UNIVERSITÀ DEGLI STUDI DI MILANO

Dipartimento di Scienze dell'Informazione

Dispensa ad uso dei corsi di Logica per le Lauree Specialistiche

Risoluzione e Sovrapposizione

Silvio Ghilardi

Questi appunti sono destinati ad un uso didattico per studenti delle Lauree Specialistiche di classe Informatica. Essi presuppongono la lettura preventiva della dispensa sulla logica proposizionale adottata negli stessi corsi.

La preparazione dell'esame comprende anche la padronanza di un opportuno supporto software; fra i vari dimostratori automatici basati su metodi di saturazione, consigliamo il sistema SPASS disponibile all'indirizzo:

http://www.spass-prover.org/

In questi appunti cercheremo di indicare nelle note alcune informazioni utili per acquisire familiarità con il sistema, anche se la consultazione della documentazione inclusa nella distribuzione deve essere considerata un punto di riferimento imprescindibile per la preparazione. Ampie collezioni di problemi già formalizzati da sottoporre al dimostratore automatico sono disponibili in rete e sono accessibili anche dal sito stesso di SPASS.

La presente versione di questa dispensa porta la data del **12 settembre 2007**.

Indice

1 Preliminari insiemistici

Riassumiamo brevemente alcune nozioni di teoria ingenua degli insiemi, che dovrebbero essere note da corsi di base di materie matematiche.[1]

La nozione di **insieme** viene data per intuitiva e inanalizzata (in un corso specifico di teoria degli insiemi si vedrà eventualmente qualche sistema assiomatico che la tratta in maniera più rigorosa). Similmente, diamo per intuitiva la nozione di **funzione** $f : X \longrightarrow Y$ fra gli insiemi X e Y: la f viene genericamente definita come una 'legge' che ad ogni elemento di X (detto insieme dominio) fa corrispondere uno ed un solo elemento di Y (detto insieme codominio). Raddoppiare un numero per esempio è una funzione dall'insieme $\mathbb{N}$ dei numeri naturali in sè. Considerare la madre, è una funzione dall'insieme degli esseri umani nell'insieme degli esseri umani di sesso femminile. Invece, considerare i figli non è una funzione dall'insieme degli esseri umani verso l'insieme degli esseri umani per ben due motivi: non tutti gli esseri umani hanno figli e non tutti ne hanno uno solo, sicchè viene violato il tratto distintivo della nozione di funzione che, ripetiamo, consiste nell'essere una corrispondenza sempre definita e definita in modo univoco.

La nozione di **prodotto cartesiano** di n-insiemi $X_1, \ldots, X_n$ è la seguente: il prodotto cartesiano, che scriviamo con

$$X_1 \times \cdots \times X_n$$

è l'insieme delle liste $\langle a_1, \ldots, a_n \rangle$ di n elementi (dette n-ple) di elementi appartenenti, rispettivamente, ad $X_1, X_2, \ldots, X_n$. Così ad esempio, l'insieme dei vestiti interi spezzati è il prodotto cartesiano dell'insieme delle giacche e dell'insieme dei pantaloni; un altro esempio è fornito dai punti del piano che si possono identificare con le coppie di punti presi uno dalla retta delle ascisse e uno dalla retta delle ordinate.

Quando $X_1, \ldots, X_n$ concidono tutti con uno stesso insieme X, usiamo la notazione X^n per indicare il relativo prodotto cartesiano, detto anche **potenza** n-esima dell'insieme X. Per $n = 1$, X^1 è X stesso; per $n = 0$, è utile identificare la potenza 0-esima X^0 di X con l'insieme $\mathbf{1}$ definito come un qualsiasi insieme con un elemento solo ($\mathbf{1}$ è chiamato 'insieme terminale' e, se si preferisce, lo si può definire esplicitamente mediante $\mathbf{1} := \{*\}$). Si noti che una funzione

$$c : \mathbf{1} \longrightarrow X$$

[1] Questo riassunto é strettamente finalizzato ad introdurre nozioni che verranno utilizzate nei paragrafi seguenti.

è univocamente specificata una volta noto l'elemento $c(*) \in X$; quindi le funzioni $\mathbf{1} \longrightarrow X$ possono a buon diritto essere confuse con gli elementi di X stesso.

Un **sottoinsieme** di un insieme X è un insieme composto da alcuni (magari nessuno, magari tutti) degli elementi di X.

L'insieme delle **parti** $\mathcal{P}(X)$ di un insieme X è l'insieme di tutti i sottoinsiemi di X. Ad esempio, $\mathcal{P}(\mathbf{1})$ consta dei due elementi

$$\emptyset, \qquad \{*\}.$$

Ci tornerà utile identificare questi due sottoinsiemi con i due valori di verità, rispettivamente 'falso' e 'vero', che abbiamo già incontrato studiando la logica proposizionale.

Una **relazione** n-aria su un insieme X è un sottoinsieme della potenza n-esima di X (per $n = 1$, dunque, una relazione unaria è semplicemente un sottoinsieme di X e, per $n = 0$, una relazione n-aria su X è un valore di verità). Ad esempio, la relazione 'essere amico di' è la relazione che contiene le coppie di esseri umani che sono amici tra di loro; 'essere minore di' è l'ovvia relazione fra numeri naturali che tutti conosciamo (tale relazione conterrà ad esempio le coppie $\langle 2, 3\rangle, \langle 12, 45\rangle, \ldots$, ma non la coppia $\langle 3, 2\rangle$).

2 Linguaggi elementari

2.1 Motivazioni

Un primo modo per addentrarsi nelle tematiche della logica consiste nel porsi il problema della correttezza delle inferenze (ossia degli 'schemi di ragionamento'). Ad esempio, l'inferenza

$$\frac{\begin{array}{l}\text{Se piove, prendo l'ombrello.}\\ \text{Non prendo l'ombrello.}\end{array}}{\text{Quindi non piove.}}$$

è corretta. Per provarlo, associamo all'enunciato 'Piove' la lettera proposizionale p, all'enunciato 'Prendo l'ombrello' la lettera proposizionale q, sicchè l'inferenza viene schematizzata con

$$\frac{\begin{array}{l}p \to q\\ \neg q\end{array}}{\neg p}$$

Per rilevare la correttezza di tale inferenza, basta ora osservare che la formula

$$(p \to q) \wedge \neg q \to \neg p \tag{1}$$

è una tautologia. La formula (1) è stata ottenuta considerando l'implicazione che ha come antecedente la congiunzione delle premesse dell'inferenza da testare e come conseguente la sua conclusione.

Tuttavia vi sono inferenze intuitivamente corrette che non possono essere studiate e giustificate mediante il calcolo proposizionale. Consideriamo ad esempio la seguente inferenza:

Tutti gli uomini sono mortali.
Socrate è un uomo.

Quindi Socrate è mortale.

È facile vedere che questa inferenza, pur essendo intuitivamente corretta, non è giustificabile mediante il calcolo proposizionale. Infatti riscrivendo l'inferenza mediante il simbolismo della logica proposizionale otteniamo

$$\begin{array}{l} p \\ q \\ \hline r \end{array}$$

Condizione necessaria e sufficiente affinchè questa inferenza sia corretta è che la formula

$$p \wedge q \to r$$

sia una tautologia. Tuttavia è facile osservare l'assegnamento V tale che $V(p) = V(q) = 1$ e $V(r) = 0$ la falsifica. Dunque l'inferenza data, intuitivamente corretta, risulta formalmente scorretta se utilizziamo come strumento di indagine il calcolo proposizionale.

In effetti la correttezza delle inferenze non si basa solamente sulle relazioni vero funzionali tra le proposizioni di cui sono composte (cosa che invece succede nelle inferenze giustificabili mediante il solo calcolo proposizionale), ma può basarsi anche sulla struttura interna di queste proposizioni e sul significato di espressioni quali, ad esempio, "ogni", "tutti", ecc. In altre parole, per poter trattare inferenze come l'ultima che abbiamo visto è necessario considerare più a fondo la struttura interna delle proposizioni.

Occorre quindi introdurre un linguaggio più ricco e con maggiore capacità espressiva del calcolo proposizionale. Più precisamente, dovremo utilizzare un nuovo formalismo e

introdurre poi delle regole che permettano di trattare formule contenenti i nuovi simboli introdotti.

Vediamo brevemente e in modo molto informale da che cosa è costituito un linguaggio del primo ordine.

Innanzitutto ampliamo l'insieme degli operatori logici. Ai connettivi $\wedge, \vee \rightarrow, \neg$ aggiungiamo due nuovi operatori $\forall$ e $\exists$. Questi due nuovi operatori sono di natura sensibilmente differente dai connettivi proposizionali, si dicono *quantificatori*. Il simbolo $\forall$ si chiama *quantificatore universale* e il suo significato intuitivo è "per ogni", Il simbolo $\exists$ si chiama *quantificatore esistenziale* e il suo significato intuitivo è "esiste".

Avremo poi, in particolare, due classi distinte di enti linguistici: le "costanti" e i "predicati". Le costanti rappresenteranno elementi del dominio del discorso (numeri, persone, oggetti, etc) mentre i "predicati" rappresenteranno le relazioni che possono intercorrere tra questi oggetti (esempi di predicati sono: "essere un uomo", "essere minore di", etc.). In altre parole, i predicati ci consentono di esprimere proprietà e relazioni su insiemi di oggetti.

Dunque se ad esempio $P(x)$ indica che x gode di una data proprietà P allora l'espressione $\forall x P(x)$ significherà "per ogni x vale la proprietà P" o, equivalentemente, "la proprietà P vale per ogni x", mentre l'espressione $\exists x P(x)$ significherà "esiste un x che gode della proprietà P".
Come esempio riconsideriamo l'inferenza

$$\begin{array}{l} \text{Tutti gli uomini sono mortali.} \\ \text{Socrate è un uomo.} \\ \hline \text{Quindi Socrate è mortale.} \end{array}$$

Supponiamo che

- U stia per il predicato "essere uomo"

- M stia per il predicato "essere mortale"

- la costante a stia per "Socrate"

Possiamo quindi formalizzare la nostra inferenza con

$$\begin{array}{l} \forall x(U(x) \rightarrow M(x)) \\ U(a) \\ \hline M(a) \end{array}$$

Abbiamo in questo modo formalizzato l'inferenza in maniera tale che sarà poi possibile, mediante opportune tecniche, analizzarla e studiarne la correttezza.

Più in generale, nel linguaggio del primo ordine avremo i seguenti simboli: i connettivi proposizionali e i due quantificatori, le *variabili individuali* $x_0, x_1 \ldots$, le *costanti individuali* $a_1, a_2, \ldots$, le *lettere predicative* $P, Q \ldots$ e le *lettere funzionali* $f, g \ldots$ (in realtà, consideremo le costanti individuali come simboli di funzione 0-arie).

Nel nostro esempio abbiamo utilizzato una sola costante a, una sola variabile individuale x, due predicati unari U e M, e nessuna lettera funzionale.

2.2 Segnature, Termini e Formule

Dunque un linguaggio del primo ordine (che chiameremo anche 'linguaggio elementare') è più ricco di un linguaggio proposizionale e consente di nominare individui, di costruire designazioni di individui a partire da altre, di parlare di proprietà di individui, di quantificare su di essi, ecc. Diamo ora la relativa definizione formale:

Definizione 2.1. *Un* **linguaggio elementare** *(o segnatura)* $\mathcal{L}$ *è una quadrupla*

$$\langle \mathcal{P}, \mathcal{F}, \alpha, \beta \rangle,$$

dove

- $\mathcal{P}$ *è un insieme (detto insieme dei simboli di predicato) e* $\alpha : \mathcal{P} \longrightarrow \mathbb{N}$ *è una funzione a valori nei numeri naturali (se* $\alpha(R) = n$ *diciamo che* R *è un* simbolo di predicato di arietà n*);*

- $\mathcal{F}$ *è un insieme (detto insieme dei simboli di funzione) e* $\beta : \mathcal{F} \longrightarrow \mathbb{N}$ *è una funzione a valori nei numeri naturali (se* $\beta(f) = n$ *diciamo che* f *è un* simbolo di funzione di arietà n*).*

I simboli di predicato di arietà 0 sono le vecchie lettere proposizionali: essi possono essere utilizzati per formalizzare frasi costituite da soli verbi impersonali (come 'Piove', 'Nevica'). I simboli di predicato di arietà 1 rappresentano proprietà di individui: essi possono essere utilizzati per modellizzare nomi comuni, aggettivi qualificativi, nonchè verbi intransitivi (con questo intendiamo dire che 'uomo' rappresenta la proprietà di essere uomo, 'bello' la proprietà di essere bello, 'dormire' la proprietà di essere addormentato, ecc). I

simboli di predicato di arietà 2 rappresentano relazioni fra coppie di individui: essi possono essere utilizzati per modellizzare verbi transitivi (ad esempio, 'amare' rappresenta l'insieme delle coppie di esseri umani il cui primo componente è innamorato del secondo). I simboli di predicato di arietà 3 rappresentano relazioni fra terne di individui: essi possono essere utilizzati per modellizzare verbi dativi come 'dare', 'regalare', ecc.[2]

I simboli di funzione di arietà zero sono detti costanti individuali e possono essere utilizzati per modellizzare i nomi propri, nonchè costanti matematiche (come $0, 1, \pi, e, \dots$). I simboli di funzione di arietà 2 rappresentano operazioni binarie (come ad esempio le operazioni aritmetiche di somma e prodotto); naturalmente i simboli di funzione di arietà 1 rappresentano operazioni unarie (come il seno, l'esponenziale in una base fissata, il logaritmo, ecc.). Simboli di funzione di arietà 1 sono presenti anche nel linguaggio naturale, dove possono essere utilizzati per rappresentare le funzioni espresse da certe locuzioni come 'il padre di$\cdots$', 'il professore di$\cdots$'.

Simboli di funzione e di predicato di arietà elevata (maggiore di 3, per intenderci) sono raramente usati, ma li abbiamo inclusi nella definizione di segnatura per omogeneità di trattazione.

In aggiunta ai simboli di predicato e di funzione, per costruire le formule avremo a disposizione dei simboli universali (cioè comuni ad ogni linguaggio elementare), che sono, oltre ai simboli ausiliari (parentesi e vigole):

- l'insieme $\mathcal{V} = \{x_0, x_1, x_2, \dots, y_0, y_1, y_2, \dots\}$ detto insieme delle **variabili individuali**;
- i **connettivi proposizionali** $\wedge, \vee, \rightarrow, \neg$ che abbiamo incontrato nella parte del corso relativa alla logica proposizionale;
- i **quantificatori** $\forall$ (per ogni) ed $\exists$ (esiste).

Se $\mathcal{P}$ contiene il predicato $=$ di arietà 2, diciamo che $\mathcal{L}$ è un linguaggio con identità.

Le variabili individuali rappresentano individui indeterminati, la cui designazione può ad esempio essere fissata dal contesto (in tal senso, si può costruire un parallelo fra le variabili dei linguaggi formali e i pronomi dei linguaggi naturali).

Per riferirci ad individui, abbiamo ora a disposizione costanti e variabili: se applichiamo a queste ultime i simboli di funzione, possiamo costruire designatori più complessi. In tal

[2]Per un'analisi meno schematica e più approfondita di queste classificazioni occorre consultare un testo sulla semantica dei linguaggi naturali.

modo si costruiscono espressione matematiche come $\pi + log\,(x+y)$, ma anche locuzioni del linguaggio naturale come 'padre di Carlo', 'padre del padre di Carlo' (ossia 'nonno paterno di Carlo'), ecc. Tutto questo è riassunto nella definizione formale di termine:

Definizione 2.2. *Data una segnatura $\mathcal{L}$, l'insieme degli $\mathcal{L}$-termini (o più semplicemente* **termini***) è cosí definito:*

- *ogni $c \in \mathcal{F}$ con $\beta(c) = 0$ è un termine;*
- *ogni $x \in \mathcal{V}$ è un termine;*
- *se $f \in \mathcal{F}$, se $\beta(f) = n$ (per $n \geq 1$) e se $t_1, \ldots, t_n$ sono termini, $f(t_1, \ldots, t_n)$ è un termine.*[3]

Abbiamo visto che i termini aumentano le possibilità di un linguaggio elementare di riferirsi ad individui (possibilità inizialmente ristretta a costanti e variabili). In modo simile, la nozione di formula offre la possibilità di costruire relazioni e affermazioni complesse:

Definizione 2.3. *Data una segnatura $\mathcal{L}$, l'insieme delle $\mathcal{L}$-formule (o più semplicemente* **formule***) è cosí definito:*

- *se $R \in \mathcal{P}$, se $\alpha(R) = n$ e se $t_1, \ldots, t_n$ sono termini, $(R(t_1, \ldots, t_n))$ è una formula;*[4]
- *se A_1, A_2 sono formule, tali sono anche $(A_1 \wedge A_2)$, $(A_1 \vee A_2)$, $(A_1 \rightarrow A_2)$, $(\neg A_1)$;*
- *se A è una formula e $x \in \mathcal{V}$, allora $(\forall x A)$ e $(\exists x A)$ sono formule.*

La scelta di un opportuno linguaggio dipende da ciò di cui vogliamo parlare e da ciò che intendiamo realmente esprimere. Ad esempio, se vogliamo parlare di numeri (naturali, interi, razionali o reali) un opportuno linguaggio $\mathcal{L}_1$ potrebbe contenere: a) due costanti $0, 1$; b) due simboli di operazioni binarie, cioè la somma $+$ e il prodotto $\cdot$; c) due relazioni binarie, cioè l'identità $=$ e la relazione di minore $<$. Se invece vogliamo parlare di relazioni di parentela, una scelta ragionevole potrebbe essere il linguaggio $\mathcal{L}_2$ comprendente: a) due

[3] Si osservi che, nel caso dei simboli di funzione binaria, questa definizione obbliga ad adottare una notazione prefissa (cioè obbliga a scrivere ad esempio $+(x, y)$ anzichè $x + y$). Tuttavia, noi ci sentiremo liberi di usare notazioni infisse nei casi in cui lo riterremo opportuno.

