

LA LOGICA DI PROLOG

Giovanna D'Agostino,
Dipartimento di Matematica e Informatica Università di Udine
Versione del 20/11/09

1. PROLOG

Definizione 1.1. Una clausola di Horn è una clausola che contiene al più un letterale positivo. Una clausola di Horn è detta clausola di programma se contiene esattamente un letterale positivo, mentre è detta goal se non contiene alcun letterale positivo. Una clausola di programma che non contiene letterali negativi è detta un atto; se contiene letterali negativi si chiama regola. In una regola, l'unico letterale positivo si chiama la testa della regola, mentre se $\neg A_1, \dots, \neg A_k$ sono i letterali negativi, allora $\{A_1, \dots, A_k\}$ si chiama il corpo della regola.

Prolog utilizza una speciale notazione per le clausole di programma e per i goals. Nella tabella 1 riassumiamo le differenti notazioni per una clausola di programma $\{A, \neg A_1, \dots, \neg A_k\}$ ed un goal $\{\neg A_1, \dots, \neg A_k\}$ (dove $A, A_1, \dots, A_k$ sono letterali positivi che contengono le variabili $\bar{z}$), quella logica, quella insiemistica e quella usata in Prolog.

TABELLA 1. Notazioni

Notazione Logica	Notazione Insiemistica	Notazione Prolog
$\forall \bar{z}(A_1 \wedge \dots \wedge A_k \rightarrow A)$	$\{A, \neg A_1, \dots, \neg A_k\}$	$A : -A_1, \dots, A_k$
$\forall \bar{z}(\neg A_1 \vee \dots \vee \neg A_k)$	$\{\neg A_1, \dots, \neg A_k\}$	$: -A_1, \dots, A_k$

Un programma Prolog $\mathcal{P}$ è un insieme finito di clausole di programma. Dato un goal $G =: -A_1, \dots, A_k$, poiché $\neg G$ è equivalente all'enunciato $\exists \bar{z}(A_1 \wedge \dots \wedge A_k)$, dove $\bar{z}$ è una sequenza che contiene tutte e sole le variabili presenti negli A_i , si ha

$$\mathcal{P} \cup \{G\} \text{ insoddisfacibile} \Leftrightarrow \mathcal{P} \models \exists \bar{z}(A_1 \wedge \dots \wedge A_k).$$

Come abbiamo visto nell'esercizio 6 della precedente sezione, (solo nel caso proposizionale, ma il risultato si estende anche alla logica predicativa) per stabilire se $\mathcal{P} \cup \{G\}$ è insoddisfacibile è possibile restringersi a risoluzioni lineari da input. In particolare, la refutazione cercata può sempre partire dal goal G e da una clausola di programma. In Prolog proposizionale per fare un passo di risoluzione fra un goal $: -A_1, \dots, A_k$ ed una regola $A : -B_1, \dots, B_n$ (un fatto $A : -$) occorre che la testa della regola (o del fatto) A sia uguale ad A_i per un qualche i ; il risultato della risoluzione è allora il goal $: -A_1, \dots, A_{i-1}, B_1, \dots, B_n, A_{i+1}, \dots, A_k$.

Ad esempio:

$$: -r(a, b), r(fa, b), q(a) \text{ e } r(fa, b) : -q(fb), r(b, b) \text{ si risolvono nel goal } : -r(a, b), q(fb), r(b, b), q(a).$$

In Prolog predicativo, bisognerà unificare la testa della regola (o del fatto) con uno o più letterali del goal. Ad esempio, il goal $: -r(z, y), r(fy, y), q(a), r(z, z)$ ed la regola $r(fv, gu) : -q(fv), r(v, v)$ si possono risolvere unificando $r(fv, gu)$ con $r(z, y), r(fy, y)$: l' m.g.u. corrispondente è

$$\sigma = \{z/fgu, v/gu, y/gu\}$$

e il risultato della risoluzione è il goal

$$: -q(a), r(fgu, fgu), r(gu, gu)$$

Riassumendo, nel cercare una refutazione di $\mathcal{P} \cup \{G\}$ possiamo limitarci alle risoluzioni che iniziano con G e una clausola di programma, ottenendo un nuovo goal che cercheremo di risolvere con una nuova clausola di programma etc, fino ad arrivare alla clausola vuota.

Ad esempio, supponiamo che

$$\mathcal{P}_{\leq} = \begin{cases} NaCl : -NaOH, HCl; \\ H_2O : -NaOH, HCl; \\ NaOH : - \\ HCl : - \end{cases}$$

dove $NaOH$ è l'idrossido di sodio, HCl è l'acido cloridrico, $NaCl$ è cloruro di Sodio e H_2O è l'acqua. Se G è il goal $: -NaCl, H_2O$, possiamo ottenere una refutazione di $\mathcal{P} \cup \{G\}$ in due passi:

$$\begin{array}{ll} : -NaCl, H_2O & NaCl : -NaOH, HCl \\ : -NaOH, HCl, H_2O & NaOH : - \\ : -HCl, H_2O & HCl : - \\ : -H_2O & H_2O : -NaOH, HCl \\ : -NaOH, HCl & NaOH : - \\ : -HCl & HCl : - \\ : - & \end{array}$$

1.1. Esempi aritmetici: programmi per $\leq$, add sui naturali. Consideriamo il linguaggio $L = \{0, S, \leq, add, mult\}$, dove 0 è un simbolo per costante, S un simbolo funzionale unario, $\leq$ un simbolo relazionale binario e add è un simbolo relazionale ternario. Per semplicità notazionale, utilizzeremo il simbolo $\bar{n}$ per denotare il termine chiuso $S^n(0)$.

