Risoluzione di insiemi infiniti di formule e di insiemi con formule predicative

Eric Miotto

15 giugno 2005

Attenzione! Questi appunti non sono stati rivisti completamente dal professor Valentini, per cui saranno *sicuramente* presenti numerosi errori, in particolar modo nella dimostrazione del teorema di compattezza. Si spera comunque che siano utili lo stesso per l'apprendimento degli argomenti presentati.

1 Affrontare insiemi infiniti di formule

Abbiamo visto in dettaglio come lavorare con insiemi finiti di formule proposizionali. Facciamo ora un passo in avanti e passiamo a trattare insiemi infiniti di formule proposizionali. Ciò può apparire di nessuna utilità pratica, ma in realtà ci sono almeno due ragioni per cui ci serve:

- i quantificatori verranno approssimati come infinite formule proposizionali: $\forall$ può essere visto come una grande congiuzione tra infinite formule, mentre $\exists$ come una grande disgiunzione tra infinite formule. Questa approssimazione non sarà più possibile quando dovremo affrontare sistemi non numerabili, come i numeri reali: i nomi componibili con un numero finito di caratteri sono numerabili e non ci bastano per indicare tutti gli oggetti delle strutture non numerabili.
- abbiamo spesso a che fare con insiemi infiniti di formule. Pensiamo per esempio all'induzione: possiamo esprimerla come segue

  $$(A[x := 0] \wedge (\forall y(A[x := y] \rightarrow A[x =: y + 1]))) \rightarrow \forall x.A$$

  Sembra una sola formula, ma in realtà sono infinite. Se potessimo quantificare sulle formule potremmo scrivere tutte queste formule con una sola nel modo che segue

  $$\text{per ogni formula} A \ ((A[x := 0] \wedge (\forall y(A[x := y] \rightarrow A[x =: y + 1]))) \rightarrow \forall x.A)$$

  È evidente perciò che insiemi con infinite formule nascano in modo naturale. Introduciamo ora un importantissimo teorema che ci aiuterà.

Teorema 1.1 (Teorema di compattezza) *Sia Φ un insieme di formule. Allora Φ è soddisfacibile se e solo se ogni sottoinsieme finito Φ_0 di Φ è soddisfacibile. In simboli:*

$$\Phi \textit{ soddisfacibile} \iff \textit{per ogni } \Phi_0 \subseteq_\omega \Phi, \quad \Phi_0 \textit{ soddisfacibile}$$

Prima di dimostrarlo, cerchiamo di capire in che modo lo utilizzeremo. Il teorema ci dice anche quando un insieme di formule è insoddisfacibile:

$$\Phi \text{ insoddisfacibile} \iff \text{esiste } \Phi_0 \subseteq_\omega \Phi, \quad \Phi_0 \text{ insoddisfacibile}$$

Vediamo come questa nuova affermazione ci sia più utile nel determinare le proprietà di una struttura:

$$\begin{aligned}\Phi \models A \text{ sse } & \Phi \cup \{\neg A\} \text{ non soddisfacibile} \\ & \text{sse } \Phi_0 \cup \{\neg A\} \text{ non soddisfacibile} \quad \text{per qualche } \Phi_0 \subseteq_\omega \Phi \\ & \text{sse } \Phi_0 \models A\end{aligned}$$

Per scoprire se una certa affermazione vale su una struttura descritta da infinite proprietà, possiamo farlo partendo da un sottoinsieme finito delle sue proprietà. Il teorema di compattezza non ci dice però quale sia il sottoinsieme da usare. Ci da' in compenso una strategia di lavoro: per scoprire se un insieme è insoddisfacibile, ci basterà comporre man mano tutti i suoi sottoinsiemi finiti e scoprirne uno di insoddisfacibile. L'unica incognita è il tempo in cui otterremo la risposta, che potrebbe anche essere infinito nel caso l'insieme di partenza sia davvero soddisfacibile.