[4] Formule di questo tipo, cioè formule ottenute applicando un simbolo di predicato a termini, si dicono formule *atomiche*. Le formule atomiche non contengono quindi altre formule come parti proprie.

simboli di funzione unarie, cioè p ='il padre di' e m ='la madre di'; b) un predicato binario, l'identità.

Come sappiamo, i termini servono a costruire designazioni di individui complesse, partendo dalle designazioni di base specificate da costanti e variabili. Nel caso di $\mathcal{L}_2$, i termini

$$p(p(x)), \qquad p(m(x))$$

servono a specificare il nonno paterno e il nonno materno dell'individuo x. Analogamente, le formule costruiscono proprietà e relazioni complesse. Nel caso di $\mathcal{L}_2$, la formula

$$p(x) = p(y) \wedge m(x) = m(y)$$

dice che x e y sono fratelli/sorelle. La formula

$$(p(p(x))=p(p(y))) \vee (p(p(x))=p(m(y))) \vee (p(m(x))=p(p(y))) \vee (p(m(x))=p(m(y)))$$

è un modo per esprimere che x e y sono cugini (tramite la comunanza di uno dei nonni maschi).

Convenzioni notazionali:

- Valgono le solite convenzioni per eliminare le parentesi, ossia le parentesi più esterne vengono di regola omesse. In più stipuliamo anche che $\neg, \forall, \exists$ legano più strettamente di $\wedge, \vee$, che a loro volta legano più strettamente di $\rightarrow$.[5]

- Un termine è *chiuso* (o '**ground**') se è costruito senza usare variabili.

- Una occorrenza di una variabile x in una formula A è detta **vincolata** qualora si trovi all'interno di una sottoformula[6] del tipo $\forall xB$ o $\exists xB$, altrimenti è detta **libera**. Ad esempio, nelle formula

  $$\forall x(R(x,y)) \vee P(x)$$

  la x ha due occorrenze, la prima delle quali è vincolata (perchè è all'interno della sottoformula $\forall x(R(x,y))$), mentre la seconda è libera; la y ha una sola occorrenza, che è libera.

- Una variabile x occorre libera in A se e solo se qualche sua occorrenza in A è libera.

[5] Così, ad esempio, $\forall xP(x) \rightarrow \exists zP(z) \vee \exists yQ(y)$ sta per $(\forall x(P(x))) \rightarrow ((\exists z(P(z))) \vee (\exists y(Q(y))))$.

[6] La nozione di sottoformula è quella ovvia (volendo essere precisi, si potrebbe definirla induttivamente).

- Un **enunciato** ('sentence', in inglese) o formula chiusa è una formula in cui nessuna variabile occorre libera.

- Con notazioni del tipo $A(t/x)$ (o, più semplicemente $A(t)$) indicheremo il risultato dell'operazione di sostituzione del termine t al posto di tutte le occorrenze libere della variabile x in A.

3 La Semantica di Tarski

Diamo ora la semantica per i linguaggi elementari. Il problema della semantica è presto detto: simboli di funzione, simboli di predicato, termini e formule di per sè non sono altro che caratteri e stringhe di caratteri e in quanto tali non hanno nessun significato intrinseco. È ben vero che nel definire termini e formule abbiamo tenuto presente che i termini devono rappresentare individui e che le formule devono rappresentare relazioni complesse o affermazioni, tuttavia queste motivazioni intuitive non sono sufficienti ad attribuire loro un significato pieno. In altri termini, il fatto che il dizionario riporti la parola 'cane' come parola dotata di significato, non significa ancora che tale significato sia noto o compreso, non prima almeno di aver letto cosa la parola 'cane' significhi o di averlo imparato per altra via da piccoli.

La semantica dei linguaggi elementari è stata rigorosamente fissata dal logico polacco Tarski negli anni 30; tuttavia la definizione tarskiana non fa che riprendere la millenaria tradizione filosofica della definizione di verità come corrispondenza con lo stato di fatto. All'interno di un corso di logica, la definizione tarskiana di verità rappresenta un passaggio imprescindibile, perchè è solo su di essa che si possono fondare tutti i procedimenti algoritmici che vengono poi introdotti. Ciononostante, l'impatto con la semantica formalizzata può risultare difficile di primo acchito per un semplice motivo: i significati delle parole e delle locuzioni che usiamo nella vita quotidiana (e quindi anche la nozione di verità che su di essi si basa) sono fissati dalle convenzioni linguistiche e sociali, per cui non si sente il bisogno di fare un passo indietro e di riesaminarli da un punto di vista astratto. Tuttavia tale riesame è per noi indispensabile per proseguire; invitiamo lo studente perciò a leggere senza troppe pretese il presente paragrafo in prima battuta e a ritornare (magari più volte) su di esso quando la sua comprensione degli argomenti si sarà affinata.

3.1 La Nozione di Struttura

La prima cosa da tenere presente è che il significato dei simboli dipende da una situazione concreta. Nel caso della logica proposizionale, una situazione concreta è modellata da un assegnamento: l'assegnamento fa sì ad esempio che l'enuciato 'Piove' venga a denotare un significato (il vero od il falso) a seconda appunto della situazione che l'assegnamento simula in modo schematico, cioè a seconda che realmente piova o meno. Nella logica predicativa, occorrerà la nozione più complessa di *struttura* invece della nozione di assegnamento: una struttura deve assegnare un insieme di oggetti ad ogni simbolo di predicato unario, una relazione (vista come insieme di coppie di oggetti) ad ogni simbolo di predicato binario, un oggetto ad ogni costante, ecc.[7] In altre parole, la nozione di struttura fotografa e schematizza (mediante strumenti insiemistici) quanto è noto sul significato degli enti linguistici in una data situazione.

Per introdurre formalmente una struttura, si fissa un insieme non vuoto $\mathbf{A}$, detto dominio della struttura, sul quale assumeranno i valori le variabili individuali; poi si fissa una funzione che associ ad ogni simbolo di funzione n-ario una funzione da $\mathbf{A}^n$ a $\mathbf{A}$ (cioè una operazione a n posti) e ad ogni simbolo di predicato n-ario una relazione n-aria (cioè un sottoinsieme di $\mathbf{A}^n$). Tutto questo è scritto nella seguente definizione:

Definizione 3.1. *Data una segnatura $\mathcal{L}$, una* **$\mathcal{L}$-struttura** *$\mathcal{A}$ è una coppia $\langle \mathbf{A}, \mathcal{I} \rangle$ dove $\mathbf{A}$ è un insieme (non vuoto),*[8] *detto* **dominio**, *e $\mathcal{I}$ è una funzione, detta* **interpretazione**, *che opera come specificato qui di seguito. $\mathcal{I}$ associa*

- *ad ogni $P \in \mathcal{P}$ tale che $\alpha(P) = n$, un sottoinsieme $\mathcal{I}(P)$ di $\mathbf{A}^n$.*
- *ad ogni $f \in \mathcal{F}$ tale che $\beta(f) = n$, una funzione $\mathcal{I}(f) : \mathbf{A}^n \longrightarrow \mathbf{A}$.* ⊣

In particolare, se $c \in \mathcal{F}_0$, $\mathcal{I}(c)$ è un elemento di $\mathbf{A}$ e se $P \in \mathcal{P}_0$, $\mathcal{I}(P)$ è un valore di verità (si ricordi quanto convenuto nel paragrafo 1). Se $\mathcal{L}$ contiene l'identità, stipuliamo che $\mathcal{I}(=)$ sia sempre l'insieme delle coppie identiche, cioè $\{(a, a) \mid a \in \mathbf{A}\}$.

[7] Per farsi un'idea precisa, può essere utile pensare ai nomi propri di persona, come 'Pietro', 'Paolo', ecc.: questi ultimi non hanno un significato universale fissato apriori (come possiamo essere erroneamente tentati di credere che succeda per altre entità linguistiche come nomi comuni e verbi), il loro significato è di volta in volta adattato dalla nostra mente alla situazione in cui ci troviamo.

[8] Si faccia attenzione al fatto che usiamo la lettera calligrafica $\mathcal{A}$ per indicare una struttura nel suo complesso e la lettera in grassetto $\mathbf{A}$ per indicarne il dominio (invece la lettera stampata maiuscola A, come le lettere $B, C, \ldots$, continueranno ad essere usate come metavariabili per formule). Questi aspetti notazionali non vanno trascurati, pena cadere in confusione completa!

3.2 La Nozione di Verità

Fissata un'interpretazione (cioè una $\mathcal{L}$-struttura $\mathcal{A}$) per i simboli di un linguaggio $\mathcal{L}$, è possibile dire quali enunciati di $\mathcal{L}$ sono veri (in $\mathcal{A}$) e quali no. Ciò rispecchia una pratica intuitiva: possiamo dire se 'Paolo è simpatico' è vero o no, una volta che ci siamo intesi sul significato delle parole, cioè una volta che abbiamo fissato una $\mathcal{L}$-struttura. Avendo fissato una $\mathcal{L}$-struttura, sappiamo chi è 'Paolo', sappiamo quali sono i nostri criteri di simpatia perchè abbiamo fissato l'insieme delle persone simpatiche, per cui per stabilire il valore di verità della frase 'Paolo è simpatico' si tratta solo di vedere se Paolo appartiene o meno a tale insieme delle persone simpatiche. Più in generale, avremo che una formula del tipo $P(c_1, \ldots, c_n)$ (dove $c_1, \ldots, c_n$ sono costanti) sarà vera nella $\mathcal{L}$-struttura $\mathcal{A}$ sse la n-pla $(\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n))$ (che fissa gli individui denotati da $c_1, \ldots, c_n$ in $\mathcal{A}$) appartiene a $\mathcal{I}(P)$ (ossia all'insieme delle n-ple che fissa il significato della relazione n-aria P in $\mathcal{A}$).

Per definire il valore di verità di formule non atomiche avremo delle ovvie clausole ricorsive. Tuttavia, nel dare tale definizione, si incontrano alcuni problemi che necessitano l'introduzione di qualche accorgimento tecnico, dovuto al fatto che non tutti gli elementi di $\mathbf{A}$ sono nominabili con termini chiusi di $\mathcal{L}$. Il problema può essere evidenziato nel modo seguente. La relazione $\mathcal{A} \models A$ ('l'enunciato A è vero nella $\mathcal{L}$-struttura $\mathcal{A}$') verrà, come si è detto, definita per induzione sul numero di connettivi e quantificatori di A; avremo ad esempio clausole ricorsive del tipo[9]

$$\mathcal{A} \models A_1 \wedge A_2 \qquad \text{sse} \qquad (\mathcal{A} \models A_1 \quad \text{e} \quad \mathcal{A} \models A_2)$$

che dicono che una congiunzione è vera sse lo sono entrambi i congiunti. Tuttavia, nel caso dei quantificatori, tali clausole non possono essere formulate ingenuamente con

$$\mathcal{A} \models \exists x A \qquad \text{sse} \qquad \mathcal{A} \models A(a/x), \text{ per qualche } \ a \in \mathbf{A}$$

perchè $A(a/x)$ non è una $\mathcal{L}$-formula, in quanto a è un elemento del dominio di interpretazione e non un simbolo del linguaggio (per cui la scrittura $A(a/x)$ semplicemente non ha senso). Se tentiamo di risolvere questo problema, ci accorgiamo subito inoltre che non c'è nessuna motivo affinchè il linguaggio $\mathcal{L}$ abbia a disposizione un nome per ogni elemento $a \in \mathbf{A}$.[10] Se avessimo a disposizione un nome $\bar{a}$ per ogni $a \in \mathbf{A}$, la clausola di verità per il

[9] Qui e nel seguito, abbreviamo 'se e solo se' con 'sse'.

[10] Si pensi soltanto al fatto seguente: il linguaggio usualmente è numerabile (cioè le formule e i termini possono di solito essere messi in corrispondenza biunivoca con i numeri naturali), mentre $\mathbf{A}$ può essere un insieme qualunque molto più grande, come quello dei numeri reali.

quantificatore esistenziale potrebbe essere agevolmente corretta con

$$\mathcal{A} \models \exists x A \qquad \text{sse} \qquad \mathcal{A} \models A(\bar{a}/x), \text{ per qualche } a \in \mathbf{A}.$$

Per questo motivo, decidiamo di ampliare preventivamente $\mathcal{L}$ stesso. Se $\mathcal{A}$ è una $\mathcal{L}$-struttura, $\mathcal{L}_{\mathbf{A}}$ indica il linguaggio ottenuto aggiungendo a $\mathcal{L}$ una costante $\bar{a}$ per ogni $a \in \mathbf{A}$ ($\bar{a}$ è detta essere il nome di a). Cosí $\mathcal{L}_{\mathbf{A}}$ contiene un nome per ogni elemento del dominio di $\mathcal{A}$. Allarghiamo $\mathcal{I}$ a queste nuove costanti ponendo $\mathcal{I}(\bar{a}) = a$ (in futuro, se non c'è pericolo di confusione, ometteremo spesso di distinguere fra elementi di $\mathbf{A}$ e i loro nomi, cioè scriveremo direttamente a invece di $\bar{a}$).

C'è ancora un punto da chiarire. L'interpretazione $\mathcal{I}$ di una $\mathcal{L}$-struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ fissa il significato delle costanti (e dei simboli di funzione), ma non fissa direttamente il significato dei termini composti. Ad esempio, la $\mathcal{L}$-struttura $\mathcal{A}$ fissa il significato della costante $c =$'Paolo', fissa il significato delle funzioni unarie $p =$'padre di' e $m =$'madre di'; possiamo da tutto ciò risalire al significato dell'espressione 'nonno materno di Paolo'? Certamente, tale espressione 'nonno materno di Paolo' altri non è che $p(m(c))$ e il suo significato sarà $\mathcal{I}(p)(\mathcal{I}(m)(\mathcal{I}(c)))$ (ossia il valore della funzione che interpreta 'padre di' calcolato sul valore della funzione che interpreta 'madre di' calcolato sull'elemento denotato da 'Paolo').

Formalmente, si procede così: per induzione, *definiamo* $\mathcal{I}(t)$ *per ogni* $\mathcal{L}_{\mathbf{A}}$*-termine ground* t. Se t è una costante (vecchia o nuova) $\mathcal{I}(t)$ è già stato definito. Se t è $f(t_1, \ldots, t_n)$, $\mathcal{I}(t)$ sarà $\mathcal{I}(f)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$ (dove $\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n)$ sono dati per induzione e $\mathcal{I}(f)$ è l'interpretazione del simbolo f specificata dalla $\mathcal{L}$-struttura $\mathcal{A} = (\mathbf{A}, \mathcal{I})$).

Siamo ora in grado di dare le definizione di verità di una $\mathcal{L}$-formula in una $\mathcal{L}$-struttura:

Definizione 3.2. *Data una* $\mathcal{L}$*-struttura* $\mathcal{A}$ *e dato un* $\mathcal{L}_{\mathbf{A}}$*-enunciato* A*, la relazione* $\mathcal{A} \models A$ *(che si legge con '*A *è* **vero** *in* $\mathcal{A}$*), è definita induttivamente sul numero di connettivi e*

quantificatori di A come segue:

$$\begin{array}{rcl}
\mathcal{A} \models P(t_1, \ldots, t_n) & \text{sse} & (\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n)) \in \mathcal{I}(P) \\
\mathcal{A} \models A_1 \wedge A_2 & \text{sse} & (\mathcal{A} \models A_1 \quad \text{e} \quad \mathcal{A} \models A_2) \\
\mathcal{A} \models A_1 \vee A_2 & \text{sse} & (\mathcal{A} \models A_1 \quad \text{oppure} \quad \mathcal{A} \models A_2) \\
\mathcal{A} \models \neg A_1 & \text{sse} & \mathcal{A} \not\models A_1 \\
\mathcal{A} \models A_1 \rightarrow A_2 & \text{sse} & (\mathcal{A} \not\models A_1 \quad \text{oppure} \quad \mathcal{A} \models A_2) \\
\mathcal{A} \models \forall x A_1 & \text{sse} & \mathcal{A} \models A_1(\bar{a}/x) \quad \text{per ogni } a \in \mathbf{A} \\
\mathcal{A} \models \exists x A_1 & \text{sse} & \mathcal{A} \models A_1(\bar{a}/x) \quad \text{per qualche } a \in \mathbf{A}.
\end{array}$$

Se A è una formula qualunque (non necessariamente un enunciato) $\mathcal{A} \models A$ sta per $\mathcal{A} \models \forall x_1 \cdots \forall x_n A$ (dove abbiamo assunto che $x_1, \ldots, x_n$ siano tutte e sole le variabili che occorrono libere in A).

Si noti che nel caso in cui $\mathcal{L}$ contenga l'identità abbiamo sempre

$$\mathcal{A} \models t_1 = t_2 \quad \text{sse} \quad \mathcal{I}(t_1) = \mathcal{I}(t_2)$$

per ogni coppia t_1, t_2 di $\mathcal{L}_{\mathbf{A}}$-termini chiusi.

La seguente importante definizione completa il quadro della semantica della logica elementare. Chiamiamo **$\mathcal{L}$-teoria** $\mathcal{T}$ un qualsiasi insieme di enunciati di $\mathcal{L}$ (le formule appartenenti a $\mathcal{T}$ saranno dette assiomi di $\mathcal{T}$).

Definizione 3.3. *Se $\mathcal{T}$ è una $\mathcal{L}$-teoria, si dice che $\mathcal{A}$ è* **modello** *di $\mathcal{T}$ (in simboli $\mathcal{A} \models \mathcal{T}$) qualora $\mathcal{A} \models A$ valga per ogni formula A appartenente a $\mathcal{T}$. Si dice che $\mathcal{T}$ è* **soddisfacibile**, *qualora abbia almeno un modello. $\mathcal{T} \models A$ (letto con A è* **conseguenza logica** *di $\mathcal{T}$) significa che A è vera in ogni modello di $\mathcal{T}$. Una $\mathcal{L}$-formula A è una* **verità logica** *qualora $\mathcal{A} \models A$ valga per ogni $\mathcal{A}$.*

Lo scopo delle presenti note è lo studio del seguente problema: data una teoria (che supponiamo consti di un numero finito di assiomi) $\mathcal{T}$ e dato un enunciato A, come stabilire se

$$\mathcal{T} \models A$$

vale oppure no?[11] Si noti che questo equivale a stabilire se la teoria $\mathcal{T}' = \mathcal{T} \cup \{\neg A\}$ è insoddisfacibile oppure no. Se $\mathcal{T}$ è finita, tale è $\mathcal{T}'$ e $\mathcal{T}'$ può essere a sua volta sostituita dalla congiunzione delle formule che la compongono. Quindi il nostro problema sarà quello di stabilire se, dato un enunciato B, B è o meno soddisfacibile, cioè vero in una qualche struttura. Il primo passo sarà quello di preprocessare B stesso in modo da portarlo in una forma più trattabile.

