Sfidare l'Indecidibile

Tecniche e Metodologie del Ragionamento Automatico nella Logica Elementare con Identità

Silvio Ghilardi

Al mio piccolo Tiberio.

Indice

Capitolo 1

Introduzione

La logica a livello elementare è solo semidecidibile e non decidibile, questo si sa. La semidecidibilità significa in concreto che non si può determinare alcun criterio che consenta di staccare la spina della corrente al momento appropriato ad un calcolatore che visibilmente arranchi nello svolgimento di laboriosi calcoli, nel tentativo eroico di verificare la congettura di un teorema matematico.

Nonostante queste limitazioni teoriche, '... eppur si muove' si sarebbe tentati di dire. Senza inopportuni e ingiustificati trionfalismi, possiamo sostenere con qualche buona ragione che i dimostratori automatici odierni funzionano e possono essere utili ed efficaci: dopo le acquisizioni degli anni '90, essi sono in grado non solo di risolvere facilmente elementari esercizi di algebra astratta, ma anche di fornire informazioni interessanti in campi applicativi quali la verifica di programmi o di protocolli. Ed è proprio la possibilità concreta di tali applicazioni (più che forse l'illusione di ottenere risultati matematici avanzati) che guida lo sviluppo e stimola la ricerca del settore - un settore giovane, di appena poche decine di anni, ma sempre vitale e in costante progresso.

In questa introduzione, cercheremo di chiarire alcuni punti chiave relativi all'immagine della logica che lo sviluppo della ricerca nel campo della dimostrazione automatica ci propone. Non ci nascondiamo, infatti, che il logico professionista, leggendo le parti centrali del presente lavoro, potrebbe essere preso da una qualche forma di sconcerto, se non addirittura di marcato disagio.

Le ragioni di tale sconcerto e di tale disagio sono presto dette e possono risiedere ad esempio nei seguenti motivi:

- *nei calcoli che proponiamo non compaiono nè connettivi, nè quantificatori - a guardar bene non compaiono neppure formule, visto che i dati logici vengono trattati di fatto come multiinsiemi variamente costruiti a partire dai termini di un linguaggio;*

- *i calcoli che proponiamo compiono sì inferenze, ma poi (come si evince dal comportamento dei calcolatori su cui sono implementati) fanno uno sforzo ancor maggiore e spesso prevalente per semplificare e distruggere informazioni già acquisite e ritenute ridondanti;*

- *le regole di inferenza stesse sono governate e dirette da principi operativi basati su vincoli imposti dall'esterno che in apparenza poco hanno a che vedere con il contenuto dichiarativo del problema logico che viene trattato.*

Come si è arrivati a tutto questo? Non tenteremo qui di ricostruire lo sviluppo storico della dimostrazione automatica a partire dagli anni '60 in poi (si vedano le Note Bibliografiche al termine di ogni Capitolo per informazioni in tal senso), piuttosto cercheremo di mettere in luce alcuni nodi concettuali, cominciando da certe problematiche emergenti già all'interno di calcoli logici più tradizionali.

Ci possono essere più motivazioni (tutte altrettanto valide nel loro ambito) per costruire e studiare calcoli logici, ma noi ci atterremo ad un criterio di valutazione legato all'efficacia nella ricerca della prova di una data congettura (eventualmente a partire da un numero finito di assiomi di una teoria specifica). Siccome parliamo di 'ricerca di una prova', astraiamo dalla qualità della struttura della prova stessa e puntiamo invece a valutare la dimensione dei tentativi necessari per raggiungerla, compresi quelli infruttuosi e devianti, purchè permessi dalle regole del calcolo stesso. Solo in questo modo, infatti, la formalizzazione logica diventa strumento di acquisizione di conoscenze e non riduce il suo ruolo a mera certificazione di correttezza di risultati ottenuti per altra via.

In questa ottica, i calcoli logicisti storici, basati sull'economia e sulla semplicità della loro formulazione (economia e semplicità pur utili a certi livelli metateorici), mostrano tutti i loro limiti. Essi consentono infatti solo procedimenti enumerativi nella ricerca delle prove e la loro completezza semantica si rivela, da questo punto di vista, un'arma spuntata e controproducente: se tutte le formule logicamente valide sono teoremi e se per stabilire la teoremicità di una formula A si ha a

disposizione solo un metodo enumerativo, fatalmente la ricerca di una prova per A comprenderà la derivazione di una quantità innumerevole di enunciati B che non hanno con A alcuna parentela di contenuto effettivo.

Una valida alternativa ai calcoli logicisti è costituita dai calcoli basati su principi analitico/refutativi (pensiamo ad esempio a tableaux semantici o anche parzialmente a calcoli dei sequenti): in tali calcoli, la tesi da dimostrare viene via via decomposta secondo precise istruzioni legate al significato attribuito ai singoli connettivi e quantificatori, fino a ridurla ad asserzioni tautologiche particolarmente evidenti del tipo $A \supset A$. In questi casi, il calcolo resta semanticamente completo, ma di fatto consente di produrre solo enunciati strettamente connessi alla forma della formula che viene testata.

La ricerca sui calcoli di tipo analitico/refutativo è ancor oggi attiva ed ha anzi ricevuto rinnovato vigore in tempi recenti. A tali calcoli si ispirano tutti coloro che, per mille più che valide ragioni, non se la sentono di sottoscrivere la brutale eliminazione di tutti gli operatori logici operata nei passi di preprocessamento tramite skolemizzazione dai calcoli basati su metodi di saturazione. D'altra parte però, il metodo analitico/refutativo nella sua versione ingenua 'ground'[1] *è soggetto a critiche profonde che cercheremo di enucleare qui di seguito. È nostra opinione che da tali critiche non si possa uscire se non importando all'interno dei calcoli di tipo analitico/refutativo metodi e soluzioni emersi nel settore rivale della dimostrazione automatica basata su metodi di saturazione, come del resto la letteratura recente più accorta ha ampiamente ammesso.*[2]

Il metodo analitico/refutativo contrasta in maniera stridente con la pratica delle dimostrazioni matematiche correnti sotto alcuni aspetti importanti. Da un lato, non si può ragionevolmente sperare che tutta l'informazione necessaria per testare la validità logica di un'implicazione del tipo $A_1 \wedge \cdots \wedge A_n \supset B$ (dove gli A_i sono assiomi di una teoria significativa) sia contenuta in modo facilmente accessibile all'interno dell'implicazione stessa: come ogni matematico esperto sa, prima di

[1] Sono termini ground, dimostrazioni ground, formule ground, ecc. quelli che non contengono variabili (il termine inglese 'ground' è divenuto di uso così corrente che riteniamo ormai inopportuno tradurlo). Parliamo qui di versione 'ground' dei tableaux o del calcolo dei sequenti per intenderne la versione corrente, in cui termini e variabili introdotti dall'applicazione di una regola che analizza un quantificatore non sono soggetti ad istanziazione mediante unificazione e si comportano quindi di fatto come dei termini ground.

[2] Cfr. ad esempio [38], [27].

affrontare la dimostrazione di un teorema importante, occorre preparare il terreno raccogliendo le tesi rilevanti della teoria in oggetto, onde avere a disposizione tutti gli strumenti concettuali necessari. Del resto, non è difficile produrre esempi di esercizi relativamente semplici che richiedono l'acquisizione di lemmi preliminari che possono essere anche molto nascosti.

Da un altro punto di vista, sembra difficile sostenere che l'analisi del significato degli operatori logici possa costituire la sostanza delle dimostrazioni matematiche correnti. Tale pretesa contrasta ad esempio con le numerose manipolazioni puramente algebriche che sono spesso necessarie nelle dimostrazioni e poco si concilia con la (generalmente scarsa) sensibilità logica di matematici anche molto navigati. La sostanza dei problemi, in conclusione, sembra risiedere altrove e sembra non giustificare proprio l'eccessiva enfasi posta dalla teoria della dimostrazione classica sull'analisi delle costanti logiche.

Infine, la stessa eliminazione (a seguito di appropriata analisi) degli operatori logici risulta nella pratica impossibile: le formule universalmente quantificate 'vere' devono ad esempio essere duplicate, riutilizzate e quindi non possono essere mai cancellate. Ciò sembra riflettere da un punto di vista formale quanto abbiamo sostenuto sopra, circa la delicatezza e la problematicità della gestione degli assiomi di una teoria specifica (non per nulla, tali assiomi sono generalmente asserzioni universali).