Dimostrazione del teorema di compattezza Nella dimostrazione del teorema, adotteremo le seguenti ipotesi semplificative:

- Φ sarà un insieme di (infinite) formule proposizionali. Il teorema funziona anche con formule predicative, perché sappiamo che il caso predicativo può essere approssimato con infinite formule proposizionali;
- supporremo di avere a che fare con una struttura numerabile, per non avere problemi con i nomi. In ogni caso i nomi che è possibile ottenere nel linguaggio sono una quantità numerabile.

In questo modo non ci complichiamo la vita e riusciamo a focalizzarci ugualmente sulle cose che ci interessano.

Il verso da sinistra a destra del teorema è quasi ovvio. Sappiamo che abbiamo un'interpretazione che rende vere tutte le formule di Φ; in particolare renderà vere le formule di ogni sottoinsieme finito di Φ.

Più complicato e laborioso è il verso opposto. Sappiamo infatti che per ogni sottoinsieme Φ' finito esiste un'interpretazione I' tale che rende vere tutte le formule. È evidente che una formula compaia in più sottoinsiemi, ed in questi sottoinsiemi può essere interpretata in maniera differente. Se I' e I'' sono due interpretazioni rispettivamente per Φ' e Φ'', in generale sarà $I' \neq I''$ (rispetto alle formule comuni). Non è dunque possibile fondere le possibili interpretazioni in una unica valida per Φ, dato che ci saranno sicuramente delle contraddizioni che a priori non riusciremmo a eliminare.

L'idea è quella di costruire un'interpretazione che vada bene per la maggior parte dei sottoinsiemi. Siano $A_1, A_2, \ldots, A_n, \ldots$ le formule prime usate. Consideriamo a tal scopo i sottoinsiemi di Φ le cui formule sono fatte con le prime n formule prime:

$$\begin{aligned}\Phi_0 &\equiv \text{insieme fatto con 0 formule prime} \\ \Phi_1 &\equiv \text{insieme fatto con } A_1 \\ \Phi_2 &\equiv \text{insieme fatto con } A_1, A_2 \\ &\ldots \\ \Phi_n &\equiv \text{insieme fatto con } A_1, \ldots, A_n \\ &\ldots\end{aligned}$$

Siccome Φ è infinito, sono infiniti anche i vari sottoinsiemi Φ_i: possiamo fare infatti formule sempre più lunghe anche in modo stupido (e perciò semplificabili).

D'altronde, in Φ_n le possibili combinazioni di valori di verità che possono assumere $A_1, \ldots, A_n$ sono 2^n. Una formula F, in corrispondenza di queste combinazioni, può assumere i due consueti valori di verità, per cui potremmo avere al massimo 2^{2^n} funzioni diverse, 2^{2^n} tabelline di verità possibili. Per quante siano le formule dentro Φ_n, quindi, al massimo possiamo dar luogo ad una quantità finita di tabelline di verità.

Per il modo in cui possono essere costruite le formule (si tenderà a ripetere le stesse in forma diversa), ci basterà dunque soddisfare un sottoinsieme di Φ_n per soddisfare automaticamente tutte le formule di Φ_n. Il numero di formule contenute nel sottoinsieme è finito, dato che potrà essere al massimo pari al numero di possibili funzioni di verità possibili, ovvero 2^{2^n}. Il sottoinsieme individuato, essendo finito, è anche soddisfacibile per ipotesi, essendo anche sottoinsieme di Φ.