3.3 Skolemizzazione

Osserviamo che ogni formula B di un linguaggio elementare è logicamente equivalente[12] ad una formula del tipo

$$Qx_1 \cdots Qx_n \; M$$

dove M non ha quantificatori (è 'aperta') e Qx_i è $\forall x_i$ oppure $\exists x_i$. Tale forma si chiama **forma normale prenessa**: M ne è la **matrice** e i quantificatori $Qx_1 \cdots Qx_n$ ne sono il **prefisso**.

La trasformazione in forma normale prenessa si ottiene applicando ripetutamente le seguenti leggi logiche che consentono si spostare i quantificatori verso l'esterno:[13]

$$\begin{aligned}
&\neg\forall y C \leftrightarrow \exists y \neg C,\\
&\neg\exists y C \leftrightarrow \forall y \neg C,\\
&\exists y A_1 \wedge A_2 \leftrightarrow \exists y (A_1 \wedge A_2),\\
&\exists y A_1 \vee A_2 \leftrightarrow \exists y (A_1 \vee A_2),\\
&\forall y A_1 \wedge A_2 \leftrightarrow \forall y (A_1 \wedge A_2),\\
&\forall y A_1 \vee A_2 \leftrightarrow \forall y (A_1 \vee A_2),
\end{aligned}$$

[11] Per il teorema di Church, questo problema nella sua forma generale è indecidibile.

[12] Una formula B è logicamente equivalente ad una formula B' se e solo se $B \leftrightarrow B'$ è una verità logica.

[13] Le leggi possono essere utilizzate in ordine qualsiasi, il risultato *può non essere lo stesso*, ma si ottiene comunque una formula equivalente a quella di partenza. Gli algoritmi di trasformazione esposti in questo paragrafo **sono soggetti a varie e molto importanti ottimizzazioni** (alcune implementate per esempio su SPASS), su cui non ci soffermiamo per semplicità. Se il risultato del processo di Skolemizzazione prodotto da SPASS non risulta chiaro, lo studente può utilmente disabilitare certe ottimizzazioni; ad esempio con l'opzione -CNFRenaming=0 si può disabilitare la rinomina di sottoformule mediante predicati-etichetta (si noti che però tale disabilitazione può avere effetti disastrosi sulla successiva esecuzione, cfr. ad esempio il caso del Problema 38 di Pelletier, inserito nel file 'Examples' della distribuzione di SPASS).

dove nelle ultime quattro verità logiche elencate si suppone y non libera in A_2. Quest'ultima limitazione fa sì che spesso si debbano operare delle *rinomine* (cioè dei cambi di nome) delle variabili vincolate che compaiono nella formula che si vuol portare in forma normale prenessa.

Abbiamo visto che ogni formula è logicamente equivalente ad una in forma normale prenessa. Rimanendo nel campo dell'equivalenza logica, non si può fare di più. Tuttavia, se si passa all'equisoddisfacibilità si possono fare passi ulteriori. Ricordiamo che una formula A è soddisfacibile sse esiste una struttura $\mathcal{A}$ tale che $\mathcal{A} \models A$. A e B sono equisoddisfacibili sse (A è soddisfacibile se e solo se lo è B). Se per quanto riguarda la soddisfacibilità ogni enunciato equivale ad un enuciato in una data classe ristretta, possiamo per stabilire se B è soddisfacibile o meno, trasformare B in un enunciato di tale classe ristretta e vedere se il trasformato è soddisfacibile o meno. In effetti, pur di ampliare il linguaggio originario, **dal punto di vista della soddisfacibilità ogni enunciato B equivale ad un enunciato universale**, ossia del tipo $\forall x_1 \cdots \forall x_n M$ con M aperta. La trasformazione avviene come segue. Per prima cosa portiamo B in forma normale prenessa e facciamo un'induzione sul numero dei quantificatori esistenziali del prefisso. Se tale numero è zero, abbiamo già raggiunto lo scopo. Se no, la forma normale prenessa è del tipo

$$(*) \qquad \forall x_1 \cdots \forall x_n \exists y Q z_1 \cdots Q z_m \; C.$$

Ampliamo il linguaggio con l'aggiunta di un nuovo simbolo funzionale f di arietà n (f è una costante, se $n = 0$); allora l'enunciato

$$\forall x_1 \cdots \forall x_n Q z_1 \cdots Q z_m \; C(f(x_1, \ldots, x_n)/y)$$

è equisoddisfacibile con $(*)$ (tale enunciato ha un prefisso con meno quantificatori esistenziali, quindi poi si applica l'ipotesi induttiva).

La matrice di una forma normale prenessa può a sua volta essere portata in una forma standard, detta **forma normale congiuntiva**(fnc). Una forma normale congiuntiva è una congiunzione di disgiunzioni formate solo da formule atomiche o atomiche negate. Anche per la trasformazione in fnc esponiamo il metodo più semplice ed ingenuo che consiste nell'applicare le seguenti leggi logiche proposizionali fino ad ottenere una formula nella

forma desiderata:[14]

$$\neg\neg B \leftrightarrow B,$$
$$\neg(B \vee C) \leftrightarrow \neg B \wedge \neg C,$$
$$\neg(B \wedge C) \leftrightarrow \neg B \vee \neg C,$$
$$A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C),$$
$$(B \wedge C) \vee A \leftrightarrow (B \vee A) \wedge (C \vee A).$$

Dopo aver portato la matrice in fnc, il nostro enunciato B originario che volevamo testare per la soddisfacibilità ha assunto la forma del tipo

$$(+) \qquad \forall x_1 \cdots \forall x_n (C_1 \wedge \cdots \wedge C_m)$$

dove le $C_1, \ldots, C_m$ sono disgiunzioni di formule atomiche e atomiche negate. La soddisfacibilità della $(+)$ equivale a trovare un modello per la teoria formata dalle m formule $\forall x_1 \cdots \forall x_n C_i$ $(i = 1, \ldots, m)$. Siccome una formula è vera in una struttura sse è vera la sua chiusura universale,[15] ci riduciamo a considerare la teoria formata dalle m formule

$$C_1, \ldots, C_m.$$

Queste formule non solo sono completamente senza quantificatori, ma sono anche senza congiunzioni. Una formula di tale tipo è necessariamente della forma

$$\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$$

dove le A_i, B_j sono atomiche. Usiamo la notazione a clausole, cioè la notazione $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$ per indicare tali formule (questo è il formato delle formule che vengono processate dagli attuali dimostratori automatici basati su metodi di saturazione). Formalmente, una **clausola** è una coppia di **multiinsiemi** di formule atomiche scritta nella forma[16]

$$(C) \qquad A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$$

[14] Un metodo efficiente di trasformazione in fnc si può ottenere rinunciando al requisito che la formula e la sua trasformata siano logicamente equivalenti (anche in questo caso infatti per i nostri scopi è sufficiente il requisito molto più debole dell'equisoddisfacibilità). È ad esempio qui che risulta interessante il procedimento di rinomina tramite predicati-etichetta, tipico delle cosiddette 'trasformazioni strutturali'.

[15] Si veda la Definizione 2 del paragrafo3 (la chiusura universale di una formula D è la formula $\forall x_1 \cdots \forall x_n D$, dove $x_1, \ldots, x_n$ sono tutte e sole le variabili che occorrono libere in D).

[16] In generale, usiamo le lettere greche $\Gamma, \Delta, \ldots$ per indicare *multiinsiemi* di formule atomiche, ossia insiemi (eventualmente vuoti) di formule atomiche con possibili ripetizioni. Ad esempio, Δ potrà indicare $\{A, B, C\}$, oppure $\{A, B, B\}$, oppure $\{\ \}$, ecc. Una clausola sarà quindi notata con $\Gamma \Rightarrow \Delta$.

Le formule $A_1, \ldots, A_n$ sono dette **letterali negativi** della clausola (C) e le formule $B_1, \ldots, B_m$ sono dette **letterali positivi** della clausola (C). Ripetiamo che la (C), vista come assioma di una teoria, denota la formula $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$[17] (o, se si preferisce, la sua chiusura universale).

Riassumendo, data una formula B che si vuole preprocessare allo scopo di sapere poi se è soddisfacibile o meno, si devono eseguire i seguenti **quattro** passaggi: a) trasformazione in forma normale prenessa (mediante spostamento - con eventuali rinomine - dei quantificatori verso l'esterno); b) skolemizzazione (mediante eliminazione dei quantificatori esistenziali del prefisso con nuovi simboli di funzione e/o di costante); c) trasformazione in fnc della matrice; d) riduzione ad un insieme di clausole.

ESEMPIO. Facciamo i passi necessari di preprocessamento per sapere se la formula

$$\forall x \exists y (R(x,y) \rightarrow Q(f(x)) \tag{1}$$

implica o meno logicamente la formula[18]

$$\forall x \exists y (R(x,y) \rightarrow Q(f(y))). \tag{2}$$

La (1) già in forma normale prenessa; la skolemizzazione dà $\forall x (R(x, s(x)) \rightarrow Q(f(x)))$ e la trasformazione in fnc della matrice produce $\forall x (\neg R(x, s(x)) \vee Q(f(x)))$. La trasformazione in clausole produce la singola clausola

$$R(x, s(x)) \Rightarrow Q(f(x)). \tag{C_1}$$

[17] Tale formula è ovviamente logicamente equivalente alla formula $\neg A_1 \vee \cdots \neg A_n \vee B_1 \vee \cdots \vee B_m$.

[18] Questo è il formato generale dei problemi che SPASS accetta: vengono specificate n formule $A_1, \ldots, A_n$ sotto la voce 'axioms' e m formule $B_1, \ldots, B_m$ sotto la voce 'conjecture'. Il problema è quello di stabilire se la congiunzione delle A_i ha come conseguenza logica la disgiunzione delle B_j. Il preprocessamento consiste nei quattro passi di cui sopra applicati alle formule

$$A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m.$$

Il preprocessamento si può fare per ciascuna formula separatamente avendo cura però di non utilizzare mai due volte lo stesso simbolo di funzione per skolemizzare.

Osservazione: dal luglio del 2007 è in linea la versione 3.0 di SPASS. Se utilizzando tale versione (o la precedente) sorgessero problemi con l'editor, per eliminarli, basta modificare opportunamente le font (usare l'ultimo pulsante a destra del menu in alto).

La (2) va negata (si ricordi che il problema deve essere ridotto ad un problema di soddisfacibilità aggiungendo la negazione della tesi alle ipotesi) e poi portata in forma normale prenessa con $\exists x \forall y \neg(R(x,y) \rightarrow Q(f(y)))$. La skolemizzazione produce $\forall y \neg(R(c,y) \rightarrow Q(f(y)))$ e la trasformazione della matrice in fnc dà $\forall y(R(c,y) \wedge \neg Q(f(y)))$. Otteniamo infine le due clausole

$$(C_2) \qquad \Rightarrow R(c,y)$$

$$(C_3) \qquad Q(f(y)) \Rightarrow .$$

Vedremo nel capitolo 4.2 che l'insieme delle tre clausole $\{C_1, C_2, C_3\}$ è davvero inconsistente (quindi la (1) ha come conseguenza logica la (2)).

3.4 Il metodo di Herbrand

In questo paragrafo illustreremo come ridurre il problema della soddisfacibilità di un insieme finito di clausole ad un problema di soddisfacibilità nella logica proposizionale. La riduzione che proporremo è stata storicamente utilizzata prima della nascita del calcolo basato sulla Risoluzione (ossia del calcolo implementato su SPASS) ma, con l'aumento delle prestazioni dei moderni SAT-solvers, è stata riconsiderata positivamente anche in tempi recenti. Data l'indecidibilità della logica del primo ordine, la riduzione che proporremo non fornirà una procedura di decisione e l'algoritmo basato su di essa sarà passibile di divergere.

Una *astrazione proposizionale* è una funzione che associa (in modo iniettivo) ad ogni formula atomica ground A una lettera proposizionale p_A. Se C è la clausola $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$, definiamo *astrazione proposizionale di* C la clausola proposizionale

$$\neg p_{A_1} \vee \cdots \vee \neg p_{A_n} \vee p_{B_1} \vee \cdots \vee p_{B_m}.$$

L'**universo di Herbrand** di un linguaggio $\mathcal{L}$ è formato da tutti i termini ground di $\mathcal{L}$. Una istanza ground di una clausola C è una clausola che si ottiene da C sostituendo tutte le variabili che occorrono in C con termini ground.

La riduzione di cui parlavamo si ottiene immediatamente dal seguente risultato, noto come teorema di Herbrand:

Teorema 3.4. *Un insieme di clausole $\mathcal{C}$ è insoddisfacibile se e solo se esiste un insieme finito di istanze ground di clausole appartenenti a $\mathcal{C}$ la cui astrazione proposizionale è insoddisfacibile.*

Quindi per scoprire che $\mathcal{C}$ è insoddisfacibile basta generare le astrazioni proposizionali delle istanze ground delle clausole di $\mathcal{C}$ e passarle ad un SAT-solver proposizionale. Il problema è che l'universo di Herbrand è di solito infinito, quindi le clausole da passare al SAT-solver per il test di soddisfacibilità proposizionale sono infinite. Occorre quindi generare un insieme finito di tali clausole e passarlo al SAT-solver; se il test di soddisfacibilità è positivo, si genera un soprainsieme di tali clausole, si ripete il test di soddisfacibilità e così via.

Se $\mathcal{C}$ è insoddisfacibile e se le istanze ground delle clausole di $\mathcal{C}$ vengono generate *tutte*, prima o poi si scoprirà che $\mathcal{C}$ è insoddisfacibile perchè il test di soddisfacibilità sarà negativo. Se invece $\mathcal{C}$ è soddisfacibile, la procedura continuerà all'infinito, salvo nei casi in cui l'universo di Herbrand è finito:[19] in tali casi, la procedura si arresterà comunque.

ESEMPIO. Risolviamo il seguente problema:

Nessuno compera il Times se non è molto istruito. Nessun porcospino sa leggere. Chi non sa leggere non è molto istruito. Dimostrare che: 'Nessun porcospino compera il Times'.

Le ipotesi del problema si possono formalizzare mediante le formule:

$$\forall x(C(x) \to I(x)), \quad \forall x\neg(P(x) \wedge L(x)), \quad \forall x(\neg L(x) \to \neg I(x)),$$

dove abbiamo usato i predicati unari C, I, P, L per 'comprare il Times', 'essere molto istruito', 'essere porcospino' e 'saper leggere'. La tesi viene formalizzata con

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

Negando la tesi e skolemizzando tutto, otteniamo le clausole:

$$C(x) \Rightarrow I(x), \quad P(x), L(x) \Rightarrow, \quad I(x) \Rightarrow L(x), \quad \Rightarrow P(c), \quad \Rightarrow C(c),$$

[19] Si tratta ovviamente dei casi in cui il linguaggio di $\mathcal{C}$ non contiene simboli di funzione, ma solo costanti (si tenga presente che il linguaggio di $\mathcal{C}$ può essere più ricco del linguaggio originario, perchè contiene anche costanti e funzioni introdotte dalla skolemizzazione).

dove c è una nuova costante. L'universo di Herbrand consta solo della costante c, per cui le istanze ground che ci servono sono semplicemente le seguenti

$$C(c) \Rightarrow I(c), \quad P(c), L(c) \Rightarrow, \quad I(c) \Rightarrow L(c), \quad \Rightarrow P(c), \quad \Rightarrow C(c),$$

Astraiamo proposizionalmente $C(c), I(c), P(c), L(c)$ con p_1, p_2, p_3, p_4, rispettivamente; il test proposizionale andrà quindi eseguito sull'insieme di clausole seguenti:

$$\neg p_1 \vee p_2, \quad \neg p_3 \vee \neg p_4, \quad \neg p_2 \vee p_4, \quad p_3, \quad p_1.$$

Tale test proposizionale è negativo, come si può constatare sia direttamente, sia passando ad un SAT-solver il seguente file:

```
p cnf  4 5
-1 2 0
-3 -4 0
-2 4 0
3 0
1 0
```

Con il calcolo della Risoluzione, vedremo che sarà possibile ottenere casi di terminazione per insiemi di clausole soddisfacibili su universi di Herbrand infiniti (anche se la terminazione non potrà mai essere garantita da nessun calcolo nel caso generale, stante l'indecidibilità della logica del primo ordine).

4 Risoluzione

In questo capitolo studiamo il calcolo della Risoluzione (introdotto da Robinson nel 1965). Tale calcolo si applica a linguaggi *senza identità* e consente di scoprire se un insieme di clausole è o meno soddisfacibile. Il calcolo della Risoluzione è soggetto a varie ottimizzazioni, tra cui anche la Risoluzione Ordinata (implementatata ad esempio su SPASS), di cui però ci occuperemo più avanti nel paragrafo 5.3.

Prima di spiegare la regola di Risoluzione, introdurremo il problema dell'unificazione, che è centrale in tutti gli aspetti della dimostrazione automatica odierna. L'uso dell'unificazione serve essenzialmente a fare un uso oculato delle istanziazioni nelle deduzioni (invece di istanziare un'ipotesi o un lemma disponibile in tutti i modi possibili, si va a ricercare l'istanziazione più efficace mediante l'algoritmo di unificazione).