Sia $\mathcal{N}$ la seguente interpretazione di L (detta interpretazione *standard* di L):

$$\begin{aligned} D^{\mathcal{N}} &= \mathbb{N}, \quad 0^{\mathcal{N}} = 0, \quad S^{\mathcal{N}}(n) = n + 1, \\ \leq^{\mathcal{N}} &= \{(n, m) \in \mathbb{N}^2 : n \leq m\}, \quad add^{\mathcal{N}} = \{(n, m, k) \in \mathbb{N}^3 : n + m = k\}. \end{aligned}$$

Nello scrivere programmi prolog sui numeri naturali, utilizzeremo clausole vere nel modello standard:

$$\begin{aligned} \mathcal{P}_{\leq} &= \begin{cases} 0 \leq x : - \\ Sx \leq Sy : -x \leq y \end{cases} \\ \mathcal{P}_{add} &= \begin{cases} add(x, \bar{0}, x) : - \\ add(x, Sy, Sz) : -add(x, y, z) \end{cases} \end{aligned}$$

Il programma Prolog viene memorizzato nel computer. L'utente interroga il programma tramite goals. Il compilatore unisce il programma Prolog $\mathcal{P}$ che ha in memoria con il goal $G = \{\neg A_1, \dots, \neg A_k\}$ che gli viene fornito come input e risponde (utilizzando un raffinamento di risoluzione) alla domanda

$$\mathcal{P} \cup \{G\} \text{ è insoddisfacibile?}$$

o, equivalentemente,

$$\text{È vero che } \mathcal{P} \models \exists \bar{z}(A_1 \wedge \dots \wedge A_k)?$$

Possiamo utilizzare questo strumento per scrivere programmi che, ad esempio, calcolano funzioni ricorsive, o interrogano una base di dati, o che generano tutte le liste che sono permutazioni di una data lista, etc. etc.

Facciamo un esempio aritmetico, considerando di nuovo il programma P_{add} descritto sopra e i goal chiusi, $G =: -add(\bar{1}, \bar{1}, \bar{2})$ e $G' =: -add(\bar{1}, 0, \bar{2})$.

È chiaro che $P_{add} \cup \{G'\}$, cioè $P_{add} \cup \{-add(\bar{1}, 0, \bar{2})\}$ è soddisfacibile: infatti il nostro modello privilegiato $\mathcal{N}$ rende vere tutte le clausole di P_{add} e anche $-add(\bar{1}, 0, \bar{2})$. Quindi il compilatore risponderà *NO* alla nostra interrogazione, coerentemente con le nostre aspettative: infatti $1 + 0$ non è uguale a 2.

Consideriamo ora il goal G . Notiamo che $\neg G$, o , equivalentemente $add(\bar{1}, \bar{1}, \bar{2})$ è vera in $\mathcal{N}$. Vorremmo quindi che il compilatore, usando risoluzione, risponda *SI* alla nostra interrogazione. Come possiamo essere sicuri che questo avvenga? Il solo fatto che le clausole di P_{add} siano vere in $\mathcal{N}$ non ci garantisce in generale che $P_{add} \models \neg G$, ovvero che $P_{add} \cup \{G\}$ sia insoddisfacibile: l'insieme di clausole potrebbe infatti avere un altro modello diverso da I e il compilatore, usando risoluzione, non darebbe la risposta *SI* che ci aspettiamo.

Nel caso generale il programma prolog P e il goal utilizzato nella interrogazione sono espressi in un dato linguaggio al prim'ordine L . Abbiamo in mente una particolare interpretazione I di L e quello che vogliamo ottenere dal programma è che riesca a refutare (via risoluzione) tutti e soli i goal che sono falsi in I . (Nota bene: se il goal $G = \{\neg A_1, \dots, \neg A_k\}$ è privo di variabili, abbiamo che G è falso in I se e solo se $A_1 \wedge \dots \wedge A_k$ è vero in I).

In sintesi, vogliamo scrivere un programma P per cui valga, per ogni goal chiuso $G =: -A_1, \dots, A_k$

$$P \cup \{G\} \text{ è insoddisfacibile } \Leftrightarrow I \models A_1 \wedge \dots \wedge A_k.$$

Notiamo che, come nell'esempio precedente, la freccia da sinistra a destra è sempre garantita se $I \models P$: se il programma è vero in I , allora è corretto. È più difficile, invece, scrivere programmi completi, cioè programmi per i quali vale anche la freccia da destra a sinistra. Infatti, I è solo uno dei tanti modelli di P e dobbiamo garantire che se $\neg G$ è vero in I allora P ha conseguenza logica il goal $\neg G$ (cioè: $P \cup \{G\}$ è insoddisfacibile).

Dimostriamo ad esempio che il programma $P_{\leq}$ è corretto e completo rispetto al modello standard:

$$P_{\leq} = \begin{cases} 0 \leq x : - \\ Sx \leq Sy : - x \leq y \end{cases}$$

Correttezza rispetto ai goals chiusi. Se $P_{\leq} \cup \{:-\bar{n} \leq \bar{m}\}$ è insoddisfacibile , allora $\mathcal{N} \models \bar{n} \leq \bar{m}$: basta notare che $\mathcal{N}$ è un modello del programma $P_{\leq}$.

