiti di formule proposizionali, poi per insiemi infiniti di formule proposizionali ed infine con formule predicative.

Inizialmente, per semplificare la trattazione, non considereremo le formule in cui compaiano i quantificatori $\forall$ e $\exists$: questi sono spesso critici perché richiedono di analizzare tutte le interpretazioni possibili, cosa non fattibile negli insiemi infiniti. Ci limiteremo perciò a considerare formule in cui ci siano solo connettivi e a tratteremo quelle parti che presentano quantificatori come formule atomiche.

Definizione 1.1 *Una* formula prima *è una formula atomica o una formula che comincia con un quantificatore.*

Definizione 1.2 *Una* formula proposizionale *è una formula prima o è composta da formule proposizionali unite da connettivi.*

Definizione 1.3 *Una formula proposizionale che sia una verità logica per ogni interpretazione proposizionale è detta* tautologia.

Notiamo che, se da un lato semplifichiamo il lavoro da svolgere, dall'altro rischiamo di dare soluzioni sbagliate: una formula quantificata universalmente che noi supponiamo essere vera (in quanto formula prima) può rivelarsi in realtà falsa (per l'interpretazione effettiva).

Se Φ è finito e trattiamo solo formule prime, per risolvere una conseguenza logica mediante la verità logica associata possiamo usare agevolmente le tabelline di verità e verificare che in ogni caso l'interpretazione sia vera. Questo metodo funziona direttamente anche per la conseguenza logica, cercando prima le righe della tabella che rendono vere le formule di Φ e verificando se rendono vera anche la formula A. Considerando il nostro solito esempio, $A, A \to B \models B$, compileremo la seguente tavola

A	B	$A \to B$
$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\top$
$\top$	$\top$	$\top$

Consideriamo le righe che soddisfano A

A	B	$A \to B$
$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$

ed anche $A \to B$

A	B	$A \to B$
$\top$	$\top$	$\top$

e verifichiamo nelle righe rimaste che valga B. In questo caso possiamo dire che la conseguenza logica vale.

Ciò funziona solo se Φ è finito, ed in ogni caso le tabelline di verità sono pesanti dal punto di vista computazionale - per n formule è necessario compilare 2^n righe. Non possiamo quindi rimanere nella semantica: dobbiamo ritornare alla sintassi e cercare degli algoritmi che tramite manipolazioni di segni ci aiutino a risolvere i problemi di conseguenza logica, di verità logica e di soddisfacibilità (naturalmente sfruttiamo la correlazione tra questi ultimi).

Vediamo innanzitutto l'idea del metodo che vogliamo adottare. Supponiamo che il seguente insieme di formule sia soddisfacibile

$$S = \{\ldots, \neg A \vee B, \neg B \vee C, \ldots\}$$

che esista cioè una interpretazione che renda vere tutte le formule in S. Nella fattispecie anche $\neg A \vee B$ e $\neg B \vee C$ dovranno essere vere. Queste possono essere sostituite dalla formula $\neg A \vee C$. Se per assurdo $\neg A \vee C$ fosse falsa, sarebbero falsi anche $\neg A$ e C; quindi B dovrà essere vero per rendere vero $\neg A \vee B$, ma così facendo $\neg B$ è falso e $\neg B \vee C$ non può essere vera. Di conseguenza anche il seguente insieme

$$\{\ldots, \neg A \vee B, \neg B \vee C, \neg A \vee C, \ldots\}$$

sarà soddisfacibile. Se continuando questa semplificazione (anche riutilizzando le stesse formule) si ottiene una formula non soddisfacibile, come ad esempio $\neg\top \vee \bot$, l'insieme ottenuto sarà non soddisfacibile, così come i precedenti e in particolare quello di partenza. Notiamo come per l'applicazione del metodo sia fondamentale la forma della formule.