4.1 Unificazione

Una **sostituzione** σ è una mappa avente per dominio l'insieme delle variabili e per codominio l'insieme $T_{\mathcal{L}}$ dei termini di $\mathcal{L}$

$$\sigma : \mathcal{V} \longrightarrow T_{\mathcal{L}}$$

con la proprietà che l'insieme delle variabili x tali che $x \not\equiv \sigma(x)$ è finito (tale insieme è detto essere il dominio della σ).

Se $\{x_1, \ldots, x_n\}$ è il dominio della sostituzione σ, possiamo indicare la σ stessa mediante la notazione

$$x_1 \mapsto \sigma(x), \quad \ldots, \quad x_n \mapsto \sigma(x_n).$$

La σ può essere estesa ad una mappa

$$\hat{\sigma} : T_{\mathcal{L}} \longrightarrow T_{\mathcal{L}}$$

mediante la definizione induttiva

- $\hat{\sigma}(x) \equiv \sigma(x)$;
- $\hat{\sigma}(f(t_1, \ldots, t_n) \equiv f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n))$.

Ancora, la σ si può estendere alle formule atomiche, mediante

$$\hat{\sigma}(P(t_1, \ldots, t_n)) \equiv P(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n)).$$

Analoghe estensioni si possono fare per liste di formule atomiche, clausole, ecc. nel modo ovvio. In tutti questi casi, scriveremo

$$E\sigma$$

per indicare $\hat{\sigma}(E)$, dove E è un'espressione (cioè un termine, una formula atomica, una lista di formule atomiche, una clausola, ecc.): $E\sigma$ non è nient'altro che il risultato della sostituzione *simultanea* in E di x con $\sigma(x)$ per ogni variabile x.

Le sostituzioni possono essere composte: se

$$\sigma : \mathcal{V} \longrightarrow T_{\mathcal{L}}, \qquad \tau : \mathcal{V} \longrightarrow T_{\mathcal{L}}$$

sono due sostituzioni, la sostituzione composta (che indichiamo con $\sigma\tau$) è la sostituzione il cui valore su $x \in \mathcal{V}$ è dato da $(x\sigma)\tau$ (ossia $\sigma\tau$, calcolato su x, dà il valore della $\hat{\tau}$ calcolato sul termine $\sigma(x)$). La composizione di sostituzioni è associativa nel senso che abbiamo la legge

$$(\sigma\tau)\upsilon = \sigma(\tau\upsilon)$$

per ogni terna di sostituzioni σ, τ, υ. La sostituzione identica id (cioè la sostituzione che non cambia il valore di nessuna variabile) fa da elemento neutro nel senso che abbiamo la legge:

$$\sigma\, id = id\, \sigma = \sigma$$

per ogni sostituzione σ.

Una sostituzione che semplicemente permuta le variabili del suo dominio è detta essere una *rinomina*. Se ρ è una rinomina, esiste sempre una sostituzione, che chiamiamo ρ^{-1}, che è una rinomina a sua volta ed è tale che $\rho\rho^{-1} = \rho^{-1}\rho = id$. Ad esempio, la sostituzione

$$x \mapsto y, \qquad y \mapsto x$$

è una rinomina la cui inversa è lei stessa.

Una sostituzione σ è detta **più generale** di una sostituzione τ, qualora esista una sostituzione δ tale che $\sigma\delta = \tau$. Intuitivamente, questo significa che la τ è ottenibile dalla σ mediante una successiva istanziazione. Ad esempio, la sostituzione σ

$$x \mapsto f(z), \qquad y \mapsto z$$

è più generale della sostituzione τ

$$x \mapsto f(c), \qquad y \mapsto c, \qquad z \mapsto c$$

in quanto la τ si ottiene dalla σ componendola con l'ulteriore sostituzione δ

$$z \mapsto c.$$

Scriviamo $\sigma \geq \tau$ per dire che σ è più generale della τ. La relazione $\geq$ è riflessiva e transitiva; può essere che valgano simultaneamente le relazioni $\sigma \geq \tau$ e $\tau \geq \sigma$, ma in tal caso σ e τ differiscono solo per una rinomina (ossia esiste una rinomina ρ tale che $\sigma\rho = \tau$).

Definizione 4.1. *Un* **problema di unificazione** *è una lista di coppie di termini, che viene scritta nella forma*

$$(U) \qquad t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n.$$

Una **soluzione** *ad* (U) *è una sostituzione* σ *tale che*

$$t_1\sigma \equiv u_1\sigma, \ldots, t_n\sigma \equiv u_n\sigma.$$

Una soluzione μ *di* (U) *è detta essere un upg (***'unificatore più generale'***)*[20] *se e solo se è più generale di ogni altra soluzione possibile di* (U)*.*

Si noti che due upg, per quanto detto sopra, possono differire solo per una rinomina. Il fatto importante è che ogni problema di unificazione **o non ammette alcuna soluzione oppure ammette un upg**. Tale upg si può calcolare eseguendo l'algoritmo che spiegheremo qui oltre (nel caso in cui il problema non abbia soluzione, l'algoritmo si arresta in stato di fallimento).

Esistono varie versioni dell'algoritmo di unificazione; algoritmi efficienti ('quasi lineari') si possono ottenere mediante opportune rappresentazioni dei termini come grafi aciclici diretti. Qui proponiamo la versione più facile da spiegare (che non è certamente la migliore in assoluto).

L'algoritmo manipola insiemi di equazioni e si inizializza alle equazioni del problema originario (U). Ad ogni passo viene eseguita una delle seguenti operazioni (finchè non ci sono più operazioni da eseguire o finchè non si raggiunge lo stato di fallimento). L'ordine di applicazione delle operazioni è ininfluente (a meno di una rinomina) sul risultato. Le operazioni sono le seguenti:

(1) cancellare le equazioni del tipo $t \stackrel{?}{=} t$;

[20] 'Most general unifier' (abbreviato mgu) in inglese.

(2) sostituire un'equazione del tipo $f(t_1, \ldots, t_n) \stackrel{?}{=} f(u_1, \ldots, u_n)$ con le n equazioni $t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n$;

(3) sostituire un'equazione del tipo $t \stackrel{?}{=} x$ con $x \stackrel{?}{=} t$, qualora t non sia una variabile;

(4) sostituire, qualora si incontri un'equazione del tipo $x \stackrel{?}{=} t$ con t che non contiene occorrenze di x (ma con x che occorre in almeno una delle rimanenti equazioni), tutte le occorrenze di x nelle rimanenti equazioni con t;

(5) terminare in stato di fallimento qualora si incontri un'equazione del tipo $f(t_1, \ldots, t_n) \stackrel{?}{=} g(u_1, \ldots, u_m)$ con $f \not\equiv g$;

(6) terminare in stato di fallimento qualora si incontri un'equazione del tipo $x \stackrel{?}{=} t$ con t che è diverso da x ma contiene un'occorrenza di x.[21]

Se non fallisce, l'algoritmo termina in uno stato del tipo

$$y_1 \stackrel{?}{=} t_1(\underline{z}), \ldots, y_k \stackrel{?}{=} t_k(\underline{z})$$

dove le variabili $y_1, \ldots, y_k, \underline{z}$[22] sono tutte distinte. Tali equazioni danno direttamente l'upg voluto nella forma

$$y_1 \mapsto t_1(\underline{z}), \ldots, y_k \mapsto t_k(\underline{z}).$$

Omettiamo la verifica della correttezza e terminazione dell'algoritmo; segnaliamo solo che per verificare la correttezza basta osservare che ogni singola operazione lascia inalterato l'insieme delle soluzioni e che per verificare la terminazione basta introdurre una opportuna misura di complessità (che fa riferimento sia al numero delle variabili che hanno 'equazione risolvente' che al numero dei simboli).

ESEMPIO Consideriamo il problema di unificazione:

$$g(y) \stackrel{?}{=} x, \quad f(x, h(x), y) \stackrel{?}{=} f(g(z), w, z).$$

[21] Questo caso di fallimento è detto fallimento per *occur check*: un problema del tipo $x \stackrel{?}{=} f(x)$ non è risolvibile, perchè per ogni σ, i termini $x\sigma$ e $f(x)\sigma \equiv f(x\sigma)$ non avranno mai la stessa lunghezza e pertanto non potranno mai coincidere.

[22] Qui e in simili circostanze, $\underline{z}$ indica una lista finita e senza ripetizioni di variabili.

Applicando l'istruzione di decomposizione (2) otteniamo:

$$g(y) \stackrel{?}{=} x, \quad x \stackrel{?}{=} g(z), \quad h(x) \stackrel{?}{=} w, \quad y \stackrel{?}{=} z;$$

applicando l'istruzione di orientamento (3) abbiamo

$$x \stackrel{?}{=} g(y), \quad x \stackrel{?}{=} g(z), \quad h(x) \stackrel{?}{=} w, \quad y \stackrel{?}{=} z;$$

ora applichiamo l'istruzione di sostituzione (4), ottenendo

$$x \stackrel{?}{=} g(y), \quad g(y) \stackrel{?}{=} g(z), \quad h(g(y)) \stackrel{?}{=} w, \quad y \stackrel{?}{=} z;$$

di nuovo, per la (2) si ha (si noti che equazioni ripetute sono automaticamente eliminate perchè i tipi di dati su cui operiamo sono insiemi di equazioni)

$$x \stackrel{?}{=} g(y), \quad y \stackrel{?}{=} z, \quad h(g(y)) \stackrel{?}{=} w;$$

ci manca solo un passo di orientamento (3) per ottenere il risultato finale

$$x \stackrel{?}{=} g(y), \quad y \stackrel{?}{=} z, \quad w \stackrel{?}{=} h(g(y)).$$

L'upg è quindi dato dalla sostituzione $x \mapsto g(z), \quad y \mapsto z, \quad w \mapsto h(g(z))$.

Nel seguito, avremo bisogno di considerare problemi di unificazione relativi a due formule atomiche

$$P(t_1, \ldots, t_n) \stackrel{?}{=} Q(u_1, \ldots, u_m).$$

Questi problemi hanno soluzione solo se $P \equiv Q$ (quindi $n = m$) ed in tal caso si riducono al problema di tipo standard che abbiamo già trattato

$$t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n.$$

Un altro problema molto importante simile all'unificazione è il matching.

Definizione 4.2. *Un* **problema di matching** *è una lista di coppie di termini, che viene scritta nella forma*

$$(M) \qquad t_1 \stackrel{?}{\preceq} u_1, \ldots, t_n \stackrel{?}{\preceq} u_n.$$

Una **soluzione** *ad* (M) *è una sostituzione* σ *tale che*

$$t_1\sigma \equiv u_1, \ldots, t_n\sigma \equiv u_n.$$

Il matching differisce dall'unificazione perchè la soluzione (che si dimostra essere **unica**, se esiste) viene applicata solo ai membri sinistri del problema (M) di partenza. Il matching può essere ricondotto all'unificazione nel modo seguente: innanzitutto possiamo supporre (nei casi concreti sarà sempre così) che le variabili che compaiono nei membri destri del problema (M) (cioè negli u_i) siano disgiunte da quelle che compaiono nei membri sinistri (cioè nei t_i). A questo punto possiamo trattare (M) come un problema di unificazione considerando le variabili che compaiono nei membri destri come delle *costanti*. Va però osservato che questo modo di vedere i problemi di matching è un po' semplicistico (anche se sufficiente per sviluppare la teoria): in effetti il matching è un problema molto più semplice dell'unificazione e va implementato in modo autonomo per ragioni di efficienza (un algoritmo diretto di matching risulta molto più rapido).

4.2 Il calcolo R

Lo scopo principale di queste note è di introdurre calcoli logici per manipolare clausole. Formalmente, chiamiamo **regola di inferenza** ad n premesse ($n \geq 1$) una $n+1$-pla di clausole che scriviamo nella forma

$$\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$$

Supporremo sempre tacitamente (per tutte le presenti note) che **due premesse di una regola di inferenza abbiano sempre variabili disgiunte fra loro**.

Un **calcolo** $\mathcal{K}$ è un insieme di regole di inferenza; usualmente $\mathcal{K}$ è infinito, poichè le regole sono normalmente schemi di regole (ad esempio, lo schema di regola di Risoluzione che vedremo sta per l'insieme infinito di regole ciascuna delle quali consta di una terna di clausole del tipo indicato nella regola stessa).

Per definizione, d'ora in poi una teoria $\mathcal{T}$ è semplicemente un insieme di clausole (detti assiomi di $\mathcal{T}$). Una **dimostrazione** in $\mathcal{K}$ da una teoria $\mathcal{T}$ è una lista finita di clausole

$$C_1, \ldots, C_k$$

ciascuna delle quali è o un assioma di $\mathcal{T}$ (eventualmente **rinominato**), oppure è ottenuta da clausole che la precedono nella lista (eventualmente **rinominate**) mediante una regola di inferenza di $\mathcal{K}$.

Diciamo che il calcolo $\mathcal{K}$ è **refutazionalmente completo** sse, data una qualsiasi teoria $\mathcal{T}$, si ha che $\mathcal{T}$ **è inconsistente se e solo se esiste una dimostrazione in** $\mathcal{K}$ **da** $\mathcal{T}$ **della clausola vuota** $\Rightarrow$. Un teorema di completezza refutazionale è sempre un risultato di un certo rilievo che usualmente richiede tecniche matematiche piuttosto sofisticate (su cui non ci soffermeremo visto il taglio prevalentemente applicativo di queste note). Segnaliamo però che **tutti** i calcoli che presenteremo (ossia il calcolo $\mathcal{R}$ del presente paragrafo, il calcolo della Risoluzione Ordinata del paragrafo 5.3 e il calcolo $\mathcal{S}$ del paragrafo 6.2) sono refutazionalmente completi. Si noti che la completezza refutazionale non implica affatto che il calcolo sia in grado di trovare una dimostrazione di tutte le conseguenze logiche di una teoria $\mathcal{T}$, ma solo di scoprire se $\mathcal{T}$ è inconsistente o meno (ossia se ammetta o meno un modello, secondo la Definizione 3 del paragrafo 3). Questo punto è molto importante: la completezza *tout court* non sarebbe una proprietà logica desiderabile (in particolare per i linguaggi con identità), in quanto renderebbe il calcolo terribilmente inefficiente.

La completezza refutazionale dà un metodo di **saturazione**: per scoprire l'eventuale inconsistenza, basta chiudere l'insieme di clausole $\mathcal{T}$ per le regole di inferenza fornite dal calcolo. Se la chiusura produce la clausola vuota, $\mathcal{T}$ non sarà soddisfacibile, altrimenti il ragionamento impiegato nella dimostrazione del teorema di completezza refutazionale dirà (almeno in astratto) come costruire un modello di $\mathcal{T}$. Si noti che non c'è nessuna garanzia che il processo di saturazione termini (questo a causa dell'indecidibilità della logica del primo ordine), tuttavia vedremo dei raffinamenti nel Capitolo 6 atti ad aumentare i casi di terminazione (oltre che a rendere più efficiente il processo di ricerca della dimostrazione della clausola vuota).

In questo paragrafo presentiamo il calcolo della Risoluzione $\mathcal{R}$ (introdotto da J.A. Robinson nel 1965), che è adatto a trattare solo linguaggi **senza identità**. Il calcolo ha le due sole regole di inferenza seguenti (in entrambe le regole, μ denota un upg del problema di unificazione $A \stackrel{?}{=} B$):

Regola di Risoluzione

$$\frac{\Gamma \Rightarrow \Delta, A \qquad B, \Gamma' \Rightarrow \Delta'}{\Gamma\mu, \Gamma'\mu \Rightarrow \Delta\mu, \Delta'\mu}$$

Regola di Fattorizzazione Destra

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma\mu \Rightarrow \Delta\mu, A\mu}$$

Si osservi che (data la convenzione che abbiamo fatto all'inizio di questo paragrafo), per applicare una regola di inferenza (come la regola di Risoluzione) occorre, se necessario, preventivamente **rinominare** una delle due premesse in modo che non abbia variabili in comune con l'altra.

Per applicare la regola di Risoluzione occorre: a) scegliere un letterale positivo A (cioè una formula atomica a destra di $\Rightarrow$) nella prima premessa; b) scegliere un letterale negativo B (cioè una formula atomica a sinistra di $\Rightarrow$) nella seconda premessa; c) risolvere il problema di unificazione $A \stackrel{?}{=} B$ e d) (nel caso che il problema sia risolvibile) cancellare A dalla prima premessa, cancellare B dalla seconda, applicare ovunque l'upg trovato e fondere le due clausole.[23] Per applicare la Fattorizzazione Destra occorre puntare due letterali positivi, unificarli e cancellarne uno (l'upg trovato viene applicato anche ai rimanenti letterali della clausola stessa). Torneremo più avanti (nel paragrafo 5.3) su criteri ottimizzati di scelta dei letterali da puntare per l'unificazione in queste due regole.

Segnaliamo che la seguente

Regola di Fattorizzazione Sinistra

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma\mu, A\mu \Rightarrow \Delta\mu}$$

viene talvolta implementata (o data per opzionale) nei dimostratori automatici correnti, anche se essa non sarebbe strettamente necessaria per ottenere la completezza refutazionale del calcolo. Come già preannunciato vale il seguente

Teorema 4.3. *IL calcolo $\mathcal{R}$ è refutazionalmente completo (per i linguaggi senza identità).*

[23] Si faccia attenzione al fatto che le nostre clausole sono *multiinsiemi* (e non semplicemente insiemi) di letterali: questo previene complicazioni indesiderate (ad esempio, solo utilizzando multiinsiemi e non insiemi si ha che il togliere una occorrenza di un letterale da una clausola e l'applicare una sostituzione alla clausola stessa costituiscono due operazioni permutabili).