Completezza rispetto ai goals chiusi. Se $\mathcal{N} \models \bar{n} \leq \bar{m}$ allora $P_{\leq} \cup \{:-\bar{n} \leq \bar{m}\}$ è insoddisfacibile. Si dimostra per induzione su n : passo induttivo: se $\mathcal{N} \models \bar{n} + \bar{1} \leq \bar{m}$ allora $m \geq 1$, $m - 1 \in \mathcal{N}$ e $\mathcal{N} \models \bar{n} \leq \overline{m-1}$. Quindi, per induzione, $P \models \bar{n} \leq \overline{m-1}$. Ma P contiene la formula $\forall x \forall y (x \leq y \rightarrow Sx \leq Sy)$, e

$$\forall x \forall y (x \leq y \rightarrow Sx \leq Sy), \bar{n} \leq \overline{m-1} \models \bar{n} + \bar{1} \leq \bar{m}.$$

Esempi di programmi corretti ma non completi:

$$P = \begin{cases} \bar{0} \leq x : - \\ x \leq Sy : - x \leq y \end{cases}$$

Il programma è certamente corretto, ma i goals : $-\bar{n} \leq \bar{m}$ per cui esiste una refutazione dal programma sono solo quelli con $n = 0$. Infatti il sottoinsieme della base

$$H = \{(\bar{0}, \bar{m}) : m \in \mathcal{N}\}$$

è un modello di Herbrand del programma P e ogni goal del tipo $G = \{-\bar{n} \leq \bar{m}\}$ con $n > 0$ è vero in H . Ne segue che $P \cup \{G\}$ è soddisfacibile e P non refuta G .

1.2. Goals con variabili. Se $G =: -A_1, \dots, A_k$ si ha

$$P \cup \{G\} \text{ è insoddisfacibile } \Leftrightarrow P \models \exists \bar{z} (A_1 \wedge \dots \wedge A_k),$$

dove $\bar{z}$ contiene tutte e sole le variabili che compaiono nei letterali $A_1, \dots, A_k$. Anche nel caso dei goal con variabili, vogliamo che valga un'ulteriore equivalenza con $I \models \exists \bar{z} (A_1 \wedge \dots \wedge A_k)$, dove I è il modello privilegiato.

Come possiamo utilizzare goals con variabili?

- (1) Per calcolare funzioni ricorsive. Facciamo un esempio considerando il programma P_{add} del paragrafo precedente.

$$P_{add} = \begin{cases} add(x, \bar{0}, x) : - \\ add(x, Sy, Sz) : -add(x, y, z) \end{cases}$$

Se G è il goal : $-add(\bar{n}, \bar{m}, z)$, varrà:

$$P_{add} \cup \{G\} \Leftrightarrow \mathcal{N} \models \exists z add(\bar{n}, \bar{m}, z)$$

Notiamo anche che $\mathcal{N}$ è un modello di Herbrand: quindi, se $\mathcal{N} \models \exists z F$ allora esiste un termine chiuso $\bar{k}$ tale che $\mathcal{N} \models \exists z add(\bar{n}, \bar{m}, \bar{k})$ e, per definizione di $\mathcal{N}$, abbiamo $k = n + m$.

Quando si interroga il programma P_{add} rispetto al goal : $- : -add(\bar{n}, \bar{m}, z)$ il programma ci fornirà come vedremo come *risposta calcolata* proprio l'unico numero k per cui vale $n + m = k$.

- (2) Per invertire funzioni ricorsive. Ad esempio, interrogando il programma P_{add} rispetto al goal : $-add(x, y, \bar{3})$ possiamo ottenere tutte le coppie di naturali (n_1, n_2) per cui vale $n_1 + n_2 = 3$, mentre, se $n \leq m$, possiamo ottenere il numero $m - n$ interrogando il programma rispetto al goal : $-add(x, \bar{n}, \bar{m})$.

1.3. Liste. Linguaggio delle liste: $L_{list} = \{\epsilon, [-|-]\}$ dove ϵ è una costante per la lista vuota e $[-|-]$ è un simbolo funzionale binario. Dato un oggetto t e una lista L , il termine $[t|L]$ rappresenta la lista che ha come primo elemento t seguito dalla lista L .

Nota Bene: se ϵ rappresenta la lista vuota, allora $[\epsilon|\epsilon]$ rappresenta la lista il cui unico elemento è ϵ , mentre se t è un termine, $[t|\epsilon]$, rappresenta la lista il cui unico elemento è t . Il termine $[t_1|[t_2|\epsilon]]$ rappresenta una lista che ha due elementi, nell'ordine t_1, t_2 . Useremo la notazione $[t_1, t_2]$ come abbreviazione per il termine $[t_1|[t_2|\epsilon]]$. Più in generale, useremo $[t_1, t_2, \dots, t_n|L]$ come abbreviazione del termine

$$[t_1|[t_2|\dots[t_n|L]\dots]],$$

e quando L è la lista vuota, denoteremo la lista di cui sopra semplicemente con $[t_1, \dots, t_n]$.

L_{list} può essere aggiunto ad un qualsiasi linguaggio per creare liste di un certo tipo. Ad esempio, considerandolo insieme al linguaggio per i naturali possiamo costruire e operare su liste di naturali etc.

Attenzione alle abbreviazioni nelle unificazioni! Ad esempio: se nel linguaggio ci siano due costanti a, b e cerchiamo di unificare $[a, b]$ con il termine $[w|[v|\epsilon]]$, bisogna tener presente che $[a, b]$ è solo

un'abbreviazione per il termine $[a|[b|\epsilon]]$ e applicando l'algoritmo di unificazione otteniamo l'm.g.u. $\sigma = \{w|a, v|b\}$.

1.4. Esempi di programmi sulle liste.

Un programma per riconoscere le liste (di tipo qualsiasi):

$$P_{LIST} = \begin{cases} LIST(\epsilon) : - \\ LIST([X|L]) : - LIST(L) \end{cases}$$

Un programma per riconoscere i prefissi di una lista:

$$P_{prefix} \begin{cases} prefix(\epsilon, L) : - \\ prefix([A|S], [A|L]) : - prefix(S, L) \end{cases}$$

Un programma per riconoscere le sottoliste (da aggiungere al programma P_{prefix}):

$$P_{subl} \begin{cases} subl(S, L) : - prefix(S, L) \\ subl(S, [X|L]) : - subl(S, L) \end{cases}$$

Un programma per concatenare liste:

$$P_{append} \begin{cases} append(\epsilon, L, L) : - \\ append([x|L], L', [x|L'']) : - append(L, L', L'') \end{cases}$$