Formalizziamo quanto appena detto. Preso un insieme S di formule, scegliamo dentro S due formule aventi la forma

$$A_1 \vee A_2 \vee A_3 \vee A_4 \vee \cdots \vee A_n \tag{1}$$

$$B_1 \vee B_2 \vee B_3 \vee \cdots \vee B_m \tag{2}$$

Troviamo due formule prime tali che siano uno la negazione dell'altra (dal punto di vista sintattico), come C e $\neg C$, $\neg C$ e $\neg\neg C$, $\neg\neg C$ e $\neg\neg\neg C$, ecc..., e da queste ricaviamo una nuova formula ottenuta concatenando (1) e (2) con $\vee$ ed eliminando al contempo le formule prime individuate. Supponendo che nelle formule presentate sia $B_2 = \neg A_4$ otteniamo perciò

$$H = A_1 \vee A_2 \vee A_3 \vee \cdots \vee A_n \vee B_1 \vee B_3 \vee \cdots \vee B_m$$

Se S è soddisfacibile, allora è soddisfacibile anche H. Infatti sia (1) sia (2) sono vere: se per esempio A_4 è vero, per rendere vera (2) è necessario che un'altra formula sia vera, per cui in H sarà presente e la renderà vera. È un ragionamento analogo fatto nella presentazione dell'idea.

Possiamo anche dire che se S è soddisfacibile, allora anche $S \cup \{H\}$ è soddisfacibile. Possiamo operare similmente su quest'ultimo insieme, e continuare fino non a che non arriviamo ad un insieme S^* su cui non possiamo più operare. Se a questo punto scopriamo che S^* non è soddisfacibile, questo significa che a ritroso tutti gli insiemi precedenti erano non soddisfacibili e quindi anche l'insieme di partenza S non è soddisfacibile.

Definizione 1.4 *Un* letterale *è una formula prima o la negazione di una formula prima.*

Definizione 1.5 *Una* clausola A *è una formula che può essere definita come segue:*

- $A = \top$ *o* $A = \bot$*;*
- $A = L$*, con* L *letterale*
- $A = L_1 \vee \cdots \vee L_n$*, con* $L_1, \ldots, L_n$ *letterali.*

Supponiamo che S sia finito e che contenga solo clausole. Possiamo andare avanti all'infinito a semplificare? Ad un certo punto saremo costretti a scegliere le stesse coppie, che non ci porteranno più vantaggi, terminando la produzione di nuove formule. A questo punto possiamo trovare una contraddizione, per cui S è non soddisfacibile, o non trovarla: in quest'ultimo caso siamo sicuri che l'insieme è veramente soddisfacibile?

Prima di affrontare quest'ultima questione, facciamo vedere che se S contiene formule proposizionali di forma qualsiasi, queste possono essere espresse nella forma che vogliamo. Per dimostrarlo, introduciamo semplicemente un algoritmo di trasformazione:

1) trasformiamo le implicazioni. $A \to B$ diventa $\neg A \vee B$;

2) trasformiamo le congiunzioni a se stanti. $A \wedge B$ diventa A, B. Separiamo i componenti della formula e li inseriamo da soli: il requisito di soddisfacibilità dell'insieme impone che siano veri contemporaneamente come lo richiede $\wedge$;

3) cerchiamo di portare le negazioni accanto alle formule prime.

   - $\neg(A \wedge B)$ diventa $\neg A \vee \neg B$;
   - $\neg(A \vee B)$ si può scrivere $\neg A \wedge \neg B$, e quindi diventa $\neg A, \neg B$;
   - infine $\neg\neg A$ diventa A;

4) $(A \wedge B) \vee C$ diventa $A \vee C, B \vee C$;

5) se una clausola contiene $\top$, può essere completamente eliminata;

$$A_1 \vee \cdots \vee \top \vee \cdots \vee A_n$$

6) se una clausola contiene $\bot$, può essere semplificata. $A_1 \vee \cdots \vee \bot \vee \cdots A_n$ diventa $A_1 \vee \cdots \vee A_n$;

7) se una clausola contiene una formula prima e la sua negazione, può essere completamente eliminata;

$$A_1 \vee \cdots \vee A \vee \cdots \vee \neg A \vee \cdots \vee A_n$$

8) se una formula prima è ripetuta in una clausola, può essere mantenuta una sola occorrenza.