ESEMPIO. Riprendiamo in esame le tre clausole dell'Esempio del paragrafo 3.3:

$$1.\ R(x, s(x)) \Rightarrow Q(f(x))$$
$$2.\ \Rightarrow R(c, y)$$
$$3.\ Q(f(y)) \Rightarrow$$

Otteniamo la seguente prova della clausola vuota in due passaggi:

$$4.\ R(x, s(x)) \Rightarrow \quad [Res\ 1.2; 3.1]$$
$$5.\ \Rightarrow \quad [Res\ 2.1; 4.1]$$

dove nel primo passaggio abbiamo applicato la regola di Risoluzione (contrassegnata con la sigla Res) al secondo letterale della clausola 1 e al primo letterale della clausola 3, mentre nel secondo passaggio abbiamo applicato sempre la regola di Risoluzione al primo letterale della clausola 2 e al primo letterale della clausola 4.[24]

ESEMPIO. Il seguente esempio dimostra che la regola di Fattorizzazione è indispensabile. Vogliamo dimostrare che la formula

$$\forall x \forall y (R(x, f(y)) \vee R(y, f(x)))$$

implica logicamente la formula:

$$\exists x \exists y (R(x, f(y)) \wedge R(y, f(x))).$$

Il procedimento di skolemizzazione genera le due clausole:

$$1.\ \Rightarrow R(x, f(y)), R(y, f(x))$$
$$2.\ R(x, f(y)), R(y, f(x)) \Rightarrow$$

Utilizzando la sola regola di Risoluzione si ottengono solo clausole tautologiche (ossia con la stessa formula atomica che figura sia a destra che a sinistra di $\Rightarrow$). Invece, usando la Fattorizzazione Destra si ottiene le seguente prova:

$$3.\ \Rightarrow R(x, f(x)) \quad [Fac\ 1.1; 1.2]$$
$$4.\ R(x, f(x)) \Rightarrow \quad [Res\ 3.1; 2.1]$$
$$5.\ \Rightarrow \quad [Res\ 3.1; 4.1]$$

[24] Per ottenere la visualizzazione della dimostrazione della clausola vuota da un insieme di clausole insoddisfacibili in SPASS occorre usare l'opzione '-DocProof' (altrimenti viene dato solo il messaggio 'Proof found'). SPASS enumera i letterali partendo da 0 (e non da 1 come abbiamo fatto noi nel testo).

ESEMPIO. Nel caso di un insieme di clausole soddisfacibili il procedimento di saturazione può o meno terminare, a causa dell'indecidibilità del calcolo. Segnaliamo un esempio in cui non si ha terminazione (ma in cui la terminazione si potrà ottenere grazie alle tecniche di ordinamento che vedremo in seguito). Consideriamo l'insieme di clausole:

$$1.\ \Rightarrow P(c)$$
$$2.\ P(x) \Rightarrow P(f(x))$$

Risolvendo la prima clausola con la seconda si ottiene $\Rightarrow P(f(c))$; risolvendo ancora si ottengono le clausole $\Rightarrow P(f(f(c)))$, $\Rightarrow P(f(f(f(c))))$, ... e così via all'infinito.

ESEMPIO. (Cavalieri e Furfanti) Risolviamo il seguente quesito:

Nell'isola dei cavalieri, furfanti e lupi mannari, ogni abitante è un cavaliere o un furfante, mentre alcuni abitanti sono lupi mannari. Come è noto, i cavalieri dicono sempre la verità, i furfanti mentono sempre e i lupi mannari divorano gli uomini nelle notti di luna piena; si sa anche che i lupi mannari possono essere indifferentemente cavalieri o furfanti. Un esploratore incontra tre abitanti dell'isola a, b e c, di cui sa che esattamente uno è un lupo mannaro (pur non sapendo quale). Gli abitanti fanno le seguenti affermazioni: (I) a dice che c è il lupo mannaro; (II) b dice di non essere il lupo mannaro; (III) c dice che almeno due di loro sono furfanti. Chi deve scegliere l'esploratore fra a, b e c come compagno di viaggio, per evitare di scegliere proprio il lupo mannaro?

Dobbiamo innanzitutto formalizzare le informazioni rilevanti contenute nel testo (tale formalizzazione produrrà gli 'axioms' da inserire nel dimostratore automatico). Ci servono tre predicati unari C, F, L (che stanno per 'cavaliere', 'furfante' e 'lupo'); avremo inoltre tre costanti a, b, c. La formula

$$\forall x(C(x) \vee F(x)) \wedge \forall x \neg (C(x) \wedge F(x))$$

esprime il fatto che ogni abitante è cavaliere o furfante (e non entrambi); la formula

$$L(a) \vee L(b) \vee L(c)$$

esprime il fatto che c'è almeno un lupo fra a, b, c e la formula

$$\neg(L(a) \wedge L(b)) \wedge \neg(L(a) \wedge L(c)) \wedge \neg(Lb) \wedge L(c)$$

esprime il fatto che ce ne è al più uno. L'affermazione (I) è vera se a è un cavaliere, falsa altrimenti; formalizziamo questa informazione con la formula

$$(C(a) \to L(c)) \wedge (F(a) \to \neg L(c)).$$

Analogamente, da (II) otteniamo la formula

$$(C(b) \to \neg L(b)) \wedge (F(b) \to L(b)).$$

(III) dà origine alle due formule (che scriviamo separatamente per motivi di spazio)

$$C(c) \to (F(a) \wedge F(b)) \vee (F(a) \wedge F(c)) \vee (F(b) \wedge F(c))$$
$$F(c) \to \neg((F(a) \wedge F(b)) \vee (F(a) \wedge F(c)) \vee (F(b) \wedge F(c)))$$

Non ci resta che provare ora a vedere cosa succede aggiungendo (in tre esperimenti diversi) le 'conjecture' $\neg L(a)$, $\neg L(b)$, $\neg L(c)$. Nel primo caso il dimostratore automatico segnala 'proof found', il che vuol dire che certamente a non è il lupo. Negli altri due casi si ha 'completion found', il che vuol dire che non c'è alcuna garanzia che il lupo non sia proprio b o, rispettivamente, c (il messaggio 'completion found' significa che gli assiomi e la negazione della congettura non sono un insieme inconsistente di formule, per cui certamente esiste un modello - ossia una situazione possibile - in cui gli assiomi sono veri e la congettura falsa).

ESEMPIO. Si può colorare la carta geografica europea con tre colori? Per scoprirlo, formalizziamo il problema e inseriamo i dati in un dimostratore automatico. Ci serve un predicato binario C che esprime la relazione di 'confinare con'; ci serve anche una costante per denotare ogni paese europeo (ad esempio, i per Italia, s per Svizzera, a per Austria, ecc.). Inoltre ci servono 3 predicati unari K_1, K_2, K_3 per denotare i tre colori. Dovremo mettere fra gli assiomi il fatto che C è una relazione simmetrica, ossia la formula $\forall x \forall y(C(x,y) \to C(y,x))$; dovremo inserire le relazioni fisiche di confine, ad esempio $C(i,s), C(i,a), C(s,a), \ldots$. Inoltre va detto che ogni stato ha un colore $\forall x(K_1(x) \vee K_2(x) \vee K_3(x))$ e uno solo $\forall x \bigwedge_{i \neq j} \neg(K_i(x) \wedge K_j(x))$. Infine si deve imporre che due stati confinanti non hanno lo stesso colore

$$\forall x \forall y(C(x,y) \to \bigwedge_{i=1}^{3} \neg(K_i(x) \wedge K_i(y))).$$

Non c'è alcuna congettura in questo problema; se i dati sono consistenti (ossia se compare il messaggio 'completion found') la colorazione esiste, altrimenti (ossia se compare il messaggio 'proof found') la colorazione non esiste.

4.3 La struttura di un dimostratore automatico

Un dimostratore automatico, deve saper gestire il processo di saturazione delle clausole di una teoria $\mathcal{T}$. Per far questo, deve tentare di applicare tutte le regole di inferenza previste dal calcolo in un qualche ordine ragionevole. Siccome però il processo di saturazione è destinato a generare in casi significativi una enorme quantità di clausole, molte delle quali chiaramente inutilizzabili, doppie, ridondanti, ecc., è bene prevedere, accanto al processo deduttivo vero e prorio, anche un processo intrecciato di pulizia della memoria. Saranno quindi previste, accanto alle regole deduttive, delle regole di **riduzione** che prevedano l'eliminazione di clausole: queste regole di riduzione saranno applicate sia *in avanti* ('forward reduction') relativamente alle clausole appena dedotte, sia *all'indietro* ('backward reduction') relativamente alle clausole già memorizzate e trattate o da trattare. L'uso di regole di riduzione deve essere fatto in modo accorto, onde non distruggere la completezza refutazionale del procedimento: ritorneremo in modo più preciso sull'argomento nel paragrafo 6.4, per il momento segnaliamo tre regole di riduzione per il calcolo $\mathcal{R}$ e successivamente un semplice schema di implementazione di base.

(1) Una prima regola di riduzione è la regola **Taut** che consente di eliminare le clausole tautologiche del tipo

$$\Gamma, A \Rightarrow \Delta, A.$$

(2) Una seconda regola di semplificazione è la regola di sussunzione. Diciamo che la clausola $\Gamma \Rightarrow \Delta$ sussume la clausola $\Gamma' \Rightarrow \Delta'$, qualora per un matcher σ si abbia $\Gamma\sigma \subseteq \Gamma'$ e $\Delta\sigma \subseteq \Delta'$.[25] La regola **Sub** di sussunzione dice che, se si sono ottenute entrambe le clausole $\Gamma \Rightarrow \Delta$ e $\Gamma' \Rightarrow \Delta'$, allora si può eliminare la clausola $\Gamma' \Rightarrow \Delta'$ qualora $\Gamma \Rightarrow \Delta$ la sussuma.

(3) Una terza regola di riduzione è la regola **MRR** ('Matching Replacement Resolution') implementata ad esempio su SPASS. Tale regola è un misto fra una regola di inferenza e una regola di riduzione e opera nel seguente modo. Supponiamo di aver già dedotto le clausole[26]

$$\Gamma_1 \Rightarrow \Delta_1, A_1 \qquad \Gamma_2, A_2 \Rightarrow \Delta_2$$

[25] Trattandosi di multiinsiemi, la notazione $\Gamma\sigma \subseteq \Gamma'$ significa che ogni formula atomica che occorre in $\Gamma\sigma$ occorre anche in Γ' e in un numero maggiore o uguale di volte.

[26] Questa è la versione destra della regola, ovviamente ne esiste anche una versione sinistra (del tutto simmetrica) in cui A_1 compare sullla sinistra di $\Rightarrow$ e A_2 compare sulla destra di $\Rightarrow$.

e supponiamo anche che esista un matcher σ soddisfacente le seguenti condizioni: a) $A_1\sigma \equiv A_2$; b) $\Gamma_1\sigma \subseteq \Gamma_2$; c) $\Delta_1\sigma \subseteq \Delta_2$. In tali condizioni possiamo (mantenendo la clausola $\Gamma_1 \Rightarrow \Delta_1, A_1$) cancellare la clausola $\Gamma_2, A_2 \Rightarrow \Delta_2$ e *sostituirla* con la clausola più semplice $\Gamma_2 \Rightarrow \Delta_2$.

Sono possibili molti altri tipi di regole di riduzione[27] e noi stessi ne aggiungeremo altre importanti per i calcoli che presenteremo in seguito. Vediamo ora un semplice schema di implementazione a livello globale. Il sistema mantiene due insiemi di clausole, l'insieme *Wo* delle clausole **già elaborate** ('worked off') e l'insieme *Us* delle clausole **utilizzabili** ('usable'). All'inizio, *Wo* è vuoto e *Us* contiene le clausole generate dal processo di skolemizzazione che vengono anche inter-ridotte (ossia vengono usate subito le regole di riduzione per semplificarle, se possibile). A questo punto viene eseguito il seguente ciclo ('loop') di istruzioni. Dal ciclo si esce in due casi: a) quando *Us* contiene la clausola vuota, nel qual caso il sistema trasmette all'utente un messaggio del tipo **'Proof Found'**; b) quando *Us* è vuota, nel qual caso il sistema trasmette all'utente un messaggio del tipo **'Completion Found'** (ovviamente, il primo caso coincide con il caso in cui le clausole in ingresso sono insoddisfacibili, mentre il secondo caso coincide con il caso in cui le clausole in ingresso sono soddisfacibili e il processo di saturazione termina). Le istruzioni del ciclo sono le seguenti:

1. Si sceglie all'interno di *Us* (mediante un'opportuna funzione di scelta su cui ritorneremo) una clausola, detta **clausola data** ('Given Clause'), che viene aggiunta all'insieme *Wo*.[28]

[27] Si veda ad esempio la documentazione di SPASS. Si noti che SPASS utilizza anche regole di inferenza non previste nelle presenti note, per ottenere prestazioni migliori in certe situazioni. Tali regole di inferenza/riduzione aggiuntive **possono essere disabilitate** mediante opportune opzioni (si consiglia lo studente alle prime armi di disabilitare regole che possono creare problemi di chiarezza e di comprensione delle dimostrazioni che compaiono sul video). Ad esempio, consigliamo di disabilitare il meccanismo delle inferenze sortate (l'opzione relativa è -Sorts=0) e il meccanismo degli splittings (l'opzione relativa è -Splits=0). Tuttavia, siccome in molti casi l'uso di inferenze e soprattutto di riduzioni aggiuntive migliora in modo molto sensibile le prestazioni, lo studente è incoraggiato, man mano che prende maggiore confidenza con il prover, a *studiarne accuratamente la documentazione*, in modo da non aver più bisogno di disabilitare inferenze e riduzioni che non conosce.

[28] La lista delle clausole date viene automaticamente visualizzata nell'output di SPASS. Per visualizzare l'intero elenco delle clausole dedotte si usi l'opzione -PDer=1.

2. Si applicano tutte le inferenze possibili fra la clausola data, se stessa e le clausole in *Wo*, generando un insieme *New* di clausole nuove.

3. Si applica la riduzione in avanti per semplificare l'insieme *New* (tramite clausole in *New*, poi in *Wo* ed infine anche in *Us*).

4. Si applica la riduzione all'indietro per semplificare le clausole in *Wo* e poi anche quelle in *Us* tramite le clausole in *New*.

5. Si aggiungono le clausole in *New* alle clausole in *Us*.

6. Si torna al punto 1.

Siccome l'insieme *Us* cresce molto rapidamente, talvolta si preferisce non implementare (o lasciare la scelta all'utente tramite un'opportuna opzione) i test di riduzione che coinvolgano le clausole in *Us*.

La funzione di scelta della clausola data può essere arbitraria, ma deve sottostare ad un requisito di **equità** ('fairness'), cioè deve evitare che una clausola stazioni all'infinito nell'insieme *Us* senza avere mai nessuna possibilità di essere selezionata (solo in tal modo la completezza refutazionale viene preservata). Usualmente si sceglie la clusola più piccola (alternativamente si può scegliere in base alla profondità dell'albero delle inferenze che genera la clausola, o del numero delle variabili, o infine si possono mescolare questi criteri in base ad una ratio predefinita). Può succedere che più di una clausola soddisfi i requisiti per essere scelta, in tal caso si sceglie in un qualche modo convenzionale.

ESEMPIO.[29] Consideriamo l'insieme dato dalle tre seguenti clausole:

1. $\Rightarrow P(f(a))$
2. $P(f(x)) \Rightarrow P(x)$
3. $P(f(a)), P(f(x)) \Rightarrow$

Durante il *primo ciclo* di esecuzione Wo è vuoto, Us contiene le clausole 1,2,3 e la clausola 1 viene selezionata come clausola data (è la più piccola). Non si producono nuove clausole. Durante il *secondo ciclo* di esecuzione Wo contiene la clausola 1, Us contiene le clausole 2,3 e la clausola 2 viene selezionata. Vengono prodotte le nuove clausole

[29]Per questo esempio, ci siamo basati su un articolo di C. Weidenbach, inserito anche nella documentazione di SPASS.

4. $\Rightarrow P(a)$ $[Res\ 1.1; 2.1]$
5. $P(f(f(x))) \Rightarrow P(x)$ $[Res\ 2.1; 2.2]$

Durante il *terzo ciclo* di esecuzione Wo contiene le clausole 1, 2 e Us contiene le clausole 3,4,5; la clausola 4 è la clausola data, ma non si generano clausole nuove. Durante il *quarto ciclo* di esecuzione Wo contiene le clausole 1, 2, 4 e Us contiene le clausole 3,5; la clausola 3 viene selezionata come clausola data.[30] Si generano cinque clausole nuove:

6. $P(f(x)) \Rightarrow$ $[Res\ 1.1; 3.1]$
7. $P(f(a)) \Rightarrow$ $[Res\ 1.1; 3.2]$
8. $P(f(f(a))), P(f(x)) \Rightarrow$ $[Res\ 2.2; 3.1]$
9. $P(f(a)), P(f(f(x))) \Rightarrow$ $[Res\ 2.2; 3.2]$
10. $P(f(a)) \Rightarrow$ $[Fac\ 3.1; 3.2]$

(si noti che la 7 e la 10 sono identiche, ma ottenute in due modi diversi). Ora i test di riduzione eliminano le clausole 2,3,5,7,8,9,10 che sono tutte sussunte dalla 6. Durante il *quinto ciclo* di esecuzione Wo contiene le clausole 1, 4 e Us contiene la clausola 6, che viene ovviamente selezionata come clausola data. Poichè la clausola vuota viene immediatamente inferita (dalle clausole 1 e 6 per Risoluzione), il sistema si arresta e comunica di aver trovato una dimostrazione di inconsistenza.

[30] In base al criterio di piccolezza, si poteva selezionare anche la 5; il sistema avrebbe impiegato un ciclo in più per finire.

5 Ordinamenti

In questo capitolo illustriamo le tecniche di ordinamento dei termini e ne vediamo una prima applicazione alla risoluzione ordinata.

5.1 Ordini di riduzione

Cominciamo col rivedere alcune definizioni di base.

Un **ordine stretto** è un insieme P dotato di una relazione $>$ che sia irriflessiva (non vale $x > x$ per nessun $x \in P$) e transitiva ($x > y$ e $y > z$ implicano $x > z$ per ogni $x, y, z \in P$). Se $(P, >)$ è un ordine stretto, con $x \geq y$ intendiamo $x > y \vee x = y$ (in questo modo $\leq$ risulta essere riflessiva oltre che transitiva). L'ordine stretto è detto **terminante** qualora non esistano catene infinite

$$x_0 > x_1 > x_2 > \cdots$$

Un esempio di ordine stretto terminante è dato dai numeri naturali con la relazione di 'maggiore in senso stretto' (si noti che in questo esempio $>$ è anche **totale**, ossia vale sempre $x > y \vee x = y \vee y > x$ per ogni x, y).