Un programma per invertire gli elementi di una lista (versione naive):

$$P_{rev} \begin{cases} rev(\epsilon, \epsilon) : - \\ rev([x|L], S) : - rev(L.T), app(T, [x], S) \\ append(\epsilon, R, R) : - \\ append([x|R], U, [x|V]) : - append(R, U, V) \end{cases}$$

Un programma più efficiente per invertire i membri di una lista, usando un accumulatore:

$$P_{REV} \begin{cases} REV(x, y) : - ACC(x, \epsilon, y) \\ ACC(\epsilon, z, z) : - \\ ACC([x|w], z, y) : - ACC(w, [x|z], y) \end{cases}$$

2. ESERCIZI

(1) Dato il linguaggio $\mathcal{L} = \{0, S, add\}$, sia P il seguente programma prolog:

$$P = \begin{cases} add(X, 0, X) : - \\ add(SX, SY, SSZ) : - add(X, Y, Z) \end{cases}$$

- i) P è corretto per i goals di tipo $-add(S^n 0, S^m 0, S^k 0)?$ (cioè: è vero che se $P \models add(S^n 0, S^m 0, S^k 0)$ allora $n + m = k$ nei numeri naturali?)
- ii) P è completo per i goals di tipo $-add(S^n 0, \bar{m}, S^k 0)?$ (cioè: è vero che se $n + m = k$ nei numeri naturali allora $P \models add(S^n 0, \bar{m}, S^k 0)?$)
- iii) Descrivi quali sono i goal chiusi refutati da P .

- (2) Sia $\mathcal{P}_{prefix}$ il seguente programma:

$$\mathcal{P}_{prefix} \begin{cases} prefix(\epsilon, L) : - \\ prefix([A|S], [A|L]) : -prefix(S, L) \end{cases}$$

Quali sono le risposte calcolate per il goal : $-prefix(z, [a, b])$, dove a, b sono costanti?
 Quali sono le risposte calcolate per il goal : $-prefix([a, b], z)$, dove a, b sono costanti?

- (3) Sia $\mathcal{L} = \{0, S, <, add, resto\}$ dove *resto* è un simbolo predicativo ternario. Scrivere un programma prolog P nel linguaggio $\mathcal{L}$ tale che, per ogni terna di numeri naturali n, m, k valga:

$$P \models resto(S^n 0, S^m 0, S^k 0) \quad \Leftrightarrow \quad k \text{ è il resto della divisione di } n \text{ per } m.$$

Dimostrare la correttezza e la completezza del programma.

- (4) Dato il programma

$$P = \begin{cases} add(0, x, x) : - \\ add(x, Sy, z) : -add(Sx, y, z) \end{cases}$$

rispondere alle seguenti domande, giustificando in modo appropriato le risposte:

- se n, m sono naturali, G è il goal : $-add(S^n 0, S^m 0, S^k 0)$, è vero che : $-add(S^n 0, S^m 0, S^k 0)$ è vera nei numeri naturali?(interpretando i simboli nel modo usale)
 - Per quali n, m naturali esiste una refutazione del goal : $-add(S^n 0, S^m 0, S^{n+m} 0)$ dal programma?
 - Rispondere alle domande precedenti se la seconda clausola è sostituita dalla clausola $add(Sx, y, z) : -add(x, Sy, z)$.
- (5) Sia $\mathcal{L} = \{0, S, add\}$ l'usuale linguaggio dei numeri naturali con addizione (dove 0 è la costante "zero", S il simbolo di funzione unario "successore", e add è il simbolo relazionale ternario "addizione"). Scrivere un programma prolog P_{fib} nel linguaggio $\mathcal{L} \cup \{fib\}$, dove fib è un simbolo relazionale binario, in modo che P_{fib} "calcoli" la funzione di fibonacci f definita ricorsivamente da

$$\begin{cases} f(0) = f(1) = 1; \\ f(n+2) = f(n) + f(n+1). \end{cases}$$

Fare la stessa cosa per calcolare la moltiplicazione $f(n, m) = nm$ e l'elevamento a potenza $f(n, m) = n^m$.

- (6) Utilizzando il programma per l'addizione, scrivere un programma Prolog per riconoscere il predicato di divisibilità sui naturali

$$div(n, m) \Leftrightarrow n \text{ divide } m.$$

- (7) Sia P il seguente programma PROLOG:

$$P = \begin{cases} add(X, 0, X) : - \\ add(X, S(Y), S(Z)) : -add(X, Y, Z) \\ d(X, X) : - \\ d(X, Z) : -add(Y, Y, Z), d(X, Y) \end{cases}$$

- i) Dimostrare che se $P \cup \{:-d(S^n 0, S^m 0)\}$ ha una refutazione, allora il numero naturale n divide m .
- ii) È vero che, se il numero naturale n divide m , allora $P \cup \{:-d(S^n 0, S^m 0)\}$ ha una refutazione?
- iii) Come cambiano le risposte alle precedenti domande se l'ultima riga del programma è sostituita da

$$d(X, Z) : -add(Y, Y', Z), d(X, Y), d(X, Y')?$$

- (8) Dato il linguaggio $\mathcal{L} = \{\epsilon, [\mid], adc\}$ dove $adc(x, y, z)$ è un predicato ternario, scrivere un programma prolog in modo da esprimere la relazione “ x e y sono elementi adiacenti nella lista z ”.
- (9) Dato l'usuale linguaggio $\mathcal{L} = \{\epsilon, [\mid]\}$ per le liste, considera il programma P che descrive il predicato *append* per concatenare una lista ad un'altra:

$$P = \begin{cases} append(\epsilon, Y, Y) : - \\ append([X|Y], V, [X|W]) : -append(Y, V, W) \end{cases}$$