Se poi ('last but not least') valutiamo l'efficienza dei procedimenti analitico/refutativi, ci troviamo di fronte (non appena si superino gli esempi altamente addomesticati che usualmente si propinano nei corsi di primo livello) a sconcertanti esplosioni combinatorie, dovute principalmente di nuovo ai processi di istanziazione, per non parlare infine del comportamento non facilmente governabile dell'identità. È nostra opinione che tale inefficienza non sia un fatto contingente, ma l'esito finale di nodi concettuali affrontati in modo inadeguato e quindi sostanzialmente irrisolti.

La conclusione di questa discussione è che il processo di ricerca di una prova è un processo altamente complesso (altamente 'intelligente' si vorrebbe dire), in cui si combinano non solo il significato degli operatori logici, ma anche le conoscenze pregresse, certe importanti euristiche operative, nonchè il discernimento accorto che valuti l'importanza delle acquisizioni parziali via via raggiunte e le semplifichi se necessario, ecc. Tutti questi aspetti devono essere classificati come problemi logici, *se vogliamo che la logica diventi uno strumento realmente concreto.*

Se rianalizziamo in quest'ottica la breve storia della dimostrazione automatica dei tempi odierni, ci accorgiamo che essa si rivela un generoso tentativo di dare una risposta razionale *e* puntuale *a ciascuno di questi problemi, sfuggendo sempre alla tentazione di un facile disfattismo metodologico e avvalendosi spregiudicatamente, per contro, di tutte le risorse tecnologiche che il progresso in ambito ingegneristico mette via via a disposizione. In questa luce, anche certe caratteristiche certamente poco ortodossse di calcoli raffinati basati su metodi di saturazione ricevono una loro naturale giustificazione.*

Ma quali sono le caratteristiche principali di tali calcoli basati su metodi di saturazione? I calcoli basati su metodi di saturazione *manipolano insiemi finiti di clausole allo scopo di scoprirne l'inconsistenza (tali clausole comprendono la versione skolemizzata degli assiomi della teoria in ingresso, nonchè della negazione della congettura da dimostrare). Essi operano mediante procedimenti ottimizzati tesi a scoprire, fra le conseguenze logiche di tali clausole, solo quelle che vengono ritenute rilevanti ai fini di una possibile prova refutativa. Le regole di inferenza utilizzate sono principalmente la Regola di Risoluzione di Robinson [79] e la Regola di Paramodulazione di Robinson e Wos [80]: la prima è sostanzialmente la regola del taglio nota dai sistemi alla Gentzen, mentre la seconda non è che una riformulazione del principio lebniziano dell'indiscernibilità degli identici. Entrambe le Regole, però - qui sta un primo punto caratterizzante - conglobano al loro interno una soluzione efficace del problema dell'istanziazione: mediante il meccanismo dell'unificazione,*[3] *esse operano sempre l'istanziazione minima che consenta l'applicazione della Regola stessa.*

Circa la riduzione dello spazio di ricerca delle prove, tale riduzione è particolarmente sensibile soprattutto nel caso della Paramodulazione, ad esempio per il venir meno della proprietà di sollevamento[4] *rispetto alle dimostrazioni a livello 'ground'.*

---

[3] Tale meccanismo dell'unificazione, nonchè certe varianti della Paramodulazione, sono centrali anche nelle versioni 'free-variable' dei tableaux semantici, cfr. sempre [38], [27].

[4] Questa caratteristica sorprendente ha posto per un certo tempo problemi tecnici rilevanti rispetto alla dimostrazione del teorema di completezza refutazionale per la Paramodulazione, che non poteva più essere ottenuta tramite la scorciatoia costituita dal teorema di Herbrand e dalla semplice completezza refutazionale ground. In un primo tempo, per restaurare la proprietà di sollevamento, si erano addirittura introdotti assiomi aggiuntivi, detti di 'funzionalità riflessiva', con effetti computazionali che rendevano il calcolo improponibile.

Nonostante queste caratteristiche, la semplice Regola di Paramodulazione stentò per molto tempo ad ottenere risultati efficaci: anche dopo che si era ottenuta l'eliminazione della paramodulazione sulle variabili (operazione, questa, che sostanzialmente vanificava i vantaggi dell'introduzione del meccanismo dell'unificazione), si dovette attendere fino agli anni '90 per capire appieno come utilizzarla. Quello che nel frattempo era andato maturando negli anni e che avrebbe permesso la svolta, fu il contributo del punto di vista ispirato all'Algebra Computazionale.

L'algebra, per sua intrinseca caratteristica, corre meno della logica il rischio di separare la componente dichiarativo-descrittiva dei suoi problemi dagli aspetti operativi necessari a trattarli. Per questo motivo, è generalmente stata in grado di produrre nella sua storia algoritmi molto interessanti. Un contributo che ha dato il via a tutta una serie di importanti sviluppi negli ultimi decenni è stata la scoperta (da parte di B. Buchberger negli anni '60) del metodo delle basi di Gröbner per risolvere il problema dell'appartenenza di un polinomio ad un dato ideale dell'anello dei polinomi in più variabili su un campo. Il metodo ha una stretta riconosciuta affinità (si veda [22]) con l'algoritmo di Knuth-Bendix [54] introdotto poco dopo per il trattamento del problema della parola uniforme in Algebra Universale, algoritmo che ha dato origine alla teoria dei sistemi di riscrittura.[5] *In tale teoria, troviamo prime importanti risposte rispetto ad alcuni dei problemi cui accennavamo sopra, relativi alla distanza concettuale fra l'euristica concreta impiegata nelle dimostrazioni matematiche e la mera formalizzazione logica di tali dimostrazioni.*

In primo luogo, l'identità è un predicato asimmetrico: quando si dice che 's è uguale a t', si vuole in realtà dire che 't può essere sostituito a s e che conviene fare tale sostituzione' perchè (secondo una qualche misura, magari non banale,[6] *di complessità) 't è più semplice di s'. In secondo luogo, c'è una chiara spiegazione ai 'lemmi nascosti' che risultano necessari a risolvere esercizi anche non troppo difficili: tali lemmi nascosti corrispondono ad equazioni che, una volta orientate, danno istruzioni di semplificazione divergenti fra loro (le cosiddette*

[5] Nell'esposizione del presente testo, abbiamo in realtà operato un rovesciamento della prospettiva storica, presentando i fatti rilevanti relativi ai sistemi di riscrittura come corollari di risultati concernenti il 'Superposition Calculus', che invece, viceversa, è stato chiaramente suggerito da questi ultimi.

[6] Ai metodi di comparazione dei termini dedicheremo l'intero Capitolo 3.

'coppie critiche').[7] *In terzo luogo, una volta realizzato che la scelta di una base assiomatica di una teoria (fra le mille possibili) costituisce un momento cruciale nelle inferenze, occorre concentrare gli sforzi sulla determinazione di una base assiomatica soddisfacente ed efficace. Questo implica da un lato la necessità di elevare al rango di assiomi i lemmi nascosti e dall'altro la necessità di semplificare gli assiomi già ottenuti: anche l'economicità infatti ha un suo peso, serve ad evitare di trascinare all'interno del processo deduttivo ingombranti acquisizioni divenute obsolete col progredire della conoscenza. Così la deduzione logica diventa un processo dinamico aperto, che continuamente sa rimettere in discussione se stessa, alla luce dei fatti nuovi che via via fa emergere. In quarto luogo, infine, tale processo deduttivo (detto di 'completamento') non trascura di avvalersi della formulazione di una specifica tesi che gli si richieda di dimostrare: qualora infatti il completamento diverga (fatto in generale inevitabile per l'indecidibilità della logica equazionale), la presenza della tesi da dimostrare garantisce comunque la completezza, perchè gli assiomi via via dedotti saranno comunque sufficienti nel loro compleso a semplificare la tesi ad un'identità banale del tipo $s = s$, qualora la tesi stessa sia conseguenza logica degli assiomi della teoria iniziale.*

La storia dei primi anni della teoria dei sistemi di riscrittura è stata molto sofferta, perchè si è dovuto faticare parecchio per ottenere una formulazione definitiva accettabile della teoria: vari problemi cruciali sono emersi, ad esempio solo negli anni '80 si sono messi a punto metodi in grado di trattare equazioni (come la legge di commutatività) per loro natura non orientabili.[8] *Uno di tali metodi, la riscrittura ordinata, ha permesso di formulare una nozione più generale di completamento, che è stata poi importata nella dimostrazione automatica basata sulla Paramodulazione. Il risultato di questo lungo processo è stato il* 'Superposition Calculus' *dei primi anni '90 (cui dedicheremo l'intero Capitolo 4), ossia il calcolo che viene implementato sui dimostratori automatici ad alte prestazioni dei giorni nostri. Tale calcolo si differenzia dai calcoli basati sulla semplice Paramodulazione, perchè utilizza il meccanismo dei vincoli di ordinamento: all'interno di una clausola, solo termini massimali di letterali massimali sono utilizzabili per le inferenze e le uniche inferenze significative previste non fanno altro che calcolare coppie critiche fra tali termini. La riduzione che vie-*

[7] Si veda il Capitolo 5.