Si possono combinare fra loro ordini stretti (terminanti) e ottenere altri ordini stretti (terminanti) mediante alcuni schemi che indichiamo qui di seguito:

- *Prodotti lessicografici*: se $(P_1, >_1)$ e $(P_2, >_2)$ sono due ordini stretti terminanti, tale risulta $(P_1 \times P_2, >)$ dove

  $$(x, y) > (x', y') \qquad \Leftrightarrow \qquad x >_1 x' \vee (x = x' \wedge y >_2 y').$$

  Ossia, negli ordinamenti lessicografici a due componenti si confrontano dapprima le prime componenti e, solo nel caso in cui queste siano uguali, si procede al confronto delle seconde componenti. Il prodotto lessicografico si estende facilmente al caso di $n \geq 2$ componenti.

- *Prodotti lessicografici su liste*: sia $(P, >)$ un ordine stretto terminante e sia P^* l'insieme delle liste finite di elementi di P. Si ottiene un altro ordine stretto terminante $(P^*, >^*)$ mediante la seguente definizione:

  $$\begin{aligned}(x_1, \ldots, x_n) >^* (y_1, \ldots, y_m) \quad \Leftrightarrow \quad & n > m \vee (n = m \wedge \exists i < n \, (x_1 = y_1 \wedge \\ & \wedge \cdots \wedge x_i = y_i \wedge x_{i+1} > y_{i+1})).\end{aligned}$$

Ossia si confrontano dapprima la lunghezza e poi (finchè è necessario) nell'ordine le componenti da sinistra a destra.

- *Ordinamento di multiinsiemi*: sia $(P, >)$ un ordine stretto terminante e sia $Mul(P)$ l'insieme dei multiinsiemi finiti su P. Questi ultimi possono essere definiti come funzioni a valori nei numeri naturali $M : P \longrightarrow \mathbf{N}$ tali $M(x) \neq 0$ vale solo per un numero finito di $x \in P$ ($M(x)$ dice 'quante volte' x compare in M). Usiamo la notazione $M \cup N$ per indicare l'unione dei multiinsiemi M ed N: usando il formalismo delle funzioni a valori nei numeri naturali, avremo $(M \cup N)(x) = M(x) + N(x)$ per ogni $x \in P$.[31] Definiamo la relazione $M >^1 N$: essa vale quando M è del tipo $M' \cup \{x\}$ e N è del tipo $M' \cup \{y_1, \ldots, y_n\}$ $(n \geq 0)$ e inoltre si ha che $x > y_1, \ldots, x > y_n$ (cioè N è ottenuto da M rimpiazzando un'occorrenza di un elemento con un multiinsieme di elementi tutti minori). Ad esempio abbiamo che (se $(P, >)$ è l'insieme dei numeri naturali con la relazione di maggiore in senso stretto)

$$\{3, 1, 1\} >^1 \{2, 2, 2, 2, 1, 1, 1\}.$$

Se ora poniamo per $M, N \in Mul(P)$

$$\begin{aligned} M > N \quad \Leftrightarrow \quad & \exists\, n \geq 1, \exists\, M_0, \ldots, M_n \in Mul(P) \\ & M = M_0 >^1 M_1 >^1 \cdots >^1 M_n = N \end{aligned}$$

otteniamo che $(Mul(P), >)$ è ancora un ordine stretto terminante (totale, qualora $(P, >)$ fosse già totale in partenza).

Sia ora $\mathcal{L}$ un linguaggio del primo ordine; ci interessano ordini stretti $>$ sull'insieme $T_{\mathcal{L}}$ degli $\mathcal{L}$-termini che godano di particolari proprietà. Diciamo che $>$ è un **ordine di riscrittura** qualora soddisfi le due condizioni seguenti:

(i) se $f \in \mathcal{F}_n$ e $s_1 > s_2$, allora

$$f(t_1, \ldots, t_{i-1}, s_1, t_{i+1}, \ldots, t_n) > f(t_1, \ldots, t_{i-1}, s_2, t_{i+1}, \ldots, t_n)$$

per ogni $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \in T_{\mathcal{L}}$.

[31] Questo significa che x occorre in $M \cup N$ tante volte quante ne occorre in M più tante volte quante ne occorre in N. Per esempio $\{x, y, y\} \cup \{x, y, z\} = \{x, x, y, y, y, z\}$.

(i) se $s_1 > s_2$ e σ è una sostituzione, allora

$$s_1\sigma > s_2\sigma.$$

Un **ordine di riduzione** è un ordine di riscrittura terminante. Un **ordine di semplificazione** è un ordine di riscrittura tale che $f(x_1, \ldots, x_n) > x_i$ vale per ogni $f \in \mathcal{F}_n$. Il seguente teorema è un risultato chiave nel verificare per esempio che gli ordinamenti che introdurremo sono ordini di riduzione:

Teorema 5.1. *(conseguenza di un risultato noto come 'teorema di Kruskal') Se $\mathcal{L}$ contiene solo un numero finito di simboli di funzione, ogni ordine di semplificazione su $\mathcal{L}$ è anche un ordine di riduzione.*

Diamo ora due famiglie di ordini di semplificazione molto usate nei dimostratori automatici correnti (tali famiglie hanno anche la proprietà molto rilevante di essere ordini **totali sui termini ground**). Nel seguito di questo paragrafo assumiamo che $\mathcal{L}$ contenga solo un numero finito di simboli di funzione $f_1, \ldots, f_k$ e che sia dato un ordine totale stretto (detto di **precedenza**)[32]

$$f_1 >_p f_2 >_p \cdots >_p f_k$$

fra di essi.

Definizione 5.2. *L'ordinamento $s >_{lpo} t$ (**'lexicographic path order'**)[33] indotto dalla precedenza $>_p$ su $T_{\mathcal{L}}$ è così definito per casi (per induzione su $|s| + |t|$, cioè sulla somma delle lunghezze di s e t):*

[32] **Osservazione**: per forzare una precedenza in SPASS (diversa da quella assunta per default dal prover) occorre procedere nel seguente modo. Supponiamo che il linguaggio contenga 3 simboli di funzione f, g, h; per impostare la precedenza $f >_p g >_p h$ occorre inserire nel file di input (sopra la riga finale 'end_problem.') la seguente dicitura:

```
list_of_settings(SPASS).
{*
set_precedence(f, g, h).
*}
end_of_list.
```

[33] Questo tipo di ordinamento fa parte di una famiglia più ampia, detta famiglia RPO ('recursive path orders'). In SPASS, l'opzione -Ordering=1 forza il prover ad adottare un ordinamento del tipo LPO.

(LPO1) $s \equiv f(s_1, \ldots, s_n)$ *e per qualche* $i = 1, \ldots, n$ *vale che* $s_i > t$ *oppure che* $s_i \equiv t$*;*

(LPO2) $s \equiv f(s_1, \ldots, s_n)$, $t \equiv g(t_1, \ldots, t_m)$, $f >_p g$ *e inoltre* $s >_{lpo} t_i$ *per ogni* $i = 1, \ldots, m$*;*

(LPO3) $s \equiv f(s_1, \ldots, s_n)$, $t \equiv f(t_1, \ldots, t_n)$ *e per qualche* $i = 1, \ldots, n$ *vale che*

$$s_1 \equiv t_1, \ldots, s_{i-1} \equiv t_{i-1}, s_i >_{lpo} t_i, s >_{lpo} t_{i+1}, \ldots, s >_{lpo} t_n.$$

ESEMPIO. Usando la precedenza $a >_p s >_p 0$, possiamo verificare che[34]

$$a(0, y) >_{lpo} s(y), \quad a(s(x), 0) >_{lpo} a(x, s(0)), \quad a(s(x), s(y)) > a(x, a(s(x), y)).$$

Vediamo in dettaglio la verifica per la terza coppia. Per stabilire che vale la relazione $a(s(x), s(y)) > a(x, a(s(x), y))$, usiamo (LPO3) e verifichiamo che

$$(a) \quad s(x) > x, \qquad (b) \quad a(s(x), s(y)) > a(s(x), y).$$

(a) è immediato per (LPO1); per (b) usiamo ancora (LPO3) e verifichiamo che

$$(c) \quad s(x) \equiv s(x), \qquad (d) \quad s(y) > y.$$

Ora (c) è ovvia e (d) si ottiene da una applicazione di (LPO1).

Facciamo notare che, nonostante le complicazioni ricorsive della definizione, la verifica di $s >_{lpo} t$ è veloce (richiede tempo $O(|s| \cdot |t|)$).

Per introdurre la prossima famiglia di ordinamenti, abbiamo bisogno, oltre che di un ordine totale stretto di precedenza $>_p$ fra i simboli di funzione, anche di una **funzione peso** w che associa ad ogni simbolo di funzione e ad ogni variabile un numero reale positivo. La funzione peso w deve essere *ammissibile*, ossia deve soddisfare i seguenti requisiti:

- deve esistere $d > 0$ tale che $d = w(x)$ per ogni variabile x (cioè il peso delle variabili è sempre lo stesso numero reale positivo d) e inoltre deve valere $w(c) \geq d$ per ogni costante c di $\mathcal{L}$;
- può valere $w(f) = 0$, ma per un solo simbolo di funzione unario f e in tal caso deve essere $f >_p g$ per ogni altro simbolo di funzione o di costante g di $\mathcal{L}$.

[34] Queste tre coppie di termini, orientate da sinistra a destra secondo l'ordinamento $>_{lpo}$, danno un sistema di riscrittura convergente che calcola la funzione di Ackermann.

La funzione peso viene estesa a tutti i termini di $\mathcal{L}$, ponendo

$$w(t) = \sum_{x \in \mathcal{V}} w(x)|t|_x + \sum_{n \geq 0, f \in \mathcal{F}_n} w(f)|t|_f$$

dove $|t|_\alpha$ indica il numero di occorrenze di α (variabile o simbolo di funzione che sia) nel termine t.

Definizione 5.3. *L'ordinamento $s >_{kbo} t$ (**'Knuth-Bendix order'**)*[35] *su $T_\mathcal{L}$ indotto dalla precedenza $>_p$ e dalla funzione peso ammissibile w è così definito. Affinchè si abbia $s >_{kbo} t$ deve valere innanzitutto $|s|_x \geq |t|_x$ per ogni variabile x e inoltre deve verificarsi una delle seguenti condizioni:*

(KBO1) $w(s) > w(t)$;

(KBO2) $w(s) = w(t)$, $s \equiv f^n(x)$ *e* $t \equiv x$, *per qualche* $f \in \mathcal{F}_1$[36] *e per qualche variabile* x;

(KBO3) $w(s) = w(t)$, $s \equiv h(s_1, \ldots, s_n)$, $t \equiv g(t_1, \ldots, t_m)$, *con* $h >_p g$;

(KBO4) $w(s) = w(t)$, $s \equiv g(s_1, \ldots, s_n)$, $t \equiv g(t_1, \ldots, t_n)$ *e per qualche* $i = 1, \ldots, n$ *vale che*

$$s_1 \equiv t_1, \ldots, s_{i-1} \equiv t_{i-1}, s_i >_{kbo} t_i.$$

ESEMPIO. Usando la precedenza $i >_p *$ e la funzione peso $w(i) = 0, w(*) = w(x) = 1$ (per ogni variabile x), verifichiamo che $i(x * y) > i(y) * i(x)$. Ogni variabile occorre lo stesso numero di volte nei due termini e il peso è lo stesso. Siccome $i >_p *$, abbiamo proprio $i(x * y) > i(y) * i(x)$ per (KBO3).

Nella pratica, gli ordinamenti $>_{kbo}$ danno migliori risultati nei dimostratori automatici, tuttavia gli ordinamenti $>_{lpo}$ sono indispensabili per orientare nel senso voluto equazioni come le leggi distributive, in cui il termine minore contiene più occorrenze delle stessa variabile del termine maggiore (cosa che non è consentita dagli ordinamenti del tipo $>_{kbo}$).

[35] In SPASS, l'opzione (di default) -Ordering=0 forza il prover ad adottare un ordinamento del tipo KBO.

[36] Poichè si ha $w(s) = w(t)$, tale f non può che essere l'eventuale e unico simbolo unario di peso 0.

5.2 Letterali massimali in una clausola

Nel seguito spesso converrà pensare le formule atomiche $P(t_1, \ldots, t_n)$ il cui predicato in radice non è l'uguaglianza, come se fossero esse stesse equazioni, più precisamente equazioni del tipo $p(t_1, \ldots, t_n) = true$, dove '$true$' è una costante fittizia (che si suppone minima nell'ordinamento dei termini) e p è a sua volta un simbolo di funzione fittizio di arietà n. In tal modo, la trattazione diventa più uniforme, specialmente quando si tratta di confrontare fra loro termini, letterali e clausole rispetto ad un ordine di riduzione. Procedendo così, nel calcolo $\mathcal{S}$ che vedremo nel prossimo capitolo, si possono addirittura eliminare le regole di Risoluzione e Fattorizzazione Destra (che diventano casi particolari delle Regole di Sovrapposizione Destra e di Fattorizzazione per l'Uguaglianza).[37] In tal modo, possiamo supporre che le nostre clausole abbiano la forma

$$(C) \qquad t_1 = s_1, \ldots, t_n = s_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m.$$

Tale (C) avrà i $t_i = s_i$ $(i = 1, \ldots, n)$ come letterali negativi e gli $u_j = v_j$ $(j = 1, \ldots, m)$ come letterali positivi.

Per procedere al confronto dei letterali nella (C), fissiamo un ordine di riduzione $>$ totale sui termini ground. Associamo ad ogni letterale negativo $L \equiv (t_i = s_i)$ il multiinsieme $m(L) = \{t_i, t_i, s_i, s_i\}$ e ad ogni letterale positivo $L \equiv (u_j = v_j)$ il multiinsieme $m(L) = \{u_j, v_j\}$ (si noti che i termini nei letterali negativi vengono raddoppiati).

Diciamo che il letterale L della (C) è **massimale** qualora non esista un letterale L' nella (C) tale che $m(L') > m(L)$ (qui ovviamente facciamo riferimento all'estensione di $>$ ai multiinsiemi). Diciamo che il letterale L della (C) è **strettamente massimale** qualora L sia massimale in (C) e occorra una sola volta nella (C).

ESEMPIO. Usiamo un LPO indotto dalla precedenza $f > g > P > a > b$ e consideriamo la clausola:

$$P(x), f(x) = g(x) \Rightarrow f(y) = g(a).$$

Abbiamo $\{f(x), f(x), g(x), g(x)\} > \{P(x), P(x), true, true\}$ per cui $P(x)$ non è massimale. Siccome $\{f(x), f(x), g(x), g(x)\}$ e $\{f(y), g(a)\}$ non sono confrontabili, il letterale negativo $f(x) = g(x)$ e il letterale positivo $f(y) = g(a)$ sono entrambi massimali. Se consideriamo

[37] Per una giustificazione formale precisa di questo modo di procedere - frequente nella letteratura - occorrerebbe introdurre linguaggi a due sorte.

invece la clausola

$$f(a) = g(a) \Rightarrow f(a) = f(b)$$

abbiamo che solo $f(a) = g(a)$ è massimale in quanto $\{f(a), f(a), g(a), g(a)\} > \{f(a), f(b)\}$.

5.3 La Risoluzione Ordinata

Vediamo come trarre un primo vantaggio dagli ordinamenti per ridurre lo spazio di ricerca nelle prove refutative e riesaminiamo il calcolo $\mathcal{R}$ per linguaggi senza identità.

Tale calcolo ha sempre le regole di Risoluzione a Fattorizzazione Destra come regole di inferenza, ma **ora ne restringiamo l'applicazione ai soli letterali massimali**. Più in dettaglio, la

Regola di Risoluzione Ordinata

$$\frac{\Gamma \Rightarrow \Delta, A \qquad B, \Gamma' \Rightarrow \Delta'}{\Gamma\mu, \Gamma'\mu \Rightarrow \Delta\mu, \Delta'\mu}$$

si applica solo qualora (detto μ un upg del problema di unificazione $A \stackrel{?}{=} B$) *il letterale positivo* $A\mu$ *sia strettamente massimale in* $\Gamma\mu \Rightarrow \Delta\mu, A\mu$ *e qualora il letterale negativo* $B\mu$ *sia massimale in* $\Gamma'\mu, B\mu \Rightarrow \Delta'\mu$.

Analogamente la

Regola di Fattorizzazione Destra Ordinata

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma\mu \Rightarrow \Delta\mu, A\mu}$$

si applica solo qualora (detto μ un upg del problema di unificazione $A \stackrel{?}{=} B$) *il letterale positivo* $A\mu$ *sia massimale in* $\Gamma\mu \Rightarrow \Delta\mu, A\mu, B\mu$.

ESEMPIO. Riprendiamo l'esempio delle due clausole

1. $\Rightarrow P(c)$
2. $P(x) \Rightarrow P(f(x))$

già visto in precedenza. Questo insieme di clausole è saturo se si usa la Risoluzione Ordinata: nessuna inferenza è infatti eseguibile perchè il letterale negativo $P(x)$ non è massimale nella clausola $P(x) \Rightarrow P(f(x))$ (qualunque ordine di semplificazione si applichi).

Nei dimostratori automatici come SPASS, i letterali massimali vengono *memorizzati mediante un asterisco*: in tal modo, vengono prese in considerazione solo le inferenze che coinvolgono i letterali asteriscati (va tuttavia segnalato che, in presenza di più letterali asteriscati, il vincolo di massimalità dovrebbe essere riverificato *a posteriori* dopo l'applicazione dell'upg previsto dalla regola).

L'ordinamento è solo una delle tecniche utilizzate per ridurre lo spazio di ricerca delle prove refutative e per aumentare i casi di terminazione (più avanti vedremo un'altra tecnica basata sulla selezione di letterali negativi).