Sia l la funzione lunghezza sull'insieme T_c dei termini chiusi del linguaggio $\mathcal{L}$ (definita ricorsivamente da $l(\epsilon) = 0$, $l([t_1|t_2]) = 1 + l(t_2)$). Considera la seguente interpretazione di Herbrand del linguaggio $\mathcal{L} \cup \{append\}$:

$$H = \{append(t, u, v) : t, u, v \in T_c, \quad l(t) + l(u) = l(v)\}.$$

- (i) H è modello di P ?
 - (ii) P ha un modello minimo? Se sì, descrivilo.
- (10) Sia $\mathcal{L} = \{\epsilon, [\mid],\}$ l'usuale linguaggio per le liste.
- i) Scrivere un programma PROLOG che definisca il predicato *scambio12*(Y, Z) la cui semantica è: “la lista Z è uguale alla lista ottenuta scambiando il primo e il secondo elemento della lista Y ”.
- Soluzione:

$$scambio12([Y_1|[Y_2|Z]], [Y_2|[Y_1|Z]]) : -$$

- ii) Scrivere un programma PROLOG che definisca il predicato *scambio*(Y, Z) la cui semantica è: “la lista Z è uguale alla lista ottenuta scambiando il primo e l'ultimo elemento della lista Y ”.

3. INTERPRETAZIONI DI HERBRAND

Dato un linguaggio L , indichiamo con T_c l'insieme dei termini chiusi di L . Data un'interpretazione I di L ed un termine chiuso $t \in T_c$, l'interpretazione di t in I è un elemento del dominio D^I definito induttivamente come segue: se t è una costante c allora l'interpretazione di t in I ci viene già fornita dall'interpretazione I : $t^I := c^I$;
 se $t = f(t_1, \dots, t_n)$, allora $t^I := f^I(t_1^I, \dots, t_n^I)$.
 Poiché consideriamo solo termini chiusi, non abbiamo bisogno di alcuno stato su I per interpretare il termine; inoltre, qualsiasi sia lo stato σ su I vale $\sigma(t) = t^I$.

Proposizione 3.1. *Se $I, \sigma \models \forall x F$ allora per ogni termine chiuso t vale $I, \sigma \models F\{x/t\}$.*

Dimostrazione. Se $t \in T_c$ allora $\sigma(t) \in D^I$; da $I, \sigma \models \forall x F$ segue allora $I, \sigma[x/\sigma(t)] \models F$; poiché ogni termine chiuso t è libero per la sostituzione ad x in F , applicando il Lemma di sostituzione (Lemma 2.28 delle dispense di Elementi di Logica) otteniamo $I, \sigma \models F\{x/t\}$. Quindi, se $I, \sigma \models \forall x F$ allora per ogni termine chiuso t vale $I, \sigma \models F\{x/t\}$. $\square$

In generale il viceversa non vale, cioè, non è detto che se in un'interpretazione sono vere tutte le formule del tipo $F\{x/t\}$, al variare di $t \in T_c$, allora sia vera anche $\forall x F$. Ad esempio, dato $L = \{a, p\}$, con a costante e p simbolo predicativo unario, sia $D^I = \{0, 1\}$, $a^I = 0$, $p^I = \{0\}$; per ogni termine chiuso $t \in T_c$ vale $I \models p(a)$ (infatti l'unico termine chiuso del linguaggio è la costante a), ma $I \not\models \forall x F$.

Per ottenere la proprietà richiesta abbiamo bisogno di restringere la classe delle interpretazioni. Data un'interpretazione I di L ed un elemento $d \in D^I$, diciamo che d è il valore di un termine chiuso se esiste $t \in T_c$ tale che $t^I = d$.

Lemma 3.2. *Se un'interpretazione ha la proprietà che ogni elemento del dominio è valore di almeno un termine chiuso, vale: per ogni stato σ e formula F*

$$I, \sigma \models \forall x F \Leftrightarrow \forall t \in T_c, \quad I, \sigma \models F\{x/t\},$$

$$I, \sigma \models \exists x F \Leftrightarrow \exists t \in T_c, \quad I, \sigma \models F\{x/t\}.$$

Dimostrazione. Supponiamo che $I, \sigma \models F\{x/t\}$, per ogni termine chiuso $t \in T_c$. Per il Lemma di sostituzione abbiamo allora che per ogni termine chiuso $t \in T_c$ si ha $I, \sigma[x/\sigma(t)] \models F$. Ma $\sigma(t) = t^I$ e per ipotesi per ogni elemento del dominio d esiste $t \in T_c$ con $d = t^I$. Se ne deduce che $\forall d \in D^I, I\sigma[x/d] \models F$, cioè $I, \sigma \models \forall x F$. Abbiamo già visto che l'altra direzione della doppia freccia vale sempre. Il caso del quantificatore esistenziale è simile e viene lasciato per esercizio. $\square$

Per ottenere interpretazioni in cui ogni elemento del dominio è valore di almeno un termine chiuso, l'idea più semplice è quella di prendere proprio l'insieme dei termini chiusi come elementi del dominio, e fare in modo che ogni termine chiuso venga interpretato in sé stesso. Questa è l'idea che è alla base della definizione di interpretazione di Herbrand.

Definizione 3.1. *Dato un linguaggio predicativo L , un'interpretazione di Herbrand di L è un'interpretazione H di L che soddisfa le seguenti proprietà:*

- i) *il dominio D^H di H è l'insieme dei termini chiusi di L , se L contiene almeno una costante; se L non contiene costanti, si considera il linguaggio $L' = L \cup \{c\}$ dove c è una nuova costante e il dominio di H è l'insieme dei termini chiusi di L' , e le definizioni che seguono vanno riferite ad L' ;*
- ii) *per ogni simbolo di costante c di L , $c^H = c$;*

iii) per ogni simbolo funzionale f di L di arietà n , e termini chiusi $t_1, \dots, t_n$, $f^H(t_1, \dots, t_n) = f(t_1, \dots, t_n)$.