[8] Si veda la Nota Bibligrafica del Capitolo 5 per maggiori dettagli.

ne operata in questo modo dello spazio di ricerca delle prove refutative risulta molto significativa e i miglioramenti nelle prestazioni si sono rivelati sorprendenti. In aggiunta, il 'Superposition Calculus' dispone di una potente teoria delle ridondanze, in grado di estendere il processo di auto-aggiornamento delle conoscenze acquisite dal caso delle clausole unitarie (considerato dalle procedure di completamento nei sistemi di riscrittura) al caso generale di clausole qualunque (e quindi all'intera logica del primo ordine).

Applicazioni significative stanno emergendo, ne menzioniamo un paio. Utilizzando la AC-Paramodulazione nella versione 'basica',[9] *nel 1997 W.McCune [62], grazie al dimostratore automatico EQP, ha risolto un problema di matematica aperto fin dagli anni di Tarski: il fatto ha avuto un'eco notevole (perfino sul New York Times), quantomeno per il suo significato simbolico. Il problema era il seguente. Un'algebra di Robbins è un insieme dotato di un'operazione binaria + e di un'operazione unaria n soddisfacenti i seguenti assiomi:*

$$x + y = y + x$$
$$(x + y) + z = x + (y + z)$$
$$n(n(x) + y) + n(n(x) + n(y)) = x$$

Le algebre di Boole sono algebre di Robbins (il complemento svolge il ruolo dell'operazione n). Vale anche il viceversa: la congettura era aperta dagli anni '30. Per ottenere tale risultato, sono stati necessari

- *8 giorni di lavoro del calcolatore;*
- *30 megabytes di memoria;*
- *49548 equazioni dedotte;*
- *17663 equazioni non scartate;*
- *... per una dimostrazione finale di soli 13 passaggi !*

EQP ha impiegato solo lo 0,1 per cento del suo tempo a derivare equazioni, il resto del tempo lo ha utilizzato in varie operazioni di semplificazione.

[9] Si veda la Nota Bibliografica del Capitolo 4 per qualche informazione su questi raffinamenti del calcolo.

Come seconda applicazione, menzioniamo l'analisi (relativamente semplice, ma significativa) del protocollo di scambio di chiavi di Newman-Stubblebine condotta tramite il dimostratore automatico SPASS [88].[10] *Tale protocollo mira a far sì che due partecipanti, Alice e Bob, che dispongono solo di chiavi segrete individuali di lungo termine per comunicare con l'amministratore del sistema Trust, possano acquisire un'apposita chiave segreta a breve termine per un'operazione di comunicazione di dati sensibili. Il protocollo prevede cinque fasi:*

1. *Alice manda a Bob il messaggio* $\langle A, Na\rangle$*, che consiste del suo nome e di un numero casuale* Na *(che identificherà in modo unico il suo messaggio di richiesta d'ora in poi);*

2. *Bob manda a Trust il messaggio* $\langle B, Nb, E(Kbt, A, Na, Tb)\rangle$ *consistente del suo nome, di un altro numero casuale* Nb *e di una parte, crittata con la chiave di lungo termine* Kbt*, consistente a sua volta del nome di Alice, del numero casuale prodotto da Alice e del tempo richiesto per la validità della chiave di breve termine;*

3. *Trust costruisce la chiave di breve termine* Kab *e manda ad Alice il messaggio crittato*

$$\langle E(Kat, B, Na, Kab, Tb), E(Kbt, A, Kab, Tb), Nb\rangle;$$

4. *Alice può decrittare, tramite la chiave di lungo termine* Kat*, solo la prima parte del messaggio di Trust, ma da tale parte viene a conoscere la nuova chiave di breve termine* Kab*; così la usa per mandare a Bob il messaggio* $\langle E(Kbt, A, Kab, Tb), E(Kab, Nb)\rangle$ *(la prima parte di tale messaggio è trasmessa senza conoscerne il contenuto);*

5. *Bob riceve il messaggio di Alice, estrae dalla prima parte la chiave* Kab *e la usa per riconoscere Alice tramite la decodifica della seconda parte del messaggio.*

Sembra così che i due partecipanti abbiano raggiunto il loro scopo (con un limitato uso delle chiavi di lungo termine, il cui uso prolungato

[10] L'analisi automatica dei protocolli di rete è un settore di grande interesse attuale. Segnaliamo che essa può essere efficacemente condotta con metodi logici anche molto diversi dalla logica elementare classica (si possono utilizzare in tal senso ad esempio logiche di ordine superiore o logiche temporali). Per sviluppi più recenti, sempre all'interno degli strumenti da noi trattati, si veda [20].

è rischioso). In realtà il protocollo può essere violato da un intruso che intercettasse il messaggio 4 e mandasse al suo posto il messaggio $\langle E(Kbt, A, Na, Tb), E(Na, Nb)\rangle$ *a Bob:*[11] *a quel punto Bob verrebbe ingannato e identificherebbe in* Na *la chiave di breve termine, col risultato che potrebbe trasmettere direttamente all'intruso dati sensibili.*

La formalizzazione del protocollo tramite la logica del primo ordine è alquanto complessa: servono 8 formule per descrivere il protocollo stesso e 12 formule per descrivere cosa può fare l'intruso (cioè ascoltare messaggi altrui, fingere di essere qualcun altro quando spedisce messaggi, ecc.). Una formula, infine, farà da congettura ed esprimerà la situazione indesiderata di inganno. La dimostrabilità di tale formula testimonierà l'insicurezza del protocollo. Infine, un analogo esperimento (dopo l'ovvia correzione tendente ad impedire la confusione fra chiavi e numeri casuali) certificherà invece che il problema non si può ripetere nel protocollo così riparato.

L'emergere di questi successi fa ben sperare e traccia vie interessanti per il futuro. Al contempo, però, nuove sfide ci attendono, perchè l'euristica concreta delle dimostrazioni è talmente complessa e ricca, che non possiamo certamente pretendere di averla compresa ed esaurita appieno. Citiamo solo un problema a tale riguardo, su cui si stanno al momento concentrando alcuni sforzi della comunità scientifica. Per molte teorie importanti e utili ad esempio nella verifica del software (pensiamo al caso dell'aritmetica di Presburger o dell'aritmetica reale, per fare solo due nomi) sono note procedure di decisione basate su metodi specifici, che nulla hanno a che vedere con i metodi generali che abbiamo delineato sopra. Come è possibile integrare tale procedure di decisione fra loro e integrarle a loro volta all'interno di dimostratori automatici 'generalisti'?[12] *Inoltre, la logica del primo ordine ha un ridotto potere espressivo, che da un lato permette di ottenere risultati promettenti, ma dall'altro limita la portata delle applicazioni; il problema che si pone di conseguenza è ancora un problema di integrazione (magari in modalità non completamente automatica, ma solo interattiva), questa volta con sistemi dalle caratteristiche molto complementari (quali ISABELLE, COQ, ecc.) che sanno affrontare la logica di ordine superiore.*

[11] L'intruso dispone di $E(Kbt, A, Na, Tb)$ perchè ha anche registrato il messaggio 2 di Bob a Trust.

[12] È questo il settore della 'combinazione' all'interno del ragionamento automatico, settore su cui daremo qualche piccola informazione nella Note Bibliografiche dei Capitoli 4 e 5.

In conclusione, il settore del ragionamento automatico si pone al giorno d'oggi come un settore dinamico, in cui convergono sia aspetti matematici interessanti, sia robuste componenti ingegneristiche. Al contempo, però, esso non è privo di solide indicazioni teoretiche, tese a far sì che la logica si riappropri di uno dei suoi compiti storici - la teoria dell'inferenza - e possa così dispiegare appieno le sue potenzialità di strumento scientifico conoscitivo.

Milano, Maggio 2004

Avvertenza. *Questo testo è stato scritto con un duplice intento. Da un lato abbiamo separato, in appositi paragrafi, le dimostrazioni più impegnative in modo da consentire l'utilizzo del materiale trattato in un corso universitario regolare (seppur non, presumibilmente, in un corso di primo livello nelle Lauree Triennali). Dall'altro lato, abbiamo cercato di far sì che lo studio integrale del testo mettesse in condizione il lettore di accedere, almeno parzialmente, alla letteratura specializzata. In tal senso, ci siamo impegnati in una ampia opera di rielaborazione e di sintesi di una letteratura alquanto estesa e frammentata, molto efficiente ed attenta a mantenere in tempo reale l'aggiornamento della comunità scientifica, un po' meno impegnata invece a fornire al neofita tutte le informazioni dettagliate necessarie ad impadronirsi della materia.*

Comunque si affronti la lettura del presente testo, è nostra opinione (e anche il lettore più tradizionalista e riluttante non faticherà a rendersene conto) che non sia possibile acquisirne i contenuti fondamentali senza una adeguata attività di laboratorio: a tal fine abbiamo inserito nella Nota Bibliografica del Capitolo 4 l'indicazione di utili risorse di rete. Gli stessi Esempi che proporremo nel testo spesso non saranno risolvibili integralmente con carta e penna (...provare per credere!) e saranno piuttosto da considerarsi indicazioni per altrettanti esperimenti col calcolatore.