6 Paramodulazione e Sovrapposizione

In questo capitolo ritorniamo a trattare linguaggi del primo ordine arbitrari, cioè **con identità**. L'identità potrebbe essere trattata nel contesto dei calcoli basati sulla regola di Risoluzione del capitolo 4.2 aggiungendo sempre alle teorie che si testano per la consistenza le clausole del tipo seguente, corrispondenti agli assiomi di riflessività e congruenza che l'identità deve soddisfare:

$$\Rightarrow x = x$$
$$x = y, A[x/z] \Rightarrow A[y/z]$$

(nell'ultima clausola si suppone che A sia una formula atomica). Tuttavia il calcolo che si ottiene in questo modo è terribilmente inefficiente. Un miglioramento considerevole si ha con i calcoli basati sulla regola di **Paramodulazione** (introdotta da Robinson e Wos nel 1969). Il senso della regola di Paramodulazione risale al principio leibniziano dell'*indiscernibilità degli identici* (due enti identici hanno le stesse proprietà); tuttavia tale principio deve essere formulato in maniera adeguata per essere computazionalmente efficace. Nella sua formulazione più semplice, la regola di Paramodulazione è la seguente[38]

$$\frac{\Gamma \Rightarrow \Delta, s = t \qquad \Gamma' \Rightarrow \Delta', A}{\Gamma\mu, \Gamma'\mu \Rightarrow \Delta\mu, \Delta'\mu, (A[t]_p)\mu}$$

dove μ è upg di $A_{|p} \stackrel{?}{=} s$. Tuttavia, questa formulazione è inadeguata per vari motivi, ad esempio richiede l'aggiunta di assiomi di funzionalità riflessiva e consente la dispendiosa paramodulazione sulle variabili. Miglioramenti sono stati ottenuti da vari autori in epoche molto successive, fino agli anni '90 quando si sono introdotte nei calcoli basati sulla Paramodulazione le idee maturate nel settore dei sistemi di riscrittura. Il risultato finale di tali miglioramenti è il calcolo $\mathcal{S}$ che studieremo nel presente capitolo: tale calcolo (di cui riportiamo una delle versioni esistenti, che sono solo leggermente diverse fra loro) sostanzialmente interviene sulla regola di Paramodulazione introducendo vincoli alla sua applicabilità tramite gli ordinamenti. Nelle analisi sperimentali, i dimostratori automatici che implementano il calcolo $\mathcal{S}$ si sono rivelati di gran lunga più efficaci dei precedenti basati sulla semplice regola di Paramodulazione.

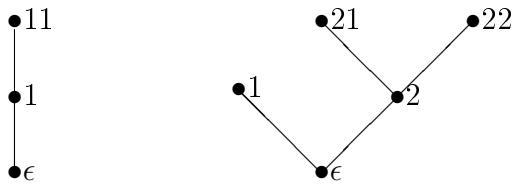
[38] Le notazioni $A_{|p}, A[t]_p$ verranno spiegate nel paragrafo 6.1 seguente.

6.1 Alberi e Posizioni

Facciamo alcune precisazioni sugli alberi. Con N indichiamo l'insieme dei numeri naturali (zero escluso) e con N^* l'insieme delle liste finite di numeri naturali (inclusa la lista vuota). Tali liste vengono indicate con le lettere $p, q, r, \ldots$.

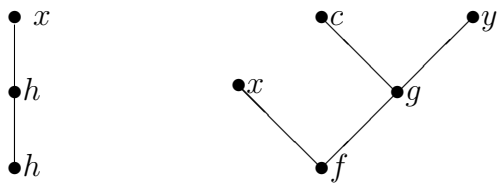
Un **albero** T è un sottoinsieme non vuoto di N^* (cioè un insieme di liste di numeri naturali) con le seguenti proprietà: i) se $p \in T$ e $p = qr$, allora $q \in T$; ii) se $pi \in T$ e $j < i$, allora $pj \in T$.

Per esempio, i due alberi della figura sottostante corrispondono rispettivamente agli insiemi di liste $\{\epsilon, 1, 11\}$ e $\{\epsilon, 1, 2, 21, 22\}$ (dove con ϵ indichiamo la lista vuota). Il nodo 21 dell'albero (b) viene così indicato perchè il percorso per raggiungerlo dalla radice consiste nel prendere il secondo nodo della biforcazione e poi il primo nodo della biforcazione successiva. Lette in questo modo, le condizioni i) e ii) della definizione di albero risultano trasparenti: la condizione i) dice che se l'albero contiene il percorso che arriva al nodo p deve contenere ogni segmento iniziale di tale percorso, mentre la condizione ii) dice ad esempio che, se la biforcazione al nodo p contiene il terzo successore $p3$, allora deve contenere anche il primo ed il secondo dei successori, cioè $p1$ e $p2$.



(a) (b)

I termini di un liguaggio $\mathcal{L}$ possono essere utilmente rappresentati come alberi i cui nodi sono etichettati da variabili o da simboli di funzione. Ad esempio, se f e g sono simboli di funzioni binarie, h è un simbolo di funzione unario e c è una costante, gli alberi



rappresentano i termini $h(h(x))$ e $f(x, g(c, y))$.

Formalmente, associamo ad ogni termine t un albero, detto l'albero delle **posizioni** $Pos(t)$ del termine t nel modo seguente:

- $Pos(t) = \{\epsilon\}$, se t è una variabile o una costante;

- $Pos(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \,|\, p \in Pos(t_i)\}$, se t è del tipo $f(t_1, \ldots, t_n)$.

Ad esempio, gli alberi delle posizioni dei termini $h(h(x))$ e $f(x, g(c, y))$ sono proprio gli alberi (a) e (b) visti in precedenza.

Se $p \in Pos(t)$, il sottotermine di t nella posizione p viene notato con $t_{|p}$ ed è così definito:[39]

- $t_{|\epsilon} \equiv t$;

- $t_{|iq} \equiv s_{i|q}$, se t è del tipo $f(s_1, \ldots, s_n)$.

Ad esempio, il sottotermine in posizione 2 del termine $f(x, g(c, y))$ è il termine $g(c, y)$.

Se $p \in Pos(s)$ e t è un altro termine, con la notazione $s[t]_p$ si indica il termine ottenuto da s rimpiazzando, nella posizione p, $s_{|p}$ con t. Formalmente, abbiamo la seguente definizione:

- $s[t]_\epsilon \equiv t$;

- $f(s_1, \ldots, s_n)[t]_{iq} \equiv f(s_1, \ldots, s_i[t]_q, \ldots, s_n)$, se s è del tipo $f(s_1, \ldots, s_n)$.

Ad esempio $f(x, g(c, y))[h(c)]_2$ è uguale a $f(x, h(c))$.

Ci capiterà di utilizzare le definizioni che abbiamo dato anche per le formule atomiche, oltre che per i termini. La relativa estensione è ovvia: la formula atomica $P(t_1, \ldots, t_n)$ ha in radice il simbolo P, in posizione 1 il termine t_1, ..., in posizione n il termine t_n, in posizione $i1$ il sottotermine in posizione 1 di t_i, ecc.

6.2 Il calcolo S

Ci occuperemo di linguaggi del primo ordine contenenti **solo l'identità** come simbolo di predicato (come osservato nel paragrafo 5.2, scriveremo le formule atomiche del tipo

[39] Usiamo $\equiv$ per indicare la coincidenza fisica di due termini (preferiamo non usare $=$ perchè il simbolo $=$ è già usato per il predicato binario dell'uguaglianza).

$P(t_1, \ldots, t_n)$ nella forma $p(t_1, \ldots, t_n) = true$, dove p è un nuovo simbolo di funzione n-ario e $true$ una nuova costante). Tratteremo inoltre le formule atomiche del tipo $s = t$ come se fossero il **multiinsieme** $\{s, t\}$: in tal modo, quando scriviamo $s = t$ possiamo intendere sia proprio l'equazione $s = t$, che anche l'equazione $t = s$ (così otteniamo una notazione più compatta e pur sempre trasparente).

Supponiamo fissato **un ordine di riduzione** $>$ totale sui termini ground di $\mathcal{L}$ (ad esempio un $>_{lpo}$ o un $>_{kbo}$). Le regole di inferenza che vedremo nel presente paragrafo avranno la seguente forma:

$$\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma\mu \Rightarrow \Delta\mu \;\; \| \;\; V}$$

dove μ è upg di un certo problema di unificazione e V è un vincolo di ordinamento (ad esempio, V esprimerà condizioni di massimalità di certi letterali). Per applicare la regola, occorre **mostrare che il vincolo è soddisfatto.**[40]

Le regole che utilizziamo nel calcolo $\mathcal{S}$ sono le quattro seguenti:[41]

Regola di Sovrapposizione Destra

$$\frac{\Gamma' \Rightarrow \Delta', l = r \qquad \Gamma \Rightarrow s = t, \Delta}{\Gamma\mu, \Gamma'\mu \Rightarrow (s[r]_p)\mu = t\mu, \Delta\mu, \Delta'\mu \;\; \| \;\; VD}$$

dove $s_{|p}$ non è una variabile, μ è upg del problema di unificazione $s_{|p} \stackrel{?}{=} l$ e dove VD è il seguente vincolo:

- $l\mu \not\leq r\mu$ e $s\mu \not\leq t\mu$;

- $l\mu = r\mu$ è strettamente massimale in $\Gamma'\mu \Rightarrow \Delta'\mu, l\mu = r\mu$;

- $s\mu = t\mu$ è strettamente massimale in $\Gamma\mu \Rightarrow \Delta\mu, t\mu = s\mu$.

[40] Sono possibili metodi raffinati per applicare queste regole, che fanno riferimento a nozioni precise di 'vincoli di ordinamento' e ad algoritmi specifici di soluzione di tali vincoli. Nelle versioni 'basiche' del calcolo poi, i vincoli vengono accumulati e gli upg non vengono applicati alle conclusioni delle inferenze.

[41] Ricordiamo che, se una regola ha più premesse, queste devono avere **variabili disgiunte fra loro** (nel caso che questo non si verifichi, occorre operare una rinomina preventiva).

Regola di Sovrapposizione Sinistra

$$\frac{\Gamma' \Rightarrow \Delta', l = r \qquad \Gamma, s = t \Rightarrow \Delta}{\Gamma\mu, \Gamma'\mu, (s[r]_p)\mu = t\mu \Rightarrow \Delta\mu, \Delta'\mu \;\; \| \;\; VS}$$

dove $s_{|p}$ non è una variabile, μ è upg del problema di unificazione $s_{|p} \stackrel{?}{=} l$ e dove VS è il seguente vincolo:

- $l\mu \not\leq r\mu$ e $s\mu \not\leq t\mu$;
- $l\mu = r\mu$ è strettamente massimale in $\Gamma'\mu \Rightarrow \Delta'\mu, l\mu = r\mu$;
- $s\mu = t\mu$ è massimale in $\Gamma\mu, t\mu = s\mu \Rightarrow \Delta\mu$.

Regola di Risoluzione per l'Uguaglianza

$$\frac{s = t, \Gamma \Rightarrow \Delta}{\Gamma\mu \Rightarrow \Delta\mu \;\; \| \;\; V_1}$$

dove μ è upg del problema di unificazione $s \stackrel{?}{=} t$ e V_1 è il seguente vincolo:

- $s\mu = t\mu$ è massimale in $\Gamma\mu, t\mu = s\mu \Rightarrow \Delta\mu$.

Regola di Fattorizzazione per l'Uguaglianza

$$\frac{\Gamma \Rightarrow s = t, s' = t', \Delta}{\Gamma\mu, t\mu = t'\mu \Rightarrow s\mu = t'\mu, \Delta\mu \;\; \| \;\; V_2}$$

dove μ è upg del problema di unificazione $s \stackrel{?}{=} s'$ e V_2 è il seguente vincolo:

- $s\mu = t\mu$ è massimale in $\Gamma\mu \Rightarrow s\mu = t\mu, s'\mu = t'\mu, \Delta\mu$.

L'uso dei vincoli realizza una serie di intuizioni su come restringere l'applicazione delle regole. Una di queste intuizioni consiste nell'utilizzare per le inferenze solo letterali massimali, a preferenza dei rimanenti. Inoltre, nelle Regole di Sovrapposizione, l'identità è trattata in modo *non simmetrico* ed è consentito operare solo sui termini massimali delle equazioni stesse.[42]

[42] Il parallelo con la teoria dei sistemi di riscrittura ordinata è immediato: nella terminologia di tale teoria, l'equazione $s[r]_p = t$ che compare nel conseguente delle Regola di Sovrapposizione Destra altri non è che una *coppia critica estesa* delle equazioni $s = t$ e $l = r$.

La Regola di Fattorizzazione per l'Uguaglianza è un lieve rafforzamento della regola di Fattorizzazione Destra del paragrafo 4.2. Anche per il calcolo $\mathcal{S}$ abbiamo il

Teorema 6.1. *Il calcolo $\mathcal{S}$ è refutazionalmente completo.*

La dimostrazione del precedente teorema è molto complessa. Diamo solo acune idee molto sommarie relativamente alla tecnica (detta di 'model generation') che viene impiegata.[43] Per dimostrare il teorema occorre trovare un modello ad ogni insieme di clausole S chiuso sotto le regole del calcolo $\mathcal{S}$ e non contenente la clausola vuota. Innanzitutto introduciamo un ordinamento per le clausole *ground*, che ci sarà utile anche nel prossimo paragrafo. Ad una clausola ground C

$$t_1 = s_1, \ldots, t_n = s_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

associamo il multiinsieme di multiinsiemi $m(C)$

$$\{\{t_1, t_1, s_1, s_1\}, \ldots, \{t_n, t_n, s_n, s_n\}, \{u_1, v_1\}, \ldots, \{u_m, v_m\}\}$$

(si noti che le equazioni a sinistra di $\Rightarrow$ vengono pesate 'il doppio' rispetto a quelle sulla destra di $\Rightarrow$). Poi si definisce $C > D$ se e solo se vale $m(C) > m(D)$ (qui si fa riferimento alla doppia estensione ai multiinsiemi dell'ordine di riduzione sui termini). Essendo questo ordinamento sulle clausole ground totale e terminante, lo si utilizza per introdurre un sistema di riscrittura ground R_S, facendo un'induzione transfinita sulle clausole ground che sono istanze di clausole in S. R_S risulta essere convergente e pertanto la R_S-congiungibilità definisce una relazione di equivalenza $\downarrow_{R_S}$ sull'universo di Herbrand $H_{\mathcal{L}}$ dei termini ground di $\mathcal{L}$. Il modello di S cercato sarà allora ottenuto introducendo sull'insieme quoziente $H_{\mathcal{L}}/\downarrow_{R_S}$ l'ovvia interpretazione di Herbrand $\mathcal{I}_H$ ($\mathcal{I}_H$ sarà fatta in modo da interpretare ogni termine ground sulla sua stessa classe di equivalenza).

6.3 Esempi

Vediamo ora alcuni esempi di utilizzo del calcolo $\mathcal{S}$.[44]

[43] Siccome la Sovrapposizione e la Paramodulazione non hanno la proprietà di sollevamento rispetto alle dimostrazioni ground, la tradizionale tecnica di sollevamento ('lifting') che funziona per il calcolo $\mathcal{R}$ deve essere modificata.

[44] Per ottenere ulteriori esempi, lo studente può utilmente formalizzare e risolvere (con il calcolatore, s'intende!) semplici problemi tratti da testi ed eserciziari di algebra elementare (ad esempio, si provi che

ESEMPIO. Formalizziamo il seguente semplice ragionamento:

Carlo e Davide sono entrambi figli di Antonio, il quale ha avuto figli solo da Barbara. Ne consegue che Carlo e Davide sono fratelli (cioè, non sono solo fratellastri).

Utilizziamo due simboli di funzione unari p, m (per 'padre di' e 'madre di'), un simbolo di relazione binario F (per 'essere fratelli') e quattro costanti a, b, c, d (per 'Antonio', 'Barbara', 'Carlo' e 'Davide'). I dati del nostro problema sono le formule

$$p(c) = a, \quad p(d) = a, \quad \forall x\,(p(x) = a \rightarrow m(x) = b).$$

A ciò va aggiunto un 'postulato di significato' relativo al termine 'fratello' che è definito nel vocabolario dell'uso corrente tramite la formula

$$\forall x\,\forall y\,(F(x, y) \leftrightarrow (p(x) = p(y) \wedge m(x) = m(y))).$$

La congettura da dimostrare è ovviamente la formula $F(c, d)$. Mediante skolemizzazione del problema otteniamo le seguenti clausole:

1. $\Rightarrow p(d) = a$
2. $\Rightarrow p(c) = a$
3. $F(c, d) \Rightarrow$
4. $p(x) = a \Rightarrow m(x) = b$
5. $p(x) = p(y), m(x) = m(y) \Rightarrow F(x, y)$
6. $F(x, y) \Rightarrow m(x) = m(y)$
7. $F(x, y) \Rightarrow p(x) = p(y)$

(come vedremo, le ultime due clausole non interverranno nella prova refutativa). Usiamo un LPO basato sulla precedenza $p > m > F > d > c > b > a$. Mediante Sovrapposizione Sinistra generiamo la clausola

$$8.\ p(y) = a,\ m(y) = m(c) \Rightarrow F(c, y) \qquad [SpL\ 2.1; 5.1]$$

Il sistema ha generato il problema di unificazione $p(c) \stackrel{?}{=} p(x)$, che ha upg $x \mapsto c$. Nella clausola $\Rightarrow p(c) = a$ c'è un unico letterale (che è quindi massimale) e si ha $p(c) > a$ (in

un anello booleano ha caratteristica due ed è commutativo, che un gruppo di periodo due è abeliano, che una funzione fra gruppi che preserva il prodotto preserva anche l'unità e l'inverso, ecc.). Alternativamente, si possono utilizzare librerie di problemi reperibili direttamente in rete (come la libreria TPTP). Nella distribuzione di SPASS sono già inclusi vari esempi preconfezionati ed altri sono accessibili dal sito stesso di SPASS.

particolare, $p(c) \not\leq a$). Nella clausola $p(c) = p(y), m(c) = m(y) \Rightarrow F(c, y)$, il letterale negativo $p(c) = p(y)$ è massimale e $p(c) \not\leq p(y)$. Quindi si può procedere: nella $p(c) = p(y), m(c) = m(y) \Rightarrow F(c, y)$, il termine $p(c)$ viene riscritto con a e il risultato è la clausola 8.