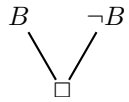
Riusciamo quindi a trasformare un insieme S composto da generiche formule in un insieme S' composto da sole clausole. A questo possiamo applicare la *risoluzione* a S', seguendo il seguente algoritmo:

```
repeat
  G:=S
  S:=Res(S)
until (□ ∈ S) or (G=S)
```

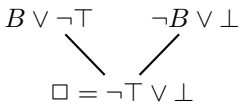
Commentiamo l'algoritmo. La funzione Res applica la risoluzione ad ogni coppia di clausole in S, creando nuove clausole. La risoluzione prende due clausole: nella prima è presente una certa formula prima, nella seconda è presente negata. Viene così creata una nuova clausola, detta *risolvente*, ottenuta unendo le due clausole con una disgiunzione ed eliminando la formula prima presente diritta e negata:

$$\begin{array}{ccc} A_1 \vee \cdots \vee B \vee \cdots \vee A_n & & C_1 \vee \cdots \vee \neg B \vee \cdots \vee C_m \\ & \searrow \quad \swarrow & \\ & A_1 \vee \cdots \vee A_n \vee C_1 \vee \cdots \vee C_m & \end{array}$$

L'algoritmo finisce o quando incontriamo la clausola vuota o quando non riusciamo più a creare nuove clausole (teniamo una copia dell'insieme prima della risoluzione per fare questo confronto). La *clausola vuota* si ottiene quando applichiamo la risoluzione ad una formula prima e alla sua negazione (che non possono essere vere contemporaneamente):



Formalmente si può scrivere:



- Esercizio: *stabilire se*

$$F = (\neg B \wedge \neg C \wedge D) \vee (\neg B \wedge \neg D) \vee (C \wedge D) \vee B$$

è una tautologia.

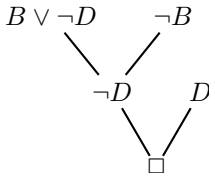
Dire che F è una tautologia equivale a dire che $\neg F$ non è soddisfacibile. Usiamo perciò la risoluzione. Prima trasformiamo l'insieme $\{\neg F\}$ in modo che contenga solo clausole:

$$\{\neg((\neg B \wedge \neg C \wedge D) \vee (\neg B \wedge \neg D) \vee (C \wedge D) \vee B)\}$$
$$\{\neg(\neg B \wedge \neg C \wedge D), \neg(\neg B \wedge \neg D), \neg(C \wedge D), \neg B\}$$
$$\{B \vee C \vee \neg D, B \vee D, \neg C \vee \neg D, \neg B\}$$

Ora possiamo procedere con l'algoritmo di risoluzione. Eseguiamo la prima iterazione, applicando la risoluzione a tutte le coppie:

$$\{B \vee C \vee \neg D, B \vee D, \neg C \vee \neg D, \neg B, B \vee C \vee B, B \vee \neg D \vee \neg D, C \vee \neg D, B \vee \neg C, D\}$$
$$\{B \vee C \vee \neg D, B \vee D, \neg C \vee \neg D, \neg B, B \vee C, B \vee \neg D, C \vee \neg D, B \vee \neg C, D\}$$

A questo punto intravediamo una possibile contraddizione. Evitando di rifare tutte le coppie si ottiene il seguente albero:



L'ultimo insieme è non soddisfacibile; a ritroso, anche i precedenti lo sono e in particolar modo $\{\neg F\}$; per cui F è una tautologia.

Dobbiamo ora dimostrare che questo metodo è corretto e sufficiente. Il metodo è corretto perché le operazioni che operiamo sull'insieme di partenza (trasformazioni delle formule in clausole e risoluzione) mantengono la soddisfacibilità, per cui se troviamo la clausola vuota alla fine significa che anche tutti gli altri insiemi sono non soddisfacibili.