Capitolo 2

Preliminari

In questo capitolo verranno richiamate le conoscenze essenziali relative ai linguaggi del primo ordine e alla loro semantica. Verrà anche spiegato come preprocessare un problema logico in modo da ridurlo ad un problema di soddisfacibilità di un insieme di clausole. Infine, si studieranno le tematiche relative all'unificazione.

2.1 Linguaggi elementari

Un linguaggio elementare consente di nominare individui, costruire designazioni di individui a partire da altre, parlare di proprietà di individui, quantificare su di essi, ecc. Diamo la relativa definizione formale:

Definizione 2.1.1 *Un* linguaggio elementare *(o segnatura)* $\mathcal{L}$ *è una coppia* $\langle \{\mathcal{P}_n\}_{n\geq 0}, \{\mathcal{F}_n\}_{n\geq 0}\rangle$*, dove*

- *per ogni* $n \geq 0$*,* $\mathcal{P}_n$ *è un insieme, detto insieme dei simboli di predicato* n*-ario;*
- *per ogni* $n \geq 0$*,* $\mathcal{F}_n$ *è un insieme, detto insieme dei simboli di funzione* n*-aria.*

Gli elementi di $\mathcal{P}_0$ sono detti lettere proposizionali e rappresentano proposizioni inanalizzate nei loro costituenti; gli elementi di $\mathcal{P}_1$ rappresentano proprietà di oggetti ('essere bianco', 'essere simpatico', ecc.),

mentre gli elementi di $\mathcal{P}_2$ rappresentano relazioni ('essere amico di', 'essere vicino di banco di', ecc.). In generale, gli elementi di $\mathcal{P}_n$ rappresentano relazioni n-arie fra individui. Se $\mathcal{P}_2$ contiene il simbolo '=', diciamo che $\mathcal{L}$ è un *linguaggio con identità*.

Gli elementi di $\mathcal{F}_0$ sono detti costanti individuali: si rappresentano come costanti individuali i nomi propri (di persona o di cose, come 'Pietro', 'Lucia', il 'Lunedì', il 'Capodanno', ecc.). Gli elementi di $\mathcal{F}_n$ ($n > 0$) indicano operazioni (ad esempio, operazioni fra numeri come somma e prodotto, o fra liste come 'append' o 'reverse'); il numero n indica l'arietà dell'operazione stessa (ad esempio, la somma e 'append' sono binarie, 'reverse' è unario, così come sono unarie l'operazione di considerare l'opposto di un numero o il padre di un essere umano, ecc.)

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 reali un opportuno linguaggio $\mathcal{L}_1$ potrebbe contenere: a) due costanti (elementi di $\mathcal{F}_0$) $0, 1$; b) due simboli di operazioni binarie (elementi di $\mathcal{F}_2$), cioè la somma '$+$' e il prodotto '$\cdot$'; c) due relazioni binarie (elementi di $\mathcal{P}_2$), 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 simboli di funzione unaria (elementi di $\mathcal{F}_1$), cioè $p =$'il padre di' e $m =$'la madre di'; b) un predicato binario, l'identità.

In aggiunta a $\langle \{\mathcal{P}_n\}_n, \{F_n\}_n \rangle$, 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 $\mathcal{V} = \{x, x_0, x_1, x_2, \ldots, y, y_0, y_1, y_2, \ldots, z, z_0, z_1, z_2, \ldots\}$ detto insieme delle variabili individuali;
- i connettivi proposizionali booleani $\wedge$ ('e'), $\vee$ ('o' inclusivo), $\neg$ ('non');
- i quantificatori $\forall$ ('per ogni') ed $\exists$ ('esiste').

Definizione 2.1.2 *L'insieme $T_{\mathcal{L}}$ degli $\mathcal{L}$-termini (o più semplicemente* termini*) è cosí definito:*

- *ogni $x \in \mathcal{V}$ è un termine;*
- *se $f \in \mathcal{F}_n$ (per $n \geq 0$) e $t_1, \ldots, t_n$ sono termini, $f(t_1, \ldots, t_n)$ è un termine.*

L'insieme $F_{\mathcal{L}}$ delle $\mathcal{L}$-formule (o più semplicemente formule*) è cosí definito:*

- *se $P \in \mathcal{P}_n$ (per $n \geq 0$) e $t_1, \ldots, t_n$ sono termini, $(P(t_1, \ldots, t_n))$ è una formula (atomica);*
- *se A_1, A_2 sono formule, tali sono anche $(A_1 \wedge A_2)$, $(A_1 \vee A_2)$, $(\neg A_1)$;*
- *se A è una formula e $x \in \mathcal{V}$, allora $(\forall x A)$ e $(\exists x A)$ sono formule.*

I termini servono a costruire designazioni di individui complesse, partendo dalle designazioni di base specificate dai simboli di funzione del linguaggio. 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.

Convenzioni notazionali:

- Usiamo le abbreviazioni $(A_1 \supset A_2)$[1] e $(A_1 \Leftrightarrow A_2)$ per $((\neg A_1) \vee A_2)$ e per $((A_1 \supset A_2) \wedge (A_2 \supset A_1))$, rispettivamente.
- Per eliminare alcune parentesi, stipuliamo che i quantificatori e la negazione leghino più strettamente di $\wedge, \vee$. Le parentesi esterne saranno di regola omesse. Infine, le parentesi verranno omesse nel caso di disgiunzioni o congiunzioni iterate (dove si stabilisce convenzionalmente che esse vadano re-inserite associandole per esempio sulla destra).
- Un termine è *chiuso* (o *'ground'*) se è costruito senza usare variabili.
- L'insieme dei sottotermini di un termine t ha la ovvia definizione: $Sott(t) = \{t\}$, se $t \in \mathcal{V} \cup \mathcal{F}_0$, $Sott(f(t_1, \ldots, t_n)) =$

[1] Ci riserviamo invece il simbolo '$\rightarrow$' per denotare in seguito relazioni di riscrittura.

$\{f(t_1, \ldots, t_n)\} \cup Sott(t_1) \cup \cdots \cup Sott(t_n)$. Definizioni come questa fanno riferimento ad una induzione, sul numero di simboli del termine stesso. Analogamente si faranno simili induzioni sulle formule.

- La nozione di sottoformula di una data formula è definita in modo analogo al caso precedente (per esempio avremo che $\forall x A$ e $\exists x A$ hanno per sottoformule se stesse e le sottoformule di A). C'è però anche una utile nozione di sottoformula in senso esteso (frequente nei testi orientati alla teoria della dimostrazione) su cui non ci soffermiamo.

- Una occorrenza di una variabile x in una formula A è detta *vincolata* qualora si trovi all'interno di una sottoformula del tipo $\forall x B$ o $\exists x B$, 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, mentre la seconda è libera; la y ha una sola occorrenza, che è libera.

- Una variabile x occorre libera in A sse[2] qualche sua occorrenza in A è libera. Una variabile x occorre libera in un termine t sse essa semplicemente occorre in t.

- Un *enunciato* o formula chiusa è una formula in cui nessuna variabile occorre libera. Una formula *ground* è una formula che non contiene variabili.

- Useremo $\underline{x}, \underline{y}, \ldots$ per indicare n-ple di variabili (di lunghezza non specificata) e $\underline{t}, \underline{u}, \ldots$ per n-ple di termini. Torna utile convenire che le n-ple di variabili $\underline{x}, \underline{y}, \ldots$ (dette arietà) siano sempre implicitamente supposte essere senza ripetizioni (la stessa convenzione non la facciamo però per le notazioni $\underline{t}, \underline{u}, \ldots$ che indicheranno quindi n-ple di termini con possibili ripetizioni).

- La notazione $\forall \underline{x} A$, denota (sia $\underline{x} = x_1, \ldots, x_n$) la formula $\forall x_1 \cdots \forall x_n A$.

- Le notazioni $t(\underline{x})$, $A(\underline{x})$ indicano che il termine t e la formula A contengono al più le variabili $\underline{x}$ libere. Con $Var(t)$, $Var(A)$ indichiamo l'insieme delle variabili che occorrono libere in t, A.