All'aumentare delle formule prime considerate, otteniamo perciò delle interpretazioni che sono delle approssimazioni per l'interpretazione di Φ. L'unico rischio che ancora sussiste è che all'aumentare di n l'interpretazione cambi per le formule prime già considerate: dobbiamo riuscire a far "convergere" l'interpretazione. Sfruttiamo allora il seguente algoritmo per ottenere l'interpretazione di Φ:

> *Passo 0* All'inizio, l'interpretazione I di Φ è vuota, mentre vengono considerati tutti i sottoinsiemi significativi individuati, inseriti nell'insieme H.
>
> *Passo* $n+1$ Se ci sono infiniti indici in H tali che $I_i(A_{n+1}) = \top$ allora
>
> $$I = I \cup \{(A_n + 1, \top)\}$$
> $$H := H - \{i | I_i(A_{n+1}) = \bot\}$$
>
> In altre parole, prendiamo per buona l'interpretazione data dalla maggioranza e scartiamo dall'insieme H quei sottoinsiemi che hanno interpretazione opposta.
>
> Se invece ci sono infiniti indici in H tali che $I_i(A_{n+1}) = \bot$ lavoriamo in modo speculare:
>
> $$I = I \cup \{(A_n + 1, \bot)\}$$
> $$H := H - \{i | I_i(A_{n+1}) = \top\}$$
>
> Notiamo che in questo modo abbiamo sempre un numero infinito di insiemi in H.

2 Limiti espressivi del linguaggio

Il teorema di compattezza ci aiuta anche a capire cosa non si può fare con il linguaggio che abbiamo adottato.

Riprendiamo in mano uno dei problemi che ci eravamo prefissati di risolvere: la descrizione di una struttura tramite le sue proprietà. Volevamo fare in modo che, date tutte le proprietà di una struttura, questa fosse univocamente definita e si potesse scoprire la verità della affermazione tramite l'utilizzo della conseguenza logica. Prendiamo come esempio la struttura dei numeri naturali: per parlarne dovremmo evidenziare una struttura adeguata che contenga le relazioni, le funzioni e le costanti che ci interessano, poi prendere un linguaggio e scrivere con questo le proprietà dei numeri naturali. Avevamo già visto che questo era un problema. Supponiamo ottimisticamente però di sapere in qualche modo le proprietà che descrivono nel modo migliore i numeri naturali:

$$\mathrm{Th}(\mathcal{N}) = \{\varphi \mid \models_{\mathcal{N}} \varphi\}$$

Questa costituirà la nostra teoria di $\mathcal{N}$: con questa identifichiamo la struttura dei numeri naturali. Consideriamo ora il seguente insieme di formule

$$\mathrm{Th}(\mathcal{N}) \cup \{x \neq 0, x \neq s(0), x \neq s^2(0), \ldots, x \neq s^n(0), \ldots\}$$

È un insieme soddisfacibile? La risposta è difficile, tenendo conto anche che dobbiamo dare un'interpretazione per la variabile x. Possiamo usare però il teorema di compattezza. Consideriamo un generico sottoinsieme finito: questo sarà composto da parte di $\mathrm{Th}(\mathcal{N})$ e da alcune formule del secondo insieme. Questo sarà

sicuramente soddisfatto nella struttura dei numeri naturali e interpretando x nel numero naturale più grande di un'unità rispetto alle formule di disuguaglianza considerate. Per il teorema di compattezza, anche l'intero insieme è soddisfacibile, ma non in $\mathcal{N}$: ragionevolmente sarà $\mathcal{N}$ con un elemento aggiuntivo.

Anche con la migliore teoria possibile sui numeri naturali, non siamo riusciti ad individuarlo univocamente nell'esempio fatto, perché abbiamo fatto uso di un'altra struttura. Non riusciamo perciò a parlare esattamente di una struttura, non riusciamo a caratterizzarle con le formule. Tutto ciò dipende dalla scelta del linguaggio: se vale la compattezza, riusciamo a domare la conseguenza logica e di conseguenza a scrivere algoritmi (per quanto non totalmente corretti), ma non riusciamo a descrivere le strutture; se non vale la compattezza, riusciamo a descrivere le strutture ma non a scrivere algoritmi. Nella scelta di un linguaggio dobbiamo dunque scegliere tra una maggiore espressività, per descrivere le strutture, e la possibilità di esprimere algoritmi.