Notiamo che, fissato il linguaggio L , il dominio e l'interpretazione dei simboli funzionali e delle costanti è lo stesso per tutte le interpretazioni di Herbrand di L . Quello che varia è solo l'interpretazione dei simboli relazionali.

Esercizio Dimostrare per induzione sull'altezza del termine chiuso t che se H è un'interpretazione di Herbrand allora $t^H = t$.

Dall'esercizio precedente segue che le interpretazioni di Herbrand soddisfano le ipotesi del Lemma 3.2, e quindi per ogni interpretazione di Herbrand H , per ogni stato σ e formula F vale:

$$\begin{aligned} H, \sigma \models \forall x F &\Leftrightarrow \forall t \in T_c, \quad H, \sigma \models F\{x/t\}, \\ H, \sigma \models \exists x F &\Leftrightarrow \exists t \in T_c, \quad H, \sigma \models F\{x/t\}. \end{aligned}$$

Dato un linguaggio L , l'insieme B_L delle formule atomiche chiuse è detto *base di Herbrand di L* . Un'interpretazione di Herbrand H è caratterizzata dal sottoinsieme della base di Herbrand dato da tutte le formule atomiche chiuse che sono vere in H . Nel seguito, identificheremo un modello di Herbrand con tale sottoinsieme ed in questo senso parleremo di unione e intersezione di interpretazioni di Herbrand.

Ad esempio, si consideri il linguaggio $L = \{a, f, p, q\}$ con f, p simboli unari, rispettivamente funzionale e predicativo e q simbolo predicativo binario; sia H l'interpretazione di Herbrand su L definita da

$$p^H = \{a, f^2(a), \dots, f^{2n}(a), \dots\}, \quad q^H = \{(a, fa), (fa, a), (fa, fa)\}.$$

(poiché H è di Herbrand, il suo dominio e l'interpretazione dei simboli di costante e funzionali è fissata a priori); Identifichiamo H con l'insieme delle formule atomiche chiuse vere in H (che denoteremo per abuso di notazione ancora con H) e cioè:

$$H = \{p(a), p(f^2(a)), \dots, p(f^{2n}(a)), \dots\} \cup \{q(a, fa), q(fa, a), q(fa, fa)\}.$$

Se inoltre H' è definito da

$$p^{H'} = \{a, f(a)\}, \quad q^{H'} = \{(t, t) : t \in T_c\},$$

l'interpretazione H' corrisponde al seguente sottoinsieme di B_L

$$H' = \{\{pa, pf(a)\} \cup \{q(t, t) : t \in T_c\}\};$$

possiamo allora definire $H \cap H'$ come l'interpretazione di Herbrand che corrisponde al sottoinsieme

$$H \cap H' = \{p(a), q(fa, fa)\};$$

avremo quindi:

$$p^{H \cap H'} = \{a\}, \quad q^{H \cap H'} = \{(fa, fa)\}.$$

Le interpretazioni di Herbrand sono particolarmente utili quando ci si restringe a enunciati universali. Infatti vale:

Lemma 3.3. *Se un insieme di enunciati universali è soddisfacibile allora ha un modello di Herbrand.*

Dimostrazione. Se $I \models \Gamma$, allora l'interpretazione di Herbrand definita da

$$H = \{A \in B_L : I \models A\}$$

rende veri tutti gli enunciati (cioè formule senza variabili libere) privi di quantificatori che sono veri in I (si dimostra facilmente per induzione sulla complessità di tali formule). In particolare H rende vero l'insieme delle istanze chiuse Γ^c di Γ . Siccome H è di Herbrand, ne segue $H \models \Gamma$. $\square$

4. MODELLO MINIMO DI UN PROGRAMMA PROLOG

Un programma Prolog $\mathcal{P}$ è sempre soddisfacibile: infatti l'interpretazione H che corrisponde a tutte le formule atomiche chiuse (cioè, l'interpretazione di Herbrand che rende vere tutte le formule atomiche chiuse) è modello di $\mathcal{P}$. Un programma Prolog, inoltre, ammette un modello di Herbrand $M_{\mathcal{P}}$, detto il modello minimo del programma, che rende veri tutti e soli gli atomi chiusi che sono conseguenza logica del programma. L'insieme $M_{\mathcal{P}} = \{A \in B_L : \mathcal{P} \models A\}$ può essere caratterizzato come segue.

Lemma 4.1. *Sia $\mathcal{P}$ un programma Prolog e $M_{\mathcal{P}} = \{A \in B_L : \mathcal{P} \models A\}$. Si ha:*

- (1) $M_{\mathcal{P}} = \{A \in B_L : \text{esiste una refutazione del goal } -A \text{ da } \mathcal{P}\}$;
- (2) $M_{\mathcal{P}}$ è un modello di Herbrand di $\mathcal{P}$;
- (3) $M_{\mathcal{P}}$ è minimo fra i modelli di $\mathcal{P}$ rispetto all'inclusione (fra sottoinsiemi della base);
- (4) $M_{\mathcal{P}} = \bigcap \{H : H \text{ è un modello di Herbrand di } \mathcal{P}\}$;
- (5) Se $M_0 = \{A\theta : A : - \in \mathcal{P}, \theta \text{ sostituzione chiusa}\}$,

$M_{i+1} = M_i \cup \{B\theta : \theta \text{ sostituzione chiusa tale che } \{A_1\theta, \dots, A_n\theta\} \subseteq M_i, B : -A_1, \dots, A_n \in \mathcal{P}\}$,
allora

$$M_{\mathcal{P}} = \bigcup M_i.$$

Dimostrazione. (1) banale; (2): Se $A : -$ è un fatto di $\mathcal{P}$, allora $\forall \bar{z} A \in P$ e per ogni sostituzione chiusa θ si ha $P \models A\theta$. Per definizione di $M_{\mathcal{P}}$ ne segue $A\theta \in M_{\mathcal{P}}$ e quindi $M_{\mathcal{P}} \models A$, essendo $M_{\mathcal{P}}$ un'interpretazione di Herbrand che rende vere tutte le istanze chiuse di A ; In modo analogo si procede per le regole.