[2] Qui e nel seguito, 'sse' abbrevia 'se e solo se'.

- Con notazioni del tipo $A(t_1/x_1, \ldots, t_n/x_n)$ (o, più semplicemente $A(\underline{t}/\underline{x})$ o $A(\underline{t})$) indicheremo il risultato dell'operazione di sostituzione simultanea dei termini t_i al posto di tutte le occorrenze libere delle variabili x_i in A.[3] Nel caso in cui solo una variabile non venga sostituita con se stessa, useremo abbreviazioni come $A(t/y)$ per $A(x_1/x_1, \ldots, x_n/x_n, t/y)$.

2.2 Strutture

Diamo ora la semantica per i linguaggi elementari: la semantica precisa il significato d'uso degli elementi di un linguaggio in una situazione specifica. Gli elementi di un linguaggio, infatti, sono puri segni e non hanno nessun significato esplicitamente collegato ad essi. Ad esempio, in matematica il simbolo '$\cdot$' può essere usato per indicare il prodotto fra numeri, la composizione di funzioni, un'operazione di moltiplicazione astratta, ecc. Anche nel linguaggio naturale certe parole (ad es. 'pesca') hanno più significati e il loro significato d'uso deve essere precisato (spesso in modo implicito) da una nozione di interpretazione che fa riferimento ad un contesto.

Per assegnare la semantica a un linguaggio del primo ordine si procede come segue: innanzitutto si fissa un insieme non vuoto $\mathbf{A}$, detto dominio della interpretazione, sul quale assumeranno i valori le variabili individuali, poi si fissa una funzione che associ ad ogni simbolo di funzione n-aria 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 $\mathbf{A}^n$). Formalmente abbiamo che:

Definizione 2.2.1 *Una $\mathcal{L}$-struttura $\mathcal{A}$ è una coppia $\langle \mathbf{A}, \mathcal{I} \rangle$ dove $\mathbf{A}$ è un insieme (non vuoto),*[4] *detto* dominio*, e $\mathcal{I}$ è una (famiglia di) funzioni, detta* interpretazione*, che opera come specificato qui di seguito. $\mathcal{I}$ associa*

- *ad ogni $P \in \mathcal{P}_n$ un sottoinsieme $\mathcal{I}(P)$ di $\mathbf{A}^n$.*

- *ad ogni $f \in \mathcal{F}_n$ una funzione $\mathcal{I}(f) : \mathbf{A}^n \longrightarrow \mathbf{A}$.*

[3] Per prevenire i noti effetti indesiderati, segnalati su tutti i manuali di logica, si faccia precedere la sostituzione da una eventuale rinomina delle variabili vincolate di A.

[4] Segnaliamo che la restrizione ai domini non vuoti, tradizionale nei testi di logica, è innaturale e può essere rimossa mediante l'esplicitazione della arietà delle formule coinvolte nelle dimostrazioni.

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.[5]

Se $\mathcal{L}$ contiene l'identità, stipuliamo che $\mathcal{I}(=)$ *sia sempre l'insieme delle coppie identiche, cioè* $\{(a, a) \mid a \in \mathbf{A}\}$.

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 $\mathcal{L}$. 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 'non-

[5] Conveniamo che, per $n = 0$, $\mathbf{A}^n$ sia l'insieme $\mathbf{1} = \{*\}$ dotato di un solo elemento. In questo modo, le funzioni $\mathbf{1} \longrightarrow \mathbf{A}$ sono gli elementi di $\mathbf{A}$ e i sottoinsiemi di $\mathbf{1}$ (che sono $\{*\}$ e $\emptyset$) svolgono il ruolo, rispettivamente, dei valori di verità 'vero' e 'falso'.

no 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 chiuso* t. Se t è una costante (vecchia o nuova) $\mathcal{I}(t)$ è già stato definito. Se t è $f(t_1, \ldots, t_n)$, allora $\mathcal{I}(t)$ 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 2.2.2 *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 sulla complessità di* A *come segue:*

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

Se A *è una formula qualunque (non necessariamente chiusa)* $\mathcal{A} \models A$ *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*).*[6]

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. Il seguente Lemma si dimostra in modo standard:[7]

[6] L'enunciato $\forall x_1 \cdots \forall x_n A$ è detto *chiusura universale* della formula A.

[7] Naturalmente, per dimostrarlo occorre fare un'induzione preventiva per stabilire un'analoga proprietà dei termini.

Lemma 2.2.3 *Siano $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ una $\mathcal{L}$-struttura e sia $A(\underline{x})$ una $\mathcal{L}$-formula contenente al più le variabili $\underline{x} = x_1, \ldots, x_n$ libere. Date due n-ple di termini chiusi (di $\mathcal{L}_{\mathbf{A}}$) $\underline{t} = t_1, \ldots, t_n$ e $\underline{u} = u_1, \ldots, u_n$, tali che valga $\mathcal{I}(t_i) = \mathcal{I}(u_i)$ (per ogni $i = 1, \ldots, n$), risulta:*

$$\mathcal{A} \models A(\underline{t}/\underline{x}) \quad \text{sse} \quad \mathcal{A} \models A(\underline{u}/\underline{x}).$$

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 2.2.4 *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 $\mathcal{A} \models B$ valga per ogni formula B appartenente a $\mathcal{T}$. Si dice che $\mathcal{T}$ è* soddisfacibile*, qualora abbia almeno un modello (altrimenti, si dice che $\mathcal{T}$ è* insoddisfacibile *o* inconsistente*). $\mathcal{T} \models A$ (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 $\mathcal{L}$-struttura $\mathcal{A}$.*

Lo scopo del presente libro 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?[8] 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 $\mathcal{T}'$, quindi il nostro problema sarà quello di *stabilire se un dato un insieme finito $\mathcal{T}'$ di enunciati è o meno insoddisfacibile*. Il primo passo sarà di preprocessare $\mathcal{T}'$ stesso in modo da portarlo in una forma più trattabile. Ancor prima però, riteniamo utile introdurre alcune notazioni sintattiche che sono divenute di uso corrente nella letteratura orientata alla dimostrazione automatica.

2.3 Alberi e occorrenze

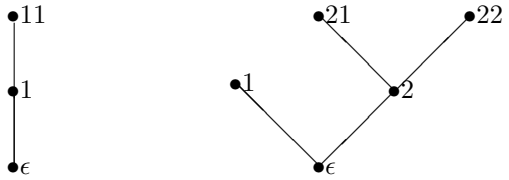
Facciamo alcune precisazioni sugli alberi. Con $\mathbf{N}$ indichiamo l'insieme dei numeri naturali, con $\mathbf{N}_+$ l'insieme dei numeri naturali strettamente

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

positivi e con $\mathbf{N}_+^*$ l'insieme delle liste finite su $\mathbf{N}_+$ (inclusa la lista vuota). Tali liste vengono indicate con le lettere $p, q, r, \ldots$. La relazione $p \leq q$ vale fra tali liste sse p è un prefisso di q (ossia sse esiste q' tale che $pq' = q$). La notazione $|p|$ indica la lunghezza della lista p.

Un *albero* T è un sottoinsieme non vuoto di $\mathbf{N}_+^*$ con le seguenti proprietà: i) se $p \in T$ e $p = qr$, allora $q \in T$; ii) dati $i, j \in \mathbf{N}_+$ tali che $j < i$, se $pi \in T$ 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 prima 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)

Dato un albero T, un *successore* di un nodo $p \in T$ è un nodo $q \in T$ che sia del tipo pi per un certo $i \in \mathbf{N}_+$. T ha diramazione finita sse ogni nodo di T ha un numero finito di successori. Un *ramo* di T è una lista (finita o infinita) di nodi di T

$$p_1, p_2, p_3, \ldots$$

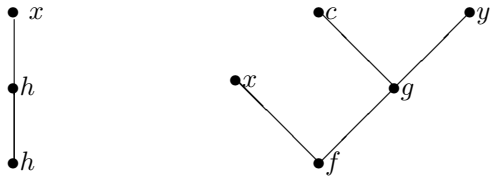
tale che $p_1 = \epsilon$ e tale che p_{k+1} è un successore di p_k per ogni k. Nelle dimostrazioni ci capiterà talvolta di utilizzare il seguente fatto ben noto:

Lemma 2.3.1 *(di König). Un albero infinito a diramazione finita contiene sempre un ramo infinito.*

Dim. Sia T infinito a diramazione finita. Un nodo p di T si dice prolungabile sse l'insieme $\{|q| : pq \in T\}$ non è superiormente limitato. Si noti che se un nodo è prolungabile, tale è uno dei suoi successori immediati (questo perché tali successori immediati sono finiti, essendo l'albero a diramazione finita). La radice di T è prolungabile: proviamo infatti per induzione su K, che se l'insieme $\{|q| : q \in T\}$ è limitato da K, allora T è finito. Se $K = 0$, T ha la sola radice ed è pertanto finito. Se $K > 0$ e la radice ha per successori immediati i nodi $1, 2, \ldots, n$, possiamo applicare l'ipotesi induttiva agli n alberi $T_i = \{q : iq \in T\}$ per $i = 1, \ldots, n$ (sono gli alberi che stanno sopra i successori immediati della radice) e concludere che T stesso è finito.