Un po' più laborioso è far vedere che il metodo è sufficiente (o completo), ci da' cioè sempre la risposta giusta, per cui se l'insieme di partenza è non soddisfacibile otteniamo sempre la clausola vuota, non esistono insiemi non soddisfacibili per i quali non la si ottiene (condizione sufficiente). La dimostrazione si fa per induzione sul numero n di formule prime che compaiono nelle formule che costituiscono l'insieme in analisi:

- (base) Supponiamo che l'insieme abbia 0 formule. Le uniche clausole ammesse saranno $\bot$, $\top$, $\neg\bot$, $\neg\top$. Appare evidente che devono apparire o $\bot$ o $\neg\bot$ o entrambe nell'insieme affinché l'insieme sia non soddisfacibile. Siccome la clausola vuota è per definizione $\bot$ (indica che abbiamo una contraddizione), la sufficienza è dimostrata.

- (passo) Supponiamo per ipotesi induttiva che se ad un insieme non soddisfacibile le cui formule sono scritte utilizzando n formule prime applichiamo la risoluzione otteniamo la clausola vuota. Dobbiamo dimostrare che questo vale anche per un insieme non soddisfacibile le cui formule prime sono scritte utilizzando $n+1$ formule prime.

  Siano $A_1, \ldots, A_n, A_{n+1}$ le formule prime di un insieme non soddisfacibile S. Possiamo avere sostanzialmente tre tipi di clausole:

  1) clausole in cui appare A_{n+1} non negato, per esempio $A_1 \vee A_5 \vee \neg A_3 \vee \cdots \vee A_{n+1}$;
  2) clausole in cui appare A_{n+1} negato, per esempio $A_2 \vee \neg A_{n+1}$;
  3) clausole in cui non appare A_{n+1}, per esempio $A_4 \vee A_8$.

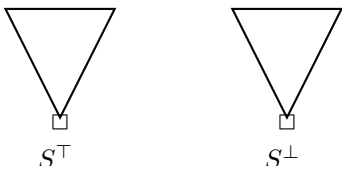
  Per sfruttare l'ipotesi induttiva, simuliamo di effettuare un'interpretazione con la quale interpretiamo A_{n+1}:

  – $I^\sigma(A_{n+1})$ =vero. Possiamo eliminare le clausole in cui compare A_{n+1} non negato (sono sempre vere) e togliere $\neg A_{n+1}$ dalle clausole in cui compare (non influisce), ottenendo così un insieme $S^\top$;
  – $I^\sigma(A_{n+1})$ =falso. Possiamo eliminare le clausole in cui compare $\neg A_{n+1}$ e togliere A_{n+1} da quelle in cui appare, ottenendo così un insieme $S^\bot$.

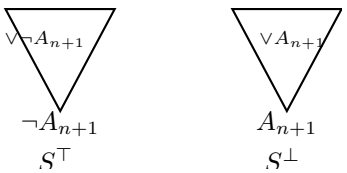
  Anche $S^\top$ e $S^\bot$ sono non soddisfacibili, infatti:

  – se $S^\top$ fosse soddisfacibile, esiste un'interpretazione in cui le sue clausole sono tutte vere, ma di conseguenza sarebbero vere anche quelle di S (per i criteri con cui abbiamo ottenuto $S^\top$ interpretando A_{n+1} in $\top$), per cui S sarebbe soddisfacibile, contro l'ipotesi fatta;
  – analogo discorso duale per $S^\bot$.