(3) Sia H un modello di Herbrand di $\mathcal{P}$. Dato $A \in M_{\mathcal{P}}$ per definizione si ha $\mathcal{P} \models A$, quindi $H \models A$ e $A \in H$. Ne segue $M_{\mathcal{P}} \subseteq H$.

(4) Dal punto (3) abbiamo che $M_{\mathcal{P}}$ stesso fa parte della famiglia dei modelli di Herbrand di $\mathcal{P}$, quindi $M_{\mathcal{P}} \subseteq \bigcap \{H : H \text{ è un modello di Herbrand di } \mathcal{P}\}$. Per dimostrare l'altra inclusione, basta notare che ogni programma Prolog $\mathcal{P}$ soddisfa la seguente proprietà: l'intersezione di una qualsiasi famiglia di modelli di $\mathcal{P}$ è ancora modello di $\mathcal{P}$ ed è quindi contenuta nel modello minimo.

(5) $M_i \subseteq M_{\mathcal{P}}$ per induzione su i . L'altra inclusione segue dal fatto che $\bigcup M_i$ è un modello di $\mathcal{P}$. $\square$

Corollario 4.2. *Un programma Prolog $\mathcal{P}$ è sempre corretto e completo rispetto al modello minimo $M_{\mathcal{P}}$, cioè, per ogni goal $G = \{\neg A_1, \dots, \neg A_n\}$ (in notazione Prolog G è il goal $:-A_1, \dots, A_n$) si ha*

$$P \cup \{G\} \text{ ha una refutazione} \Leftrightarrow M_{\mathcal{P}} \models \exists \bar{z}(A_1 \wedge \dots \wedge A_n).$$

Dimostrazione. La freccia da sinistra a destra segue perché $M_{\mathcal{P}}$ è modello di $\mathcal{P}$. Viceversa, se $M_{\mathcal{P}} \models \exists \bar{z}(A_1 \wedge \dots \wedge A_n)$, esiste una sostituzione chiusa θ tale che $M_{\mathcal{P}} \models (A_1 \wedge \dots \wedge A_n)\theta$. Ne segue $A_i\theta \in M_{\mathcal{P}}$ per ogni $i = 1, \dots, n$. Dimostriamo che $\mathcal{P} \cup \{G\}$ è insoddisfacibile. Altrimenti, avrebbe un modello I e, per il Lemma 3.3, ne avrebbe uno di Herbrand H . Ma, dalle proprietà del modello minimo seguirebbe $M_{\mathcal{P}} \subseteq H$ e quindi $A_i\theta \in H$ per ogni $i = 1, \dots, n$. Ne seguirebbe

$$H \models \exists \bar{z}(A_1 \wedge \dots \wedge A_n),$$

in contraddizione con $H \models G$.

□

Ad esempio, se

$$\mathcal{P} = \begin{cases} \bar{0} \leq x : - \\ x \leq Sy : - x \leq y \end{cases}$$

abbiamo

$$M_0 = \{0 \leq t : t \in T_c\}, \quad M_1 = M_0,$$

quindi $M_{\mathcal{P}} = M_0 = \{0 \leq t : t \in T_c\}$ e, come si dimostra anche direttamente, gli unici goals chiusi refutabili dal programma sono quelli del tipo

$$: -0 \leq t_1, \dots, 0 \leq t_k.$$

5. REFUTAZIONI LINEARI DA INPUT, RISPOSTE CORRETTE E CALCOLATE

Refutazioni lineari da input: definizione da aggiungere (vedere gli appunti presi a lezione...)

5.1. Ancora variabili...

Definizione 5.1. Una sostituzione σ si dice una risposta calcolata dal programma $\mathcal{P}$ rispetto al goal G se esiste una refutazione lineare da input di $\mathcal{P} \cup \{G\}$ e σ si ottiene componendo tutti gli m.g.u. utilizzati nella refutazione

Ad esempio: dato il goal $: -add(S0, z, SS0)$ e il programma

$$\mathcal{P} = \begin{cases} add(x, \bar{0}, x) : - \\ add(Sx, Sy, SSz) : -add(x, y, z) \end{cases}$$

abbiamo la seguente refutazione:

goal	m.g.u.	program clause
$: -\underline{add(S0, z, SS0)}$		$\underline{add(Sx, Sy, SSu)} : -add(x, y, u)$
	$\sigma_1 = \{x/0, z/Sy, u/0\}$	
$: -\underline{add(0, y, 0)}$		$: -\underline{add(x', \bar{0}, x')} : -$
	$\sigma_2 = \{x'/0, y/0\}$	

□

Componendo i due m.g.u. otteniamo la risposta calcolata $\sigma = \sigma_1\sigma_2 = \{z/s0\}$.

Definizione 5.2. Una sostituzione σ si dice una risposta corretta per il programma $\mathcal{P}$ rispetto al goal $G = \{\neg A_1, \dots, \neg A_n\}$ se

$$\mathcal{P} \models (A_1 \wedge \dots \wedge A_n)\sigma,$$

o, equivalentemente, se

$$M_{\mathcal{P}} \models (A_1 \wedge \dots \wedge A_n)\sigma.$$

Abbiamo:

Teorema 5.1. *Sia $\mathcal{P}$ un programma prolog e $G =: -A_1, \dots, A_n$ un goal.*

Ogni risposta calcolata σ per $\mathcal{P} \cup \{G\}$ è anche corretta.