Passiamo ora ad un altro tipo di problema. Proviamo ad esprimere il fatto che un insieme è finito. Per dire che un insieme contiene esattamente n elementi, dobbiamo dire che ne contiene almeno n e non ne contiene più di n:

$$\begin{aligned}\text{Almeno-n} \equiv \exists x_1 \dots \exists x_n\ & x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \dots \wedge x_1 \neq x_n \wedge \\ & x_2 \neq x_3 \wedge \dots \wedge x_2 \neq x_n \wedge \\ & \dots \\ & \dots \wedge x_{n-1} \neq x_n\end{aligned}$$

$$\begin{aligned}\text{Al-più-n} \equiv \forall x_1 \dots \forall x_n \forall x_{n+1}\ & x_1 = x_2 \vee x_1 = x_2 \vee \dots \vee x_1 = x_{n+1} \vee \\ & x_2 = x_3 \vee \dots \vee x_2 = x_{n+1} \\ & \dots \\ & \dots \vee x_n = x_{n+1}\end{aligned}$$

$$\text{Finito-n} \equiv \text{Almeno-n} \wedge \text{Al-più-n}$$

Notiamo che queste formule non valgono per n generico, ma ad n deve essere assegnato un preciso valore. In tale ottica, i punti messi sono eliminabili.

La formule Finito sarebbe perciò

$$\text{Finito} \equiv \text{Finito-1} \vee \text{Finito-2} \vee \dots \vee \text{Finito-n} \vee \dots$$

Ma questa non è una formula, perché è infinita e dovrebbe contemplare gli infiniti casi possibili. Anche la soluzione

$$\text{Finito} \equiv \exists n.\text{Finito-n}$$

non è va bene, perché abbiamo detto che le formule Finito-n sono valide se si assegna un valore concreto a n. A questo punto ci conviene vedere se esiste una formula che vale esattamente sulle struttura finite. Supponiamo che questa esista e sia F. F sarà dunque soddisfacibile nell'insieme delle cifre decimali. Consideriamo adesso l'insieme

$$\Sigma = \{F, \text{Almeno-1}, \text{Almeno-2}, \dots, \text{Almeno-n}, \dots\}$$

Vediamo se è soddisfacibile ricorrendo al teorema di compattezza. Un generico sottoinsieme finito di Σ sarà composta da F e/o dalle formule Almeno-n, per cui sarà soddisfacibile in una struttura finita avente numero di elementi superiore a quello della massimo formula Almeno. Tutti i sottoinsiemi finiti sono soddisfacibili, e perciò lo è anche Σ. Ma in che struttura è soddisfacibile? La struttura deve soddisfare F, valida solo sulle strutture finite, ma anche le infinite formule Almeno-n, che "imporranno" un aumento del numero di elementi della struttura, che non potrà fissarsi ad un valore finito. Σ risulterà per forza soddisfacibile su una struttura infinita, e quindi non esiste una formula F che vale solo sulle strutture finite.

3 Soddisfacibilità di insiemi predicativi

Siamo giunti ora ad affrontare il caso più generale - l'insoddisfacibilità di insiemi costituiti da formule scritte usando tutto il linguaggio che abbiamo definito. Abbiamo già un bagaglio di tecniche che abbiamo sviluppato per casi più semplici e che speriamo ci aiutino in questo caso.

La prima idea è quella di trasformare le formule predicative (che contengono i quantificatori universali) in formule proposizionali di uguale soddisfacibilità, anche a costo di aumentare il "numero" di formule da trattare. Il primo passo è quella di portare, in una formula, tutti i quantificatori a sinistra ed il resto, proposizionale, a destra. Per esempio, la formula

$$B \rightarrow \forall z.C(z)$$

diventerebbe

$$\forall z.(B \rightarrow C(z))$$

posto naturalmente che z non compaia nella variabili libere di B, altrimenti si dovrebbe procedere ad un cambio di nome

$$\forall w.(C \rightarrow C(w))$$

Verifichiamo che le due formule siano equisoddisfacibili. L'interpretazione della formula di partenza è vera in due casi:

- $I^\sigma(B)$ =falso. Nella seconda formula, quindi, per ogni possibile interpretazione B sarà sempre falso (perché non dipendente da z), quindi l'intera implicazione e l'intera formula saranno interpretate in vero.
- $I^\sigma(\forall z.C(z))$. In questo caso, sarebbe vera l'interpretazione di ogni conseguente della seconda formula, rendendo vera anche l'interpretazione di quest'ultima.