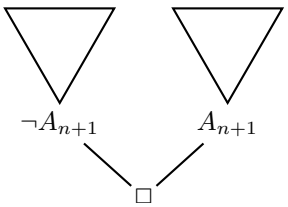
  Per ipotesi induttiva se applichiamo la risoluzione a $S^\top$ e $S^\bot$ otteniamo la clausola nulla, dato che le loro clausole sono scritte utilizzando n formule prime. Abbiamo perciò due alberi di risoluzione:



  Immaginiamo ora di sostituire alle formule degli alberi quelle corrispondenti in S: si tratta sostanzialmente di aggiungere A_{n+1} e $\neg A_{n+1}$ dove e se necessario:



  In questo modo al posto di ottenere la clausola nulla otteniamo A_{n+1} per $S^\bot$ e $\neg A_{n+1}$ per $S^\top$, dato che non si risolvono con niente. Siccome le clausole di questi alberi appartengono a S, possiamo considerare questi alberi come fossero fatti in S e applicare la risoluzione a $\neg A_{n+1}$ e A_{n+1}:



Abbiamo così ottenuto la clausola nulla per S insoddisfacibile: la sufficienza è dimostrata anche per il passo induttivo.

Naturalmente, se negli alberi di risoluzione non sono state usate formule che in origine contenevano A_{n+1}, gli alberi ottenuti sono validi direttamente per S senza un passo ulteriore di risoluzione.

- Esercizio: *dire se vale la seguente conseguenza logica:*

$$\neg A \vee B, \neg B \vee C, A \vee \neg C, A \vee B \vee C \models A \wedge B \wedge C$$

Trasformiamolo in un problema di soddisfacibilità ed adoperiamo la risoluzione:

$$\{\neg A \vee B, \neg B \vee C, A \vee \neg C, A \vee B \vee C, \neg(A \wedge B \wedge C)\}$$

Rendiamo l'ultima formula una clausola:

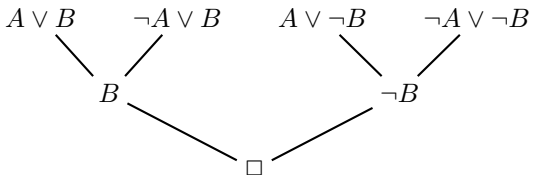
$$\{\neg A \vee B, \neg B \vee C, A \vee \neg C, A \vee B \vee C, \neg A \vee \neg B \vee \neg C\}$$

Applichiamo la risoluzione:

$$\begin{aligned}\{&\neg A \vee B, \neg B \vee C, A \vee \neg C, A \vee B \vee C, \neg A \vee \neg B \vee \neg C,\\ &\neg A \vee C, B \vee \neg C, B \vee B \vee C, \neg A \vee \neg A \vee \neg C,\\ &\neg B \vee A, C \vee A \vee C, \neg B \vee \neg A \vee \neg B,\\ &A \vee A \vee B, \neg C \vee \neg B \vee \neg C,\\ &B \vee C \vee \neg B \vee \neg C, A \vee C \vee \neg A \vee \neg C, A \vee B \vee \neg A \vee \neg B\}\end{aligned}$$

$$\begin{aligned}\{&\neg A \vee B, \neg B \vee C, A \vee \neg C, A \vee B \vee C, \neg A \vee \neg B \vee \neg C,\\ &\neg A \vee C, B \vee \neg C, B \vee C, \neg A \vee \neg C,\\ &\neg B \vee A, A \vee C, \neg A \vee \neg B,\\ &A \vee B, \neg B \vee \neg C\}\end{aligned}$$

A questo punto si può ottenere il seguente albero:



L'insieme è insoddisfacibile, per cui la conseguenza logica data è valida.

Vogliamo ora capire quanto lavoro dobbiamo fare per ottenere per ottenere la clausola vuota, qual è il suo costo computazionale. La dimostrazione di completezza appena svolta ci ha fatto ottenere un metodo alternativo di applicazione della risoluzione, che opera per ricorsione sul numero n di formule prime:

- se $n = 0$, dobbiamo semplicemente vedere se tra la clausole c'è $\perp$;