Riapplicando lo stesso ragionamento con la clausola 1, otteniamo

$$9.\ a = a,\ m(d) = m(c) \Rightarrow F(c, d) \qquad [SpL\ 1.1; 8.1]$$

Un'applicazione della Regola di Risoluzione Ordinata, ci porta a dedurre la clausola

$$10. \Rightarrow m(c) = b \qquad [Res\ 2.1; 4.1]$$

Qui il sistema ha generato il problema di unificazione $p(c) \stackrel{?}{=} p(x)\ \&\ a \stackrel{?}{=} a$ con upg $x \mapsto c$. Abbiamo già visto che la 2 ha un unico letterale che pertanto è massimale; per quanto riguarda l'altra clausola, in $p(c) = a \Rightarrow m(c) = b$, il letterale negativo $p(c) = a$ è massimale, quindi è possibile procedere, ottenendo la 10.[45] Un ragionamento simile, condotto utilizzando la clausola 1 invece della 2, porta all'inferenza:

$$11. \Rightarrow m(d) = b \qquad [Res\ 1.1; 4.1]$$

Mediante Sovrapposizione Sinistra, otteniamo

$$12.\ a = a,\ m(c) = b \Rightarrow F(c, d) \qquad [SpL\ 11.1; 9.2]$$

Infatti, il problema di unificazione $m(d) \stackrel{?}{=} m(d)$ ha la sostituzione identica per upg; inoltre il vincolo di applicazione della regola è soddisfatto (ad esempio, si ha che $m(d) = m(c)$ è massimale nella 9 e che $m(d) > m(c)$).

Per ottenere la clausola vuota, bastano ora due applicazioni della Risoluzione Ordinata e un'applicazione della Risoluzione per l'Uguaglianza:

$$\begin{aligned} &13.\ a = a \Rightarrow\ F(c, d) && [Res\ 10.1; 12.2] \\ &14.\ a = a\ \Rightarrow && [Res\ 13.2; 3.1] \\ &15.\ \Rightarrow && [EqR\ 14.1] \end{aligned}$$

ESEMPIO. Diamo un esempio che richiede l'uso della regola di Fattorizzazione per l'Uguaglianza. Consideriamo le clausole

[45] Si poteva applicare anche la Sovrapposizione Sinistra, ottenendo come risultato la clausola $a = a \Rightarrow m(c) = b$ invece della 10.

1. $\Rightarrow\ P(x, f(x))$
2. $P(f(x), x) \Rightarrow$
3. $\Rightarrow u = v, w = u, v = w$

Si ottiene la derivazione della clausola vuota mediante i seguenti passaggi (scegliamo un ordinamento basato sulla precedenza $f >_p P$):

4.	$\Rightarrow\ w = f(x), v = w, P(x, v)$	$[SpR\ 3.1; 1.1]$
5.	$\Rightarrow\ w = f(f(x)), x = w$	$[Res\ 4.3; 2.1]$
6.	$\Rightarrow\ x = w, P(f(x), w)$	$[SpR\ 5.1; 1.1]$
7.	$P(w, f(x)) \Rightarrow\ x = w$	$[SpL\ 5.1; 2.1]$
8.	$\Rightarrow\ x = f(y), y = f(x)$	$[Res\ 6.2; 7.1]$
9.	$x = x \Rightarrow f(x) = x$	$[EqF\ 8.1; 8.2]$
10.	$x = x \Rightarrow P(x, x)$	$[SpR\ 9.2; 1.1]$
11.	$P(x, x), x = x \Rightarrow$	$[SpL\ 9.2; 2.1]$
12.	$x = x \Rightarrow$	$[Res\ 10.2; 11.1]$
13.	$\Rightarrow$	$[EqR\ 12.1]$

ESEMPIO. Provare che due fratelli hanno necessariamente gli stessi cugini. Usiamo un linguaggio contenente due simboli di funzione unari $p(x), m(x)$ (che stanno, rispettivamente, per 'padre di x' e 'madre di x') e due predicati binari $F(x, y), C(x, y)$ (che stanno, rispettivamente, per 'x e y sono fratelli' e per 'x e y sono cugini'). Abbiamo per assiomi la chiusura universale delle seguenti formule, che definiscono esplicitamente F e C in termini di p e m:

$$\begin{aligned} F(x, y) \quad &\leftrightarrow \quad (p(x) = p(y) \wedge m(x) = m(y)); \\ C(x, y) \quad &\leftrightarrow \quad ((p(p(x)) = p(p(y))) \vee (p(p(x)) = p(m(y))) \vee \\ &\qquad \vee (p(m(x)) = p(p(y))) \vee (p(m(x)) = p(m(y)))) \end{aligned}$$

(qui x e y sono considerati cugini quando hanno in comune un nonno maschio). La congettura sarà

$$\forall x \forall y (F(x, y) \rightarrow \forall z (C(x, z) \rightarrow C(y, z))).$$

Inserendo i dati in un dimostratore automatico, si ottiene il messaggio 'proof found'.

Invece, due fratellastri non necessariamente hanno gli stessi cugini; infatti, si ottiene il messaggio 'completion found' cambiando l'assioma di definizione di F nel modo seguente:

$$\forall x \forall y (F(x,y) \leftrightarrow (p(x) = p(y) \vee m(x) = m(y))).$$

Questi ultimi due esempi possono essere lievemente variati adottando una definizione di 'cugino' più naturale, in termini di fratellanza dei genitori:

$$\begin{aligned} C(x,y) \quad \leftrightarrow \quad & (F(p(x),p(y)) \vee F(p(x),m(y)) \vee \\ & \vee F(m(x),p(y)) \vee F(m(x),m(y))). \end{aligned}$$

In certi esempi (come alcuni di quelli che abbiamo presentato), per ottenere risultati soddisfacenti occorre combinare il meccanismo dei vincoli del calcolo $\mathcal{S}$ con un'ulteriore meccanismo di restrizione sulle inferenze da eseguire: si tratta del meccanismo di **selezione** cui accenniamo brevemente. Supponiamo di avere a disposizione un criterio[46] che marca in ogni clausola **al più un letterale negativo**. In tal caso, le regole del calcolo vengono modificate nel modo seguente. Le vecchie regole di $\mathcal{S}$ sono attive solo per clausole in cui non ci siano letterali negativi marcati. Nel caso di clausole con un letterale negativo marcato, si possono usare solo le due regole seguenti (dove il letterale negativo marcato è contrassegnato con $(-)^+$):

Regola di Sovrapposizione Sinistra (con selezione)

$$\frac{\Gamma' \Rightarrow \Delta', l = r \qquad \Gamma, s = t^+ \Rightarrow \Delta}{\Gamma\mu, \Gamma'\mu, (s[r]_p)\mu = t\mu \Rightarrow \Delta\mu, \Delta'\mu \;\; \| \;\; VS^+}$$

dove $s_{|p}$ non è una variabile,[47] μ è upg del problema di unificazione $s_{|p} \stackrel{?}{=} l$ e dove VS^+ è il seguente vincolo:

- $l\mu \not\leq r\mu$ e $s\mu \not\leq t\mu$;

- $l\mu = r\mu$ è strettamente massimale in $\Gamma'\mu \Rightarrow \Delta'\mu, l\mu = r\mu$.

[46] La scelta di tale criterio è governata in SPASS tramite l'opzione -Select=i (dove i=0,1,2).

[47] Si osservi che nella premessa sinistra $\Gamma' \Rightarrow \Delta', l = r$ della regola, nessun letterale negativo deve essere marcato. Quindi, se il criterio di marcatura adottato prevede che si marchi sempre un letterale negativo qualora ce ne sia almeno uno, la premessa sinistra della regola dovrà automaticamente essere priva di letterali negativi (questa strategia di saturazione è detta 'strategia positiva' o di 'iperrisoluzione').

Regola di Risoluzione per l'Uguaglianza (con selezione)

$$\frac{s = t^+, \Gamma \Rightarrow \Delta}{\Gamma\mu \Rightarrow \Delta\mu}$$

dove μ è upg di $s \stackrel{?}{=} t$.

Il calcolo $\mathcal{S}^+$ (ossia il calcolo $\mathcal{S}$ modificato con il meccanismo di selezione di letterali negativi sopra illustrato) **resta refutazionalmente completo**, qualunque sia il criterio di marcatura di letterali negativi adottato per le clausole. Come esempio di utilizzo di $\mathcal{S}^+$ si consideri l'insieme consistente della sola clausola:

$$R(x, y), R(y, z) \Rightarrow R(x, z).$$

Siccome tutti e tre i letterali di questa clausola soddisfano il vincolo di massimalità, in $\mathcal{S}$ questa clausola può essere risolta con se stessa, il risultato può essere risolto ancora, ecc., per cui il processo di saturazione diverge. Se invece si marca uno dei due letterali negativi e si usa $\mathcal{S}^+$, non succede proprio nulla (siamo già in presenza di un insieme saturo).[48]

6.4 Ridondanze

Come già illustrato nel paragrafo 4.3, l'implementazione di un calcolo richiede, per essere efficace, l'implementazione anche di regole di riduzione ('backward' e 'forward'). In questo paragrafo affrontiamo in modo più accurato questo problema, introducendo opportune nozioni di ridondanza, sia per clausole che per inferenze.

Se S è un insieme di clausole, $g(S)$ indica l'insieme delle clausole che sono istanze ground di clausole in S. Per dare la nozione di istanza ground di una regola di inferenza, occorre tener conto solo delle sostituzioni ground che soddisfano i relativi vincoli. Se

$$\frac{C_1, \ldots, C_n}{D\mu \;\; \| \;\; V}$$

è un esempio di una regola di inferenza (π),[49] un'istanza ground $(\pi\sigma)$ di (π) è una regola

[48] Si provi infatti ad eseguire questo esempio con SPASS, usando prima l'opzione -Select=0 e poi l'opzione -Select=2 (oppure l'opzione di default -Select=1).

[49] Per il calcolo $\mathcal{S}$ cui facciamo riferimento, si avrà ovviamente $n \leq 2$.

del tipo

$$\frac{C_1\sigma, \ldots, C_n\sigma}{D\sigma}$$

dove σ è una sostituzione ground del tipo $\sigma = \mu\sigma'$ che rispetta il vincolo V. Ad esempio, se V richiedeva che un certo letterale $L\mu$ fosse massimale in una certa premessa $C_i\mu$, allora $L\sigma$ deve essere ancora massimale in $C_i\sigma$. Così, se V richiedeva che valesse $s\mu \not\leq t\mu$ (per certi termini s, t), ora deve valere ancora $s\sigma \not\leq t\sigma$.[50]

Diciamo che la regola (π) è **ridondante** rispetto ad un insieme di clausole S qualora per ogni sua istanza ground $(\pi\sigma)$ si abbia che

$$\{E \,|\, E \in g(S) \;\&\; E < C_m\sigma\} \models D\sigma$$

dove $C_m\sigma$ è massimale (nell'ordinamento fra clausole ground) fra le clausole $C_1\sigma, \ldots, C_n\sigma$ che sono le premesse (istanziate) di $(\pi\sigma)$.

Una clausola C è **ridondante** rispetto ad un insieme di clausole S qualora per ogni istanza ground $C\sigma$ di C valga che

$$\{E \,|\, E \in g(S) \;\&\; E < C\sigma\} \models C\sigma.$$

Possiamo ora introdurre il concetto fondamentale di derivazione (prendiamo come riferimento il calcolo $\mathcal{S}$, quindi parlando di 'regola di inferenza' intendiamo sempre uno dei quattro tipi di regole di inferenza di $\mathcal{S}$ introdotti nel paragrafo precedente). Una **derivazione** (δ) è una successione (finita o infinita) di insiemi di clausole

$$S_1, S_2, S_3, \ldots, S_i, \ldots$$

tale che per ogni i si verifica uno dei due fatti seguenti:

(i) S_{i+1} è ottenuto da S_i aggiungendovi una nuova clausola C che è conseguenza logica di S_i;[51]

(ii) S_{i+1} è ottenuto da S_i rimuovendo una clausola C che è ridondante rispetto ad S_i.

[50] Si noti che, siccome l'ordine di riduzione su cui operiamo è totale sui termini ground, $s\sigma \not\leq t\sigma$ equivale a $s\sigma > t\sigma$.

[51] Questo succede ovviamente sempre se C è conclusione di una regola di inferenza le cui premesse stanno in S_i.

Se (δ) è una derivazione, l'insieme S_∞ delle **clausole generate** da (δ) e l'insieme S_ω delle **clausole persistenti** in (δ) sono così definiti:

$$S_\infty = \bigcup_i S_i, \qquad S_\omega = \bigcup_i \bigcap_{j>i} S_j$$

(si noti che S_ω non è nient'altro che l'insieme delle clausole che da un certo punto in poi non vengono più rimosse).

Una derivazione (δ) è **equa** ('fair') se e solo se per ogni $C_1, \ldots, C_n \in S_\omega$, ogni regola di inferenza avente le $C_1, \ldots, C_n$ come premesse è ridondante in S_∞ (ossia, detto in altre parole, se e solo se ogni inferenza fra clausole *persistenti* è ridondante rispetto all'insieme delle clausole generate nella derivazione).

Possiamo allora rafforzare il teorema di completezza refutazionale del paragrafo precedente, in modo da darne un'interpretazione procedurale che tenga conto di possibili passi di di riduzione:

Teorema 6.2. *Se $S_1, S_2, S_3, \ldots, S_i, \ldots$ è una derivazione equa, allora S_1 è inconsistente se e solo se la clausola vuota appartiene a S_∞.*

L'utilizzo del teorema precedente è facilitato dalla seguente peculiarità del calcolo $\mathcal{S}$: in ogni istanza ground di una regola del calcolo $\mathcal{S}$, *la conclusione è sempre strettamente minore di una delle premesse*, quindi se la conclusione di una regola di inferenza appartiene a S_∞, l'inferenza stessa è ridondante in S_∞. Perciò una derivazione è certamente equa se ogni inferenza fra clausole persistenti viene prima o poi effettuata (nel paragrafo 4.3 si è già visto che con una opportuna funzione di scelta della clausola data, questa proprietà è automaticamente garantita). Il discorso ci riporta a rianalizzare le regole di **riduzione** in avanti ('forward') e all'indietro ('backward') le cui modalità di utilizzo all'interno del ciclo della clausola data sono già state illustrate nel paragrafo 4.3. Per la riduzione in avanti si fa riferimento alla nozione di inferenza ridondante e per la riduzione all'indietro si fa riferimento alla nozione di clausola ridondante.

- **Tautologie**: le clausole del tipo $\Gamma, s = t \Rightarrow s = t, \Delta$ e del tipo $\Gamma \Rightarrow s = s, \Delta$ sono ridondanti rispetto all'insieme vuoto di clausole e possono quindi essere rimosse ad ogni stadio di una derivazione.

- **Sussunzione**: ricordiamo che la clausola D sussume la clausola C, qualora per un matcher μ si abbia $D\mu \subseteq C$; parliamo di sussunzione *stretta* qualora si abbia $D\mu \subset C$.

La sussunzione stretta può essere utilizzata in modo illimitato senza problemi: se in uno stadio S_i sono presenti le clausole D e C con la D che sussume strettamente la C, la C è ridondante in S_i e pertanto può essere eliminata passando a S_{i+1}. Per la sussunzione non stretta, le cose sono più delicate e preferiamo non addentrarci in ulteriori dettagli. Tuttavia la sussunzione anche non stretta può essere utilizzata senza problemi per la riduzione in avanti: se $C_1, \ldots, C_k/C \parallel V$ è una regola di inferenza e S_i contiene una clausola D che sussume la conclusione C, allora la regola stessa è ridondante in S_i e pertanto può essere ignorata (questo perchè, se μ è il matcher che testimonia la sussunzione, avremo sempre $C_m\sigma > C\sigma \geq D\mu\sigma$ per ogni σ ground).[52]

- **Demodulazione** (o riscrittura): supponiamo che la clausola $\Rightarrow l = r$ appartenga ad un certo stadio S_i di una derivazione. Supponiamo anche che S_i contenga una clausola C che può essere riscritta ad una certa clausola D usando la clausola $\Rightarrow l = r$ come regola di riscrittura. Questo fatto significa che in una certa posizione p, la C contiene un termine s (questo è il termine che viene demodulato nel passo di riscrittura) per cui valgono le condizioni seguenti:

  a) per un certo matcher μ, si ha $s \equiv l\mu$;

  b) vale che $l\mu > r\mu$;

  c) la D è ottenuta dalla C sostituendo in posizione p il termine s con $r\mu$.

  Se per ogni σ ground, la $C\sigma$ è maggiore della clausola $\Rightarrow l\mu\sigma = r\mu\sigma$,[53] possiamo continuare la derivazione del modo seguente:

  $$S_{i+1} = S_i \cup \{D\}, \qquad S_{i+2} = S_{i+1} \backslash \{C\}$$

  (questo perchè la C è diventata ridondante in S_{i+1}). La demodulazione può essere utilizzata in modo illimitato invece per la riduzione in avanti (almeno nel caso delle clausole Horn): se $C_1, \ldots, C_k/C \parallel V$ è una regola di inferenza e S_i contiene la clausola

[52] Si ricordi la proprietà sopra menzionata del calcolo $\mathcal{S}$ per cui ogni istanza della conclusione è minore della corrispondente istanza della premessa massimale.

[53] Questa condizione è automaticamente soddisfatta in molti casi, ad esempio qualora il termine demodulato occorra a sinistra di $\Rightarrow$ in C oppure qualora la demodulazione non avvenga in radice. La condizione può essere ulteriormente raffinata ma non proprio del tutto cancellata.

$\Rightarrow l = r$ che giustifica il passo di riscrittura $C \rightarrow D$, allora per come sono fatte le regole di $\mathcal{S}$, si ha che ponendo $S_{i+1} = S_i \cup \{D\}$, l'inferenza diventa ridondante e può essere ignorata.