Inoltre, per ogni risposta corretta η , esiste una refutazione lineare da input di $\mathcal{P} \cup \{G\}$ tale che la corrispondente risposta calcolata σ si istanzia su η (restringendosi alle variabili del goal): esiste una sostituzione τ con $z\sigma\tau = z\eta$, per ogni z variabile di G .

(dimostrazione omessa).

Ad esempio, se

$$P_{\leq} = \begin{cases} p(hx) : - \\ p(fx) : - p(x) \end{cases}$$

e G è il goal : $-p(fx)$, abbiamo che $P \cup \{G\}$ è insoddisfacibile. La sostituzione $\eta_1 = \{x/ha\}$ è allora una sostituzione corretta, ma anche la sostituzione $\eta_2 = \{x/hy\}$ lo è. Inoltre, $P \cup \{G\}$ è refutabile, e la seguente refutazione fornisce la risposta calcolata $\sigma = \sigma_1\sigma_2 = \{x/hz\}$, che si istanzia in entrambe le sostituzioni corrette η_1, η_2 (tramite $\tau_1 = \{z/a\}$ e $\tau_2 = \{z/y\}$, rispettivamente).

goal	m.g.u.	program clause
$: -\underline{p(fx)}$		$\underline{p(fx')} : - p(x')$
	$\sigma_1 = \{x/x'\}$	
$: -\underline{p(x')}$		$\underline{p(hz)} : -$
	$\sigma_2 = \{x'/hz\}$	

□

Altri esempi:

Sia

$$P_{\leq} = \begin{cases} 0 \leq x : - \\ Sx \leq Sy : - x \leq y \end{cases}$$

e G il goal : $-x \leq S0$. Le risposte corrette sono $\theta_1 = \{x/0\}, \theta_2 = \{x/S0\}$.

Se invece consideriamo il goal $G = : -Sx \leq y$, le risposte corrette saranno tutte e sole quelle del tipo:

$$\theta_i = \{x/S^n 0, y/S^{n+1} z\}.$$

Esempio.

Consideriamo il programma

$$P_{add} = \begin{cases} add(x, \bar{0}, x) : - \\ add(x, Sy, Sz) : - add(x, y, z) \end{cases}$$

Sia G il goal : $-add(S^n 0, S^m 0, z)$.

Le risposte corrette sono tutte e sole le sostituzioni $\sigma = \{x/t\}$ tali che $P_{add} \models add(S^n 0, S^m 0, t)$; poiché il modello minimo di P_{add} coincide (a meno di isomorfismi) con il modello $\mathcal{N}$, abbiamo:

$$P_{add} \models add(S^n 0, S^m 0, t) \Leftrightarrow \mathcal{N} \models add(S^n 0, S^m 0, t),$$

esisterà quindi una sola risposta corretta, quella dove $t = S^{n+m} 0$.

Possiamo usare lo stesso programma per sottrarre numeri utilizzando come input il goal $G =: -add(z, S^n 0, S^m 0)$: se $m \geq n$ le risposte corrette sono i termini t per cui $P_{add} \models add(t, S^n 0, S^m 0)$ e questo è equivalente a $\mathcal{N} \models add(t, S^n 0, S^m 0)$; quindi l'unica risposta corretta è quella con $t = S^{n-m} 0$.

6. ESERCIZI

- (1) Considera il linguaggio $L = \{a, b, f, p, r\}$, dove a, b sono costanti, f è un simbolo funzionale unario e p ed r sono simboli relazionali unari e binari, rispettivamente.
- i) Descrivi l'insieme delle formule atomiche chiuse di L (tale insieme si chiama *base di Herbrand* e si denota con B_L).
 - ii) Per ogni enunciato F e interpretazione di Herbrand H_i seguente, stabilire se l'interpretazione soddisfa l'enunciato.

$$F = \forall x(pfx \rightarrow px) \wedge \exists xpx,$$

$$H_1 = \{pfa\}, H_2 = \emptyset, H_3 = \{pf^n a : n \geq n_0\}, \text{ per un fissato } n_0 \in \mathbb{N};$$

iii)

$$F = \exists x \exists y r(x, y) \wedge \forall x \forall y (r(x, y) \rightarrow r(x, fy)),$$

$$H_1 = \emptyset, \quad H_2 = \{r(fa, fa), r(a, a)\}, \quad H_3 = \{r(fa, fa), r(a, fa)\}.$$

- iv) Dimostra che gli enunciati seguenti sono soddisfacibili ma non hanno modelli di Herbrand.

$$F = \exists x \neg rx \wedge ra \wedge \forall x (rx \rightarrow rfx),$$

$$G = \exists x \exists y r(x, y) \wedge \neg r(a, a) \wedge \forall x \forall y (r(x, fy) \rightarrow r(x, y)) \wedge \forall x \forall y (r(x, y) \rightarrow r(y, x)).$$

- v) i) Dimostra che se M è un modello di Herbrand dell'enunciato $\forall x p(x)$, dove p è un simbolo predicativo unario e N è un sottoinsieme della base di Herbrand che contiene M , allora N rende vera $\forall x p(x)$.
- ii) La stessa proprietà vale se sostituiamo al posto della formula atomica $p(x)$ una formula $F(x)$ qualsiasi?

- vi) Sia $\mathcal{P}$ un programma e A_1, A_2 formule atomiche chiuse del linguaggio. Dimostra che

$$\mathcal{P} \models A_1 \vee A_2 \quad \Rightarrow \quad \mathcal{P} \models A_1 \text{ oppure } \mathcal{P} \models A_2.$$

(Suggerimento: usa il modello minimo di $\mathcal{P}$).

- vii) È vero che per ogni enunciato universale soddisfacibile F di un dato linguaggio al prim'ordine esiste sempre un programma PROLOG P (cioè, un insieme finito di fatti e regole) logicamente equivalente a F ?