- se $n = 1$, nel caso peggiore otteniamo l'insieme $S_1 = \{A_1, \neg A_1, \square\}$. Simulando le interpretazioni per A_1, si ottengono $S_1^\top = \{\square\}$ e $S_1^\bot = \{\square\}$: abbiamo da fare due volte il lavoro su insiemi con 0 formule prime e operare la risoluzione sui risultati ottenuti.
- per $n = 2$ possiamo ottenere nel peggiore dei casi $S_2 = \{A_1, A_2, \neg A_1, \neg A_2, A_1 \vee A_2, \neg A_1 \vee A_2, A_1 \vee \neg A_2, \neg A_1 \vee \neg A_2, \square\}$. Simulando le interpretazioni per A_2 si ottiene $S_2^\top = \{A_1, \neg A_1, \square\} = S_1$ e $S_2^\bot = \{A_1, \neg A_1, \square\} = S_1$, per cui dovremmo fare due volte il lavoro su insiemi con 1 formula prime e operare una risoluzione.

Si capisce quindi che se abbiamo n formule prime in un insieme, dovremmo svolgere 2^n risoluzioni, confrontabile con il lavoro da svolgere se usassimo le tabelline. In conclusione, la risoluzione è un algoritmo inefficiente, ma comunque non peggiore delle tabelline di verità!

1.1 Clausole di Horn

Il problema di complessità computazionale appena visto non si può risolvere alla radice, ma è possibile aggirarlo se operiamo solo con alcuni tipi di clausole.

Definizione 1.6 *Una* clausola di Horn *è una clausola che contiene al più un letterale positivo (diritto):*

$$\neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_n \vee A$$

Equivalentemente, una clausola di Horn è una clausola ottenuta dalla trasformazione di una implicazione della forma

$$B_1 \wedge B_2 \wedge \cdots \wedge B_n \rightarrow A$$

Possiamo avere due tipi particolari di clausola di Horn:

- clausola di Horn in cui $A \equiv \bot$:

  $$B_1 \wedge B_2 \wedge \cdots \wedge B_t \rightarrow \bot$$

  o più semplicemente

  $$B_1 \wedge B_2 \wedge \cdots \wedge B_t \rightarrow$$

- clausola di Horn in cui prima di $\rightarrow$ ci sia solo $\top$:

  $$\top \rightarrow A$$

  o più semplicimente

  $$\rightarrow A$$

Questo tipo di clausola è adottato anche nel linguaggio dichiarativo Prolog (proprio per ragioni di efficienza), nel quale vanno specificate al contrario e la congiunzione è sostituita dalla virgola (e usualmente $\leftarrow$ da :-):

$$A \leftarrow B_1, B_2, \ldots, B_n$$

Adotteremo anche noi d'ora in poi questa notazione.
Abbiamo quindi tre tipi di clausole:

1) clausole che asseriscono dei fatti: $A \leftarrow$ (si legge "A asserito");
2) clausole che ci dicono che delle formule sono in contraddizione (le loro interpretazioni sono in contraddizione): $\leftarrow B_1, \ldots, B_n$;

3) clausole che ci dicono che se sono vere delle formule è vera anche un'altra: $A \leftarrow B_1, \ldots, B_n$.

L'algoritmo per capire se un insieme di clausole di Horn è non soddisfacibile è composto dai seguenti passi:

1) si stabilisce che le formule prime presenti in clausole di tipo 1 sono vere;

2) si stabiliscono vere le formule prima a sinistra di $\leftarrow$ nelle clausole di tipo 3 se tutte quelle a destra risultano vere per il passo precedente o per quello attuale;

3) se si riesce a rendere vere tutte le formule in una clausola di tipo 2, abbiamo dimostrato la non soddisfacibilità.

Se percorso all'incontrario, questo metodo di risoluzione presenta un approccio precedurale. Si parte dalle clausole di tipo 2 e si vede se è possibile rendere vere ogni formula prima presente. Si vede perciò se per ognuna di queste esiste una clausola di tipo 1 o 3 che inizia proprio con quel letterale: se è una clausola di tipo 1 sappiamo che è un fatto e torniamo alla clausola di partenza, se è una clausola di tipo 3 vediamo se riusciamo a renderla vera rendendo veri i letterali dopo il $\leftarrow$.

Nell'applicazione dell'algoritmo si può applicare una metodologia top-down (dalle contraddizioni ai fatti) o bottom-up (dai fatti alle contraddizioni).