Possiamo allora definire il ramo infinito $r_1, \ldots, r_i, \ldots$ ponendo r_1 uguale alla radice e scegliendo r_{i+1} fra i nodi prolungabili successori immediati di r_i. ⊣

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 unaria e c è una costante, gli alberi



rappresentano i termini $h(h(x))$ e $f(x, g(c, y))$.[9]

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

[9]Spesso nella letteratura questi alberi sono disegnati capovolti, ossia con la radice in alto e le foglie in basso. Questa scelta può avere dei risvolti sostanziali nel significato di locuzioni (che, ad ogni buon conto, noi eviteremo di usare) quali ad esempio 'per ogni posizione *sotto* (risp. *sopra*) una data posizione p, succede che ...'

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

- $Pos(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(t_i)\}$, 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:[10]

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

- $t_{|iq} \equiv s_{i|q}$, 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 $f(x, h(c))$. Il seguente Lemma (dal significato trasparente) contiene alcune informazioni tecniche che saranno utili e si dimostra facilmente per induzione su $|p|$:

Lemma 2.3.2 *Sia t un termine e siano $p, q \in Pos(t)$.*

(i) *se $q \leq p$ (sia $p = qr$), allora $t_{|p} \equiv (t_{|q})_{|r}$;*

(ii) *se p e q sono posizioni parallele (cioè inconfrontabili rispetto alla relazione $\leq$), allora per ogni s, r, si ha $(t[s]_p)[r]_q \equiv (t[r]_q)[s]_p$;*

(iii) *se p e q sono posizioni parallele, allora per ogni s si ha $(t[s]_p)_{|q} \equiv t_{|q}$.*

Ci capiterà di utilizzare le definizioni che abbiamo dato anche per le formule, oltre che per i termini. La relativa estensione è ovvia e viene lasciata al lettore: in sostanza, gli alberi che rappresentano le

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

formule avranno nodi etichettati anche con i simboli di predicato, con i connettivi $\wedge, \vee, \neg$ e con i quantificatori-seguiti-da-variabili $\forall x, \exists y, \ldots$.

Come ulteriore convenzione, se E è un'espressione (ossia un termine o una formula), allora con $|E|$ indichiamo la *lunghezza* di E (ossia la cardinalità di $Pos(E)$) e con $top(E)$ indichiamo il simbolo in radice di E (ossia, se E è una formula non atomica, $top(E)$ ne è il connettivo o quantificatore principale e, se E è una formula atomica o un termine, $top(E)$ è il simbolo di relazione o di funzione con cui E comincia).[11]

Il seguente Lemma di Rimpiazzamento si dimostra per induzione su $|p|$:

Lemma 2.3.3 *Siano A, B, C formule, sia $p \in Pos(A)$ tale che $A_{|p} \equiv B$ e siano $\underline{x}$ le variabili che occorrono libere in B o C.*

(i) *La formula*

$$\forall \underline{x}(B \Leftrightarrow C) \supset (A \Leftrightarrow A[C]_p)$$

è logicamente valida.

(ii) *Se per ogni $q < p$ si ha che $top(A_{|q})$ non è il connettivo $\neg$, allora la formula*

$$\forall \underline{x}(B \supset C) \supset (A \supset A[C]_p)$$

è logicamente valida.

2.4 Sistemi di riscrittura astratti

I formalismi basati sulla riscrittura saranno centrali per molte questioni che affronteremo; siccome alcuni fatti importanti sono comuni a qualunque forma specifica essi assumano, preferiamo introdurli da subito in un contesto molto generale.

Definizione 2.4.1 *Un sistema di riscrittura astratto (ARS) è una coppia*

$$(A, \rightarrow)$$

dove $\rightarrow \subseteq A \times A$ è una relazione binaria. Scriviamo $a \rightarrow b$ per dire che la coppia (a, b) sta nella relazione $\rightarrow$ (in tal caso, diciamo che b è un successore diretto di a).

[11] Se E è una variabile, $top(E)$ non è definito.

Definizione 2.4.2 *Dato un ARS $(A, \to)$, diamo le seguenti definizioni:*

$$\xrightarrow{0} := \{\langle a, a\rangle : a \in A\}$$

$$\xrightarrow{i+1} := \to^i \circ \to$$

$$\xrightarrow{+} := \bigcup_{i>0} \to^i$$

$$\xrightarrow{*} := \bigcup_{i\geq 0} \to^i$$

$$\xrightarrow{-1} := \{(a, b) : b \to a\}$$

$$\longleftrightarrow := \to \cup \leftarrow$$

$$\xleftrightarrow{*} := (\leftrightarrow)^*.$$

Nella definizione appena data, '$\circ$' indica la composizione di relazioni (quindi $a \to^{i+1} b$ vale se e solo se b è raggiungibile da a mediante $i+1$ passi della relazione $\to$). La relazione $\xrightarrow{+}$ viene ad essere la più piccola relazione transitiva contenente $\to$, mentre la relazione $\xrightarrow{*}$ viene ad essere la più piccola relazione riflessiva e transitiva contenente $\to$. Infine, le relazioni $\longleftrightarrow$ e $\xleftrightarrow{*}$ vengono ad essere, rispettivamente, la più piccola relazione simmetrica e la più piccola relazione di equivalenza contenenti $\to$. La relazione $\to^{-1}$ è anche notata con $\leftarrow$.

Definizione 2.4.3 *Un elemento $a \in A$ è detto riducibile se e solo se esiste b tale che $a \to b$. Se un elemento $a \in A$ non è riducibile, a è detto essere in forma normale.*

Definizione 2.4.4 *Dati $a, b \in A$, essi sono congiungibili se e solo se esiste $c \in A$ tale che*

$$a \xrightarrow{*} c \xleftarrow{*} b$$

e si scrive in tal caso $a \downarrow b$.

Definizione 2.4.5 *Un ARS $(A, \to)$ è detto* di Church-Rosser *se e solo se per ogni $a, b \in A$ vale che*

$$(a \xleftrightarrow{*} b \textit{ implica } a \downarrow b).$$

Definizione 2.4.6 *Un ARS* $(A, \to)$ *è* confluente *se e solo se per ogni* $a, b_1, b_2 \in A$ *vale che*

$$(b_1 \overset{*}{\leftarrow} a \overset{*}{\to} b_2 \text{ implica } b_1 \downarrow b_2).$$

Proposizione 2.4.7 *Un ARS è di Church-Rosser se e solo se è confluente.*

Dim. Un lato è ovvio. Assumiamo la confluenza e dimostriamo la proprietà di Church-Rosser. Siano a, b tali che $a \overset{*}{\leftrightarrow} b$: questo significa che esistono $n \geq 1$ e che esistono $c_1, \ldots, c_n$ tali che $a = c_1$, $b = c_n$ e tali che per ogni $i = 1, \ldots, n-1$ vale $c_i \to c_{i+1}$ oppure $c_i \leftarrow c_{i+1}$. Dimostriamo per induzione su n che a e b sono confluenti. Se $n = 1$, questo fatto è ovvio perchè $a = b$. Se $n > 1$, per ipotesi induttiva esiste d tale che $c_2 \overset{*}{\to} d \overset{*}{\leftarrow} c_n = b$. Abbiamo ora due casi: $c_1 \to c_2$ oppure $c_1 \leftarrow c_2$. Nel primo caso segue subito che a e b confluiscono; nel secondo caso, per la confluenza di R, esiste d' tale che $a = c_1 \overset{*}{\to} d' \overset{*}{\leftarrow} d$, quindi a e b confluiscono entrambi a d'. ⊣

Definizione 2.4.8 *Un ARS* $(A, \to)$ *è* terminante *(o noetheriano) se e solo se non esistono successioni infinite del tipo*

$$a_0 \to a_1 \to a_2 \to \cdots$$

Definizione 2.4.9 *Un ARS* $(A, \to)$ *è* convergente *(o canonico) se e solo se è confluente e terminante.*

Proposizione 2.4.10 *Se un ARS è confluente, allora ogni elemento a ha al più una forma normale. Se è terminante, una tale forma normale esiste sempre (notiamone una con* $a\downarrow$*). Infine, in un ARS convergente, la forma normale è unica e per ogni* a, b *vale che*

$$a \overset{*}{\leftrightarrow} b \text{ sse } a\downarrow \equiv b\downarrow$$

La Proposizione precedente (di immediata verifica) è particolarmente importante: essa ci dice che, se si riesce a ottenere la convergenza, allora la relazione $a \overset{*}{\leftrightarrow} b$ è testabile mediante il confronto diretto delle forme normali di a e b.