Definizione 3.1 *Una* formula in forma premessa *si definisce induttivamente come segue:*

- *una formula proposizionale è in forma premessa;*
- $\exists x.A$ *è una formula in forma premessa, se* A *è in forma premessa;*
- $\forall x.A$ *è una formula in forma premessa, se* A *è in forma premessa.*

Una formula in forma premessa contiene tutti i quantificatori nella parte sinistra.

Introdurremo ora le regole che ci permetteranno di portare ad inizio formula i quantificatori, mantenendo la stessa interpretazione:

- $\neg\forall x.A \equiv \exists x.\neg A$
- $\neg\exists x.A \equiv \forall x.\neg A$
- $(\forall x.A) \wedge B \equiv \forall x.(A \wedge B) \quad x \notin FV(B)$. La condizione sulle variabili libere è importante, perché dobbiamo evitare di quantificare variabili che in B sono libere. Se accade che $x \in FV(B)$, sarà necessario cambiare il nome di x in $\forall x.A$ e poi applicare l'equivalenza.
- $(\forall x.A) \vee B \equiv \forall x.(A \vee B) \quad x \notin FV(B)$
- $(\exists x.A) \wedge B \equiv \exists x.(A \wedge B) \quad x \notin FV(B)$
- $(\exists x.A) \vee B \equiv \exists x.(A \vee B) \quad x \notin FV(B)$
- $\forall x.A \wedge \forall x.B \equiv \forall(A \wedge B)$

- $\exists x.A \vee \exists x.B \equiv \exists(A \vee B)$
- $\forall x.\forall y.A \equiv \forall y.\forall x.A$
- $\exists x.\exists y.A \equiv \exists y.\exists x.A$

Notiamo che *non* valgono le seguenti uguaglianze:

- $\forall x.A \vee \forall x.B \not\equiv \forall(A \vee B)$
- $\exists x.A \wedge \exists x.B \not\equiv \exists(A \wedge B)$
- $\forall x.\exists y.A \not\equiv \exists y.\forall x.A$

- *Portare in forma premessa la formula*

$$\neg(\exists.xP(x,y) \vee \forall z.Q(z)) \wedge \exists w.P(f(a),w)$$

Procediamo alla semplificazione:

$$\begin{aligned}
\neg(\exists.xP(x,y) \vee \forall z.Q(z)) \wedge \exists w.P(f(a),w) &= (\neg\exists x.P(x,y) \wedge \neg\forall z.Q(z)) \wedge \exists w.P(f(a),w) \\
&= (\forall x.\neg P(x,y) \wedge \exists z.\neg Q(z)) \wedge \exists w.P(f(a),w) \\
&= \forall x.(\neg P(x,y) \wedge \exists x.\neg Q(z)) \wedge \exists w.P(f(a),w) \\
&= \forall x.\exists z.(\neg P(x,y) \wedge \neg Q(x)) \wedge \exists w.P(f(a),w) \\
&= \forall x.(\exists z.(\neg P(x,y) \wedge \neg Q(z)) \wedge \exists w.P(f(a),w)) \\
&= \forall x.\exists z.\exists w(\neg P(x,y) \wedge \neg Q(z) \wedge P(f(a),w))
\end{aligned}$$