- Esercizio: *determinare se il seguente insieme è non soddisfabile:*

$$\{(\neg A \vee \neg B \vee \neg D) \wedge \neg E \wedge (\neg C \vee A) \wedge C \wedge B \wedge (\neg G \vee D) \wedge G\}$$

Con un semplice passo di trasformazione

$$\{\neg A \vee \neg B \vee \neg D, \neg E, (\neg C \vee A), C, B, \neg G \vee D, G\}$$

riusciamo ad ottenere un insieme composto da sole clausole di Horn:

$$\{\leftarrow A, B, D; \leftarrow E; A \leftarrow C; C \leftarrow; B \leftarrow; D \leftarrow G; G \leftarrow\}$$

Proviamo perciò a risolvere a partire dalla prima clausola:

$$\begin{gathered} \leftarrow A, B, D \\ \leftarrow C, B, D \\ \leftarrow B, D \\ \leftarrow D \\ \leftarrow G \\ \square \end{gathered}$$

L'insieme è risultato insoddisfacibile, e lo è anche quello dato.

Dobbiamo ora dimostrare che anche questo metodo sia corretto e completo. Il metodo è corretto perché usiamo un'interpretazione: interpretiamo in vero quelle che raggiungiamo dalle clausole di tipo 2 e in falso quelle che non raggiungiamo.

La dimostrazione di completezza è analoga alla dimostrazione di completezza della risoluzione. Facciamo vedere che se un insieme S composto da clausole di Horn è insoddisfacibile allora otteniamo la clausola vuota $\square$. La dimostrazione procede naturalmente per induzione sul numero n di letterali distinti presenti:

- (base) se $n = 0$, potranno essere presenti le clausole $\top$, $\bot$, $\neg\top$ e $\neg\bot$. Siccome S è insoddisfacibile dovrà essere presente $\bot$, per cui abbiamo anche la clausola vuota;

- (passo) supponiamo che da un insieme insoddisfacibile avente clausole scritte con n formule prime otteniamo la clausola vuota: dimostriamo che ciò vale anche per insiemi insoddisfacibili aventi clausole scritte con $n+1$ formule prime $A_1, \ldots, A_n, A_{n+1}$. Rispetto a A_{n+1} possiamo individuare tre tipi di clausole:

  1) clausole che la contengono diritta, come $A_{n+1} \leftarrow A_1, A_2, A_4$ e $A_{n+1} \leftarrow$;
  2) clausole che la contengono negata, come $A_5 \leftarrow A_{n+1}, A_3$ e $\leftarrow A_{n+1}, A_5$;
  3) clausole che non la contengono.

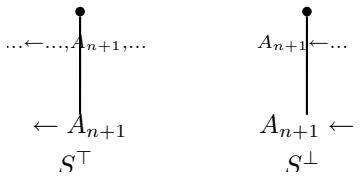
  Simuliamo allora una interpretazione con la quale interpretiamo A_{n+1}:

  - $I^\sigma(A_{n+1})$ =vero. Possiamo eliminare le clausole che contengono A_{n+1} diritta (a sinistra di $\leftarrow$) ed eliminare le occorrenze di A_{n+1} che compaiono a destra di A_{n+1}. In pratica, cancelliamo le clausole che ci dicono per cosa A_{n+1} è vera;
  - $I^\sigma(A_{n+1})$ =falso. Eliminiamo le clausole che contengono A_{n+1} a destra di $\leftarrow$ e eliminiamo le occorrenze di A_{n+1} a sinistra di $\leftarrow$. In pratica cancelliamo le clausole nella quali A_{n+1} rende vero qualcosa.