La convergenza coinvolge due aspetti, ossia la terminazione e la confluenza; i due aspetti non sono totalmente indipendenti.

Definizione 2.4.11 *Un ARS* $(A, \to)$ *è* localmente confluente *se e solo se per ogni* $a, b_1, b_2 \in A$ *vale che*

$$(b_1 \leftarrow a \to b_2 \textit{ implica } b_1 \downarrow b_2).$$

Lemma 2.4.12 *(di Newman) Un ARS terminante è confluente se e solo se è localmente confluente.*

Dim. Un lato è ovvio; l'altro lato si dimostra utilizzando uno schema di ragionamento detto 'principio di induzione noetheriana'. Sia $(A, \to)$ un ARS terminante e sia P una proprietà degli elementi di A. Allora abbiamo che la seguente inferenza è valida:[12]

$$\frac{\forall a \in A\ (\forall b \in A\ (a \xrightarrow{+} b \supset P(b)) \supset P(a))}{\forall a \in A\ P(a)}$$

Nel caso che ci interessa, la proprietà $P(a)$ è la seguente:

$$\forall b_1 \forall b_2 (b_1 \xleftarrow{*} a \xrightarrow{*} b_2 \Rightarrow b_1 \downarrow b_2).$$

Supponiamo che $P(u)$ sia vera per ogni u tale che $a \xrightarrow{+} u$ e proviamo $P(a)$. Siano quindi b_1, b_2 tali che $b_1 \xleftarrow{*} a \xrightarrow{*} b_2$. Se $b_1 \xleftarrow{*} a$ avviene in zero passi, allora $a = b_1$ e quindi banalmente b_1 e b_2 confluiscono; analoga considerazione si può fare se $b_2 \xleftarrow{*} a$ avviene in zero passi. L'unico caso non banale si verifica quindi quando esistono c_1, c_2 tali che

$$b_1 \xleftarrow{*} c_1 \leftarrow a \to c_2 \xrightarrow{*} b_2.$$

Per la confluenza locale di R, esiste c tale che

$$c_1 \xrightarrow{*} c \xleftarrow{*} c_2.$$

Siccome possiamo applicare l'ipotesi induttiva noetheriana sia a c_1 che a c_2, troviamo dapprima d tale che

$$b_1 \xrightarrow{*} d \xleftarrow{*} c_1$$

[12] Il motivo per cui l'inferenza vale è semplice: si assuma la premessa. Se esiste $a_0 \in A$ tale che $P(a_0)$ è falso, esisterà (per la validità della premessa) $a_1 \in A$ tale che $a_0 \xrightarrow{+} a_1$ e tale che $P(a_1)$ non vale a sua volta. Continuando così, si produce una catena infinita che non può esserci.

e infine d' tale che

$$d \xrightarrow{*} d' \xleftarrow{*} b_2.$$

In conclusione si ha

$$b_1 \xrightarrow{*} d' \xleftarrow{*} b_2,$$

proprio come si voleva. ⊣

Il lemma di Newman suggerisce che, se si è interessati alla convergenza, conviene affrontare per prima la questione della terminazione, perchè una volta ottenuta la terminazione, la confluenza si può sostituire con la locale confluenza, che è una proprietà molto più debole e più trattabile.

Nel prossimo paragrafo, tuttavia, non saremo interessati a formalismi di riscrittura convergenti, ma solo terminanti.

2.5 Forme normali

Se P è un insieme, denotiamo con $Mul(P)$ l'insieme dei *multiinsiemi* (finiti) su P. Questi ultimi possono essere definiti come le funzioni a valori nei numeri naturali $M : P \longrightarrow \mathbf{N}$ tali che $M(x) \neq 0$ vale solo per un numero finito di $x \in P$ ($M(x)$ dice 'quante volte' x compare in M). Usiamo la notazione $x \in M$ per abbreviare $M(x) \neq 0$ e usiamo la notazione $M \cup N$ per indicare l'unione dei multiinsiemi M ed N: con il formalismo delle funzioni a valori nei numeri naturali, avremo $(M \cup N)(x) := M(x) + N(x)$ per ogni $x \in P$.[13]

Un *letterale positivo* è una formula atomica e un *letterale negativo* è la negazione di una formula atomica; una *clausola* è un multiinsieme di letterali.

Spesso risulta utile separare nelle clausole i letterali positivi dai letterali negativi; ciò può essere realizzato mediante la notazione dei sequenti. Useremo le lettere greche $\Gamma, \Delta, \ldots$ per indicare multiinsiemi di formule atomiche. Se ora $\Gamma = \{A_1, \ldots, A_n\}$ e $\Delta = \{B_1, \ldots, B_m\}$, la clausola

$$\{\neg A_1, \ldots, \neg A_n, B_1, \ldots, B_m\}$$

verrà notata con

$$\Gamma \Rightarrow \Delta$$

[13] 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 $\{x, y, y\} \cup \{x, y, z\} = \{x, x, y, y, y, z\}$.

o, più estesamente, con

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

Si noti che non si escludono i casi in cui n o m o entrambi siano uguali a 0 (se $n = m = 0$, la clausola (C) è detta essere la *clausola vuota* '$\Rightarrow$').

Useremo spesso, senza eplicito preavviso, notazioni dal significato trasparente quali

$$\Gamma, \Gamma', C_1, \ldots, C_k \Rightarrow \Delta, \Delta', D_1, \ldots, D_l$$

per intendere

$$(*) \qquad \Gamma \cup \Gamma' \cup \{C_1, \ldots, C_k\} \Rightarrow \Delta \cup \Delta' \cup \{D_1, \ldots, D_l\}.$$

In questi casi, ribadiamo per chiarezza che i letterali negativi della clausola $(*)$ sono le formule del tipo $\neg A$ per $A \in \Gamma \cup \Gamma' \cup \{C_1, \ldots, C_k\}$ e i letterali positivi della clausola $(*)$ sono le formule atomiche $A \in \Delta \cup \Delta' \cup \{D_1, \ldots, D_l\}$.

Spesso confonderemo la clausola (C)

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

con il singolo enunciato $(C^\forall)$ che consiste nella chiusura universale della formula[14]

$$(A_1 \wedge \cdots \wedge A_n) \supset (B_1 \vee \cdots \vee B_m).$$

Una teoria $\mathcal{T}$ è in *forma clausale* quando i suoi assiomi sono del tipo $(C^\forall)$ per opportune clausole (C). Un risultato classico della logica elementare stabilisce che:

Teorema 2.5.1 *Data una teoria finitamente assiomatizzata $\mathcal{T}$ (in un linguaggio $\mathcal{L}$), esiste una teoria finitamente assiomatizzata $\mathcal{T}'$ (in un linguaggio $\mathcal{L}' \supseteq \mathcal{L}$) che è in forma clausale e che è soddisfacibile se e solo se $\mathcal{T}$ lo è. Inoltre, $\mathcal{T}'$ è effettivamente calcolabile da $\mathcal{T}$.*

Il punto centrale dell'enunciato del teorema precedente è il requisito dell'effettiva calcolabilità[15] e in effetti il teorema viene dimostrato

[14] Per convenzione, stabiliamo che la congiunzione vuota stia per un qualunque enunciato assegnato $\top$ che sia logicamente valido e che la disgiunzione vuota stia per $\neg\top$ (che abbreviamo con $\bot$). Così, la clausola vuota indica un enunciato insoddisfacibile.

[15] Senza questo requisito, l'enunciato stesso del teorema perde significato, perchè basterebbe prendere $\mathcal{T}'$ uguale a $\{\top\}$ o a $\{\bot\}$, a seconda che $\mathcal{T}$ sia soddisfacibile o meno.

esibendo un opportuno procedimento di calcolo per $\mathcal{T}'$. Le diverse dimostrazioni esistenti producono diversi procedimenti di calcolo e le differenze che risultano nella forma di $\mathcal{T}'$ risultano molto importanti per ragioni di efficienza. Diciamo subito che un procedimento ottimale probabilmente non esiste e che, al contrario di quanto si pensa solitamente, il passaggio da $\mathcal{T}$ a $\mathcal{T}'$ non è un banale fatto di preprocessamento, ma una *vera e propria euristica* per l'influenza essenziale che ha nella successiva ricerca di una dimostrazione per refutazione (ed è proprio questa successiva e altamente imprevedibile ricerca il fatto centrale che conta, molto più della mera complessità di $\mathcal{T}'$).