Dobbiamo ora dimostrare che con le regole introdotte riusciamo sempre a portare i quantificatori a sinistra. Notiamo che non tratteremo il caso dell'implicazione perché sappiamo esprimerlo con la negazione e la disgiunzione. Nella seguente dimostrazione si indicherà con $\mathcal{Q}$ il generico quantificatore, con $\overline{\mathcal{Q}}$ il suo opposto ($\overline{\forall} = \exists, \overline{\exists} = \forall$), con F^i la formula in ingresso e F^o la formula in uscita:

- $F^i = \neg A$. Per ipotesi induttiva, A sarà trasformato dall'algoritmo in $A^* = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n.x_n\ P$. Con le regole date si ha $F^o = \overline{\mathcal{Q}_1} x_1 \dots \overline{\mathcal{Q}_n} x_n.\neg P$. Siccome per ipotesi induttiva $A \equiv A^*$, allora anche $F^i \equiv F^o$.
- $F^i = A_1 \wedge A_2$. Per ipotesi induttiva, A_1 e A_2 saranno trasformati dall'algoritmo rispettivamente in $A_1^* = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n\ P$ e $A_2^* = \mathcal{Q}_{n+1} x_{n+1} \dots \mathcal{Q}_{n+m} x_{n+m}\ R$. Per applicare le regole, dobbiamo sincerarsi che la variabili libere in A_1^* non siano astratte in A_2^* e viceversa, altrimenti dovremmo procedere a rinominare le variabili. A questo punto si ha

  $$F^o = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n \mathcal{Q}_{n+1} x_{n+1} \dots \mathcal{Q}_{n+m} x_{n+m}\ (P \wedge R)$$

  Siccome per ipotesi induttiva $A_1 \equiv A_1^*$ e $A_2 \equiv A_2^*$, allora sarà anche $F^i = A_1 \wedge A_2 \equiv A_1^* \wedge A_2^* = F^o$.
- $F^i = A_1 \vee A_2$. Il procedimento è simile a quello del punto precedente, per cui si ottiene

  $$F^o = \mathcal{Q}_1 x_1 \dots \mathcal{Q}_n x_n \mathcal{Q}_{n+1} x_{n+1} \dots \mathcal{Q}_{n+m} x_{n+m}\ (P \vee R)$$
- $F^i = \forall x.A$. Richiamiamo ricorsivamente l'algoritmo su A, ottenendo A^*. Una volta che, direttamente o tramite rinominazione, siamo sicuri che x non appartenga alle variabili libere di A^*, si ha $F^o = \forall x.A^*$, dato che il quantificatore è già davanti. Siccome $A \equiv A^*$ per ipotesi induttiva, si ha $F^i \equiv F^o$.
- $F^i = \exists x.A$. Il discorso è analogo a quello del punto precedente, per cui si ha $F^o = \exists x.A^*$.

A questo punto dobbiamo affrontare i quantificatori e vedere come possiamo approssimarli. $\forall$ può essere considerato come una grande congiunzione di infinite formule:

$$\forall x.A \simeq A[x := t_1], A[x := t_2], \ldots$$

Siccome stiamo stabilendo la soddisfacibilità di un insieme di formule e quindi non sappiamo a priori una struttura in cui interpretare la formula, non possiamo sostituire alla variabile astratta da $\forall$ di volta in volta i nomi degli elementi della struttura: faremo questo lavoro rispetto ad ogni termine del linguaggio su cui stiamo lavorando, per ogni nome che ci è fornito dal linguaggio. L'approssimazione di $\forall$ può essere quindi espressa come segue:

$$\forall x.A \simeq \bigwedge_{t \in \text{Termini}} A[x := t]$$

$\exists$ è più problematica. Con un ragionamento analogo a quello fatto per $\forall$ otteniamo la seguente approssimazione

$$\exists x.B \simeq \bigvee_{t \in \text{Termini}} B[x := t]$$

ma così facendo otteniamo una clausola che ha un numero infinito di disgiunzioni, che non riusciamo certamente a gestire. Con $\forall$ abbiamo sì infinite congiunzioni, ma poi spezziamo la formula in infinite formule.