  Otteniamo rispettivamente due insiemi $S^\top$ e $S^\perp$, che saranno insoddisfacibili (se fossero soddisfacibili, per il procedimento seguito lo sarebbe anche S). Applicando a questi due insiemi il metodo, otteniamo alla fine la clausola vuota da entrambi (per ipotesi induttiva):

  Ora sostituiamo alle clausole utilizzate le loro corrispondenti in S. In $S^\top$ aggiungeremo A_{n+1} a destra di $\leftarrow$: ritroveremo così al posto della clausola vuota $\leftarrow A_{n+1}$, non avendo clausole che ci dicono come possiamo ottenerla. In $S^\perp$ aggiungeremo A_{n+1} a destra di $\leftarrow$: al posto della clausola vuota ci sarà $A_{n+1} \leftarrow$, non avendo clausole in cui utilizzarlo:

  I procedimenti fatti possono essere considerati fatti in S, per cui possiamo ancora fare un passo

  $$\leftarrow A_{n+1}$$
  $$\square$$

  che ci porta alla clausola vuota.

- Esercizio: *si sa che possono verificarsi le seguenti reazioni chimiche:*

  $$\mathsf{MgO{+}H_2 \rightarrow Mg{+}H_2O}$$

  $$\mathsf{C{+}O_2 \rightarrow CO_2}$$

$$H_2O + CO_2 \rightarrow H_2CO_3$$

Supponendo di disporre in un laboratorio chimico di MgO, H_2, O_2 e C, dimostrare che è possibile avere H_2CO_3.

Questo problema può essere ricondotto alla dimostrazione di insoddisfacibilità del seguente insieme che contiene clausole di Horn:

$$\{\leftarrow H_2CO_3; H_2CO_3 \leftarrow H_2O, CO_2; CO_2 \leftarrow C, O_2; Mg, H_2O \leftarrow MgO, H_2; \\ C \leftarrow; O_2 \leftarrow; MgO \leftarrow; H_2 \leftarrow\}$$

Prima di procedere con la risoluzione, notiamo che dalla clausola Mg,$H_2O \leftarrow$ MgO, H_2 possiamo ricavare le clausole

$$H_2O \leftarrow MgO,\ H_2$$
$$Mg \leftarrow MgO,\ H_2$$

Naturalmente butteremo via uno dei prodotti ottenuti. Si ha perciò

$$\leftarrow H_2CO_3$$
$$\leftarrow H_2O,\ CO_2$$
$$\leftarrow H_2O,\ C,\ O_2$$
$$\leftarrow H_2O$$
$$\leftarrow MgO,\ H_2$$
$$\square$$

L'insieme è insoddisfacibile: possiamo perciò ottenere H_2CO_3 con quello che abbiamo in laboratorio.

Dal punto di vista logico abbiamo sfruttato il fatto che un insieme $S = \{\ldots, A_1 \vee A_2 \leftarrow B \wedge C, \ldots\}$ è soddisfacibile se e solo se lo sono i due insiemi $S_1 = \{\ldots, A_1 \leftarrow B \wedge C, \ldots\}$ e $S_2 = \{\ldots, A_2 \leftarrow B \wedge C, \ldots\}$

- Esercizio: il linguaggio Prolog è adatto per costruire *sistemi esperti*, che da un punto di vista logico sono oggetti che operano deduzioni in base alle informazioni a loro fornite. Supponiamo di avere i seguenti fatti e regole:

$$A \equiv \text{avere le ali}$$
$$M \equiv \text{avere il motore}$$
$$B \equiv \text{avere il becco}$$
$$P \equiv \text{avere le penne}$$
$$Met \equiv \text{essere di metallo}$$
$$Aereo \leftarrow B, Met$$
$$Uccello \leftarrow A, B, P$$
$$Met \leftarrow M$$

In questo modo abbiamo un ipotetico sistema esperto che distingue gli aerei dagli uccelli. Se abbiamo le seguente clausole

$$\{M \leftarrow, A \leftarrow, \leftarrow Aereo\}$$

riusciamo a far vedere che l'insieme è insoddisfacibile, cioè che stiamo descrivendo un aereo? Proviamoci

$$\leftarrow Aereo$$
$$\leftarrow A, Met$$
$$\leftarrow A, M$$
$$\square$$