Per capire come affrontare il problema, consideriamo la seguente formula:

$$A \equiv \forall x.\exists y.P(x, y)$$

Interpretiamola, innanzitutto:

$$(\forall x.\exists y.P(x,y))^\sigma = \top \equiv \text{ per ogni } u \in U \text{ esiste } v \in U \ (P(x,y))^{\sigma[x:=u][y:=v]} = \top$$

L'interpretazione ci dice che se dato un elemento u riusciamo a trovare un elemento v tale che l'interpretazione di P sia vera. Possiamo quindi immaginare di avere una funzione f che dato u ci restituisce il giusto v:

$$f = \{\langle u, v\rangle \in U \times U \mid (P(x,y))^{\sigma[x:=u][y:=v]} = \top\}$$

La supposizione dell'esistenza di f è detta *assioma di scelta*. A questo punto, se $\mathfrak{U}$ è la struttura che soddisfa A, passiamo ad un'altra struttura $\mathfrak{U}^*$ che è praticamente $\mathfrak{U}$ alla quale abbiamo aggiunto la funzione f. Anche il linguaggio dovrà prevedere un simbolo in più per f, ma per il resto rimane identico a quello usato. A può essere espressa in $\mathfrak{U}^*$ senza l'uso di $\exists$:

$$A \to A^* \equiv \forall x.P(x, f(x))$$

La funzione f dipende dalla variabile astratta da $\forall$. Se prima di $\exists$ non ci fosse stato nessun $\forall$, f sarebbe una funzione senza argomenti, in pratica una costante:

$$\exists x.P(x) \to P(a)$$

- Esercizio: *eliminare i quantificatori* $\exists$ *nella seguente formula*

  $$\forall x.\exists y.\forall z.\exists w.(\neg P(a, w) \vee Q(f(x), y))$$

  Eliminiamo $\exists y$ introducendo una funzione di scelta g:

  $$\forall x.\forall z.\exists w.(\neg P(a, w) \vee Q(f(x), g(x)))$$

  Eliminiamo ora anche $\exists w$ introducendo una funzione di scelta h. Questa dipenderà sia da x che da z:

  $$\forall x.\forall z.(\neg P(a, h(x, z)) \vee Q(f(x), g(x)))$$

Cerchiamo ora di capire quali sono i termini che possiamo sostituire alle variabili introdotte da $\forall$. Come già accennato, i nomi vanno ricavati dal linguaggio usato per esprimere le formule: si individuano le costanti e i simboli che denotano le funzioni (anche quelle di scelta introdotte), e si combinano per ottenere tutti i termini possibili. Questi termini sono detti *termini ground* o *termini chiusi*.

- Esercizio: nell'esercizio precedente, abbiamo ottenuto la formula

$$\forall x.\forall z.(\neg P(a, h(x, z)) \vee Q(f(x), g(x)))$$

Notiamo che abbiamo una costante a e tre funzioni f, g e h. Da queste possiamo ottenere un'infinità di nomi:

$$\begin{gathered} a, f(a), f^2(a), \ldots \\ g(a), g^2(a), \ldots \\ f(g(a)), f^2(g(a)), \ldots \\ h(a, f(a)), h(g(a), a), \ldots \\ \ldots \end{gathered}$$

I nomi che abbiamo a disposizione vanno sostituiti in tutti i modi possibili nelle variabili astratte previste dai quantificatori $\forall$. A questo punto verranno generate un consistente numero di formule proposizionali (anche infinite) che possono essere affrontate con il metodo di risoluzione e il teorema di compattezza. Proporremo alcuni esempi per chiarire meglio la strategia da adottare.

- Esercizio: *stabilire se il seguente insieme è insoddisfacibile:*

$$\{\forall.P(x) \wedge \exists y.\neg P(f(y))\}$$

Portiamo l'unica formula presente in forma premessa:

$$\{\forall x.\exists y.(P(x) \wedge \neg P(f(y)))\}$$

Ora eliminiamo i quantificatori $\exists$:

$$\{\forall x.(P(x) \wedge \neg P(f(g(x))))\}$$

Ora dobbiamo ricavare i nomi. Notiamo però che non abbiamo nessuna costante esplicitata nel linguaggio usato. Siccome la struttura deve contenere sicuramente elementi, introduciamo noi una costante a (cambiando così struttura). Ora possiamo combinarla con le funzioni f e g presenti:

$$a, g(a), g^2(a), \ldots, f(a), f^2(a), \ldots$$

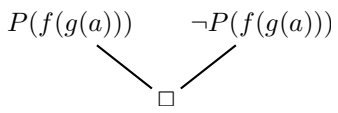
Riportiamo uno "spaccato" dell'insieme di formule proposizionali che abbiamo creato:

$$\{\ldots, P(t_1), \neg P(f(g(t_1))), \ldots\}$$

dove t_1 denota un generico termine ground. Intravediamo una possibilità di ottenere la clausola vuota. Consideriamo le clausole generate per $x := a$ e $x := f(g(a))$:

$$\{\ldots, P(a), \neg P(f(g(a))), P(f(g(a))), \neg P(f(g(f(g(a))))), \ldots\}$$

La risoluzione è ora evidente:



L'insieme dato è dunque insoddisfacibile.

Al posto di procedere immediatamente all'espansione dei $\forall$, è possibile scrivere le formule senza questi quantificatori e dopo capire quali sono le sostituzioni da svolgere man mano che permettono la risoluzione tra le varie formule e il raggiungimento della clausola vuota. La sostituzione in questo caso è detta *unificazione*, perché è fatta in modo tale da rendere le stringhe interessate uguali.

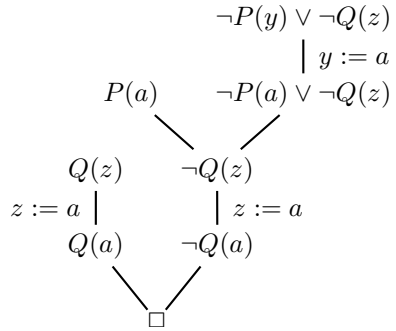
- Esercizio: *stabilire se la seguente formula è una verità logica.*

$$(\exists x.P(x) \wedge Q(x)) \rightarrow (\exists x.P(x) \wedge \exists x Q(x))$$

Dobbiamo quindi stabilire se l'insieme composto dalla negazione della formula è insoddisfacibile. Procediamo ora alla trasformazione dell'insieme:

$$\{\neg((\exists x.P(x) \wedge Q(x)) \rightarrow (\exists x.P(x) \wedge \exists x.Q(x)))\}$$
$$\{\neg(\neg(\exists x.P(x) \wedge Q(x)) \vee (\exists x.P(x) \wedge \exists x.Q(x)))\}$$
$$\{(\exists x.P(x) \wedge Q(x)) \wedge \neg(\exists x.P(x) \wedge \exists x.Q(x))\}$$
$$\{(\exists x.P(x) \wedge Q(x)) \wedge (\neg\exists x.P(x) \vee \neg\exists x.Q(x))\}$$
$$\{(\exists x.P(x) \wedge Q(x)) \wedge (\forall x.\neg P(x) \vee \forall x.\neg Q(x))\}$$
$$\{\exists x.P(x) \wedge Q(x), \forall y.\neg P(y) \vee \forall z.\neg Q(z)\}$$
$$\{P(a) \wedge Q(a), \forall y.\forall z.(\neg P(y) \vee \neg Q(z))\}$$
$$\{P(a), Q(a), \neg P(y) \vee \neg Q(z)\}$$

Con il linguaggio dato, l'unico nome utilizzabile è la costante a, dato che non ci sono altre costanti e funzioni. A questo punto cominciamo la risoluzione operando le giuste unificazioni:



L'insieme risulta insoddisfacibile, perciò la formula data è una verità logica.