Le trasformazioni sugli enunciati che vengono usate per passare da $\mathcal{T}$ a $\mathcal{T}'$ vengono talvolta suddivise in *strutturali e non*. Sono trasformazioni non strutturali quelle che alterano mediante manipolazioni (quali le leggi distributive, le leggi di esportazione/importazione dei quantificatori, ecc.) le caratteristiche delle formule cui si applicano; sono trasformazioni strutturali quelle che invece utilizzano predicati-etichetta per sottoformule ed assiomi di esplicita definibilità (questi ultimi possono essere assiomi di esplicita definibilità totale o parziale, dipendente cioè dalla polarità delle occorrenze). Il procedimento che proponiamo è misto, non ha alcuna pretesa di particolare efficienza: l'abbiamo scelto per ragioni pragmatiche (esso riflette abbastanza bene le procedure implementate sui dimostratori automatici correnti) e inoltre ha qualche suo merito qualora si voglia tenere bassa la complessità delle clausole in uscita.

Utilizzeremo un approccio basato su regole di riscrittura per formule. Una regola di riscrittura per formule è una coppia di formule scritta nella forma

$$C \to D.$$

Se R è un insieme di regole di riscrittura per formule, scriviamo $A \to^p_R B$ per esprimere il fatto che esite una regola $C \to D$ appartenente a R tale che

$$A_{|p} \equiv C \quad \text{e} \quad B \equiv A[D]_p$$

(spesso ometteremo esplicitamente l'indicazione di p e/o di R in $A \to^p_R B$ qualora siano chiare dal contesto). Quindi la relazione $A \to^p_R B$ vale quando R contiene la regola $C \to D$ e B si ottiene da A sostituendo nella posizione p la sottoformula C con la formula D. Chiaramente, la relazione $A \to_R B$ definisce un sistema di riscrittura astratto (che chiamiamo ancora R per semplicità) sull'insieme delle formule. Se R è

terminante, ogni formula A ammette una forma normale (non necessariamente unica) che si può calcolare applicando successivamente ad A le regole di R, finchè possibile.

Siamo pronti per descrivere ora un algoritmo che realizza quanto richiesto dal Teorema 2.5.1: tale algoritmo si compone di 6 passi che vanno eseguiti nell'ordine indicato. L'algoritmo opera su insiemi finiti di *enunciati* e si inizializza agli assiomi della teoria $\mathcal{T}$ in ingresso. I passi che vengono descritti mediante regole di riscrittura prevedono il calcolo di una forma normale per ogni enunciato nella teoria corrente (i sistemi di riscrittura che utilizzeremo saranno tutti terminanti). L'esecuzione completa di ogni passo avviene in tempo polinomiale e le trasformazioni effettuate conservano la soddisfacibilità della teoria cui vengono applicate.[16]

Passo 1 (Rettificazione). Una formula A è detta rettificata qualora, date $p, q \in Pos(A)$, se $p \neq q$ e se $top(A_{|p})$, $top(A_{|q})$ sono due quantificatori (Q_1x_1), (Q_2x_2), allora $x_1 \not\equiv x_2$ (ossia, due quantificatori occorrenti in A non possono legare la stessa variabile). Mediante cambi alfabetici (ossia rinomine di variabili vincolate), ogni formula è logicamente equivalente a una formula rettificata. In questo passo, tutti gli enunciati della teoria corrente $\mathcal{T}$ vengono rettificati (lasciamo al lettore di verificare che i passi successivi mantengono questa proprietà di $\mathcal{T}$).

Passo 2 (Forma Normale Negativa). Si applichino le seguenti regole di riscrittura per formule:

$$\neg\neg C \to C, \qquad \neg(C \vee B) \to \neg C \wedge \neg B, \qquad \neg(C \wedge B) \to \neg C \vee \neg B,$$
$$\neg\exists x C \to \forall x \neg C, \qquad \neg\forall x C \to \exists x \neg C.$$

Queste regole danno un sistema di riscrittura terminante: basta a tale scopo associare per esempio a ciascuna formula A il numero (che decresce ad ogni passo di riscrittura)

$$\sum\{|nf(A_{|p})| : p \in Pos(A) \ \&\ top(A_{|p}) \equiv \neg\}$$

(qui $|nf(A_{|p})|$ è la lunghezza della formula $nf(A_{|p})$ ottenuta da $A_{|p}$ cancellando tutte le negazioni). Per il Lemma 2.3.3 (i), l'applicazione

[16] Non sempre tuttavia tali trasformazioni mantengono l'equivalenza logica (due enunciati A e B si dicono logicamente equivalenti qualora $A \Leftrightarrow B$ sia una verità logica).

delle regole mantiene l'equivalenza logica (e quindi anche l'equisoddisfacibilità). Nelle forme normali che si ottengono in questo modo, le negazioni si trovano tutte davanti a formule atomiche.

Passo 3 (Riduzione d'Ambito). Questo passo serve a minimizzare l'arietà delle funzioni di Skolem che verranno introdotte al passo successivo. Si applichino le seguenti regole di riscrittura per formule:

$$\begin{array}{ll} \forall x(C \vee B) \to \forall xC \vee B, & \forall x(C \wedge B) \to \forall xC \wedge B, \\ \forall x(B \vee C) \to B \vee \forall xC, & \forall x(B \wedge C) \to B \wedge \forall xC, \\ \exists x(C \vee B) \to \exists xC \vee B, & \exists x(C \wedge B) \to \exists xC \wedge B, \\ \exists x(B \vee C) \to B \vee \exists xC, & \exists x(B \wedge C) \to B \wedge \exists xC, \end{array}$$

dove si suppone che x non occorra libera in B.[17] Queste regole danno un sistema di riscrittura terminante: basta a tale scopo associare a ciascuna formula A (come misura di complessità) la somma della lunghezze delle sottoformule $A_{|p}$ tali che $top(A_{|p})$ è un quantificatore. Per il Lemma 2.3.3 (i), l'applicazione delle regole mantiene l'equivalenza logica (e quindi anche l'equisoddisfacibilità).

Passo 4 (Skolemizzazione). Si applichi la seguente regola di riscrittura:[18]

$$\exists xC \to C(f(\underline{y})/x)$$

dove le $\underline{y}$ sono le variabili che occorrono libere in C e f è un *nuovo* simbolo di funzione dell'arietà corrispondente. La terminazione è ovvia (ogni passo elimina un quantificatore esistenziale finchè non ne rimane nemmeno uno). Questo passo mantiene l'equisoddisfacibilità (ma non l'equivalenza logica): per vederne il motivo si osservi quanto segue. Da un lato, la formula

$$\forall \underline{y}(C(f(\underline{y})/x) \supset \exists xC)$$

[17] Si possono aggiungere a questa lista anche le regole

$$\forall x(C \wedge B) \to \forall xC \wedge \forall xB, \qquad \exists x(C \vee B) \to \exists xC \vee \exists xB$$

(seguite da una rinomina rettificante). Tuttavia, l'uso di queste regole aggiuntive è a doppio taglio, in quanto se è vero che la skolemizzazione di $\exists xC \vee \exists xB$ potrebbe far introdurre al passo successivo delle funzioni di Skolem di arietà minore della corrispondente skolemizzazione di $\exists x(C \vee B)$, è altrettanto vero che l'uso di funzioni di Skolem differenti per la stessa x potrebbe far perdere un'informazione preziosa per accorciare la successiva prova refutativa.

[18] Per ragioni di efficienza, è bene applicare tale regola cominciando dalle occorrenze più esterne (ossia essa va applicata in una posizione p di un enunciato A della teoria corrente solo qualora valga $top(A_{|q}) \not\equiv \exists$ per ogni $q < p$).

è una verità logica,[19] per cui tenendo conto che tutti gli enunciati della teoria $\mathcal{T}$ corrente sono in Forma Normale Negativa, il Lemma 2.3.3 (ii) garantisce che la soddisfacibilità di $\mathcal{T}$ dopo il passo di riscrittura implica la soddisfacibilità di $\mathcal{T}$ come era prima del passo stesso. Vice versa, dato un modello $\mathcal{A}$ di $\mathcal{T}$ (prima del passo), se ne espanda la funzione interpretazione $\mathcal{I}$ al nuovo simbolo f come segue: presi $\underline{a} \in \mathbf{A}$, si ponga $\mathcal{I}(f)(\underline{a})$ uguale ad un $b \in \mathbf{A}$ tale che $\mathcal{A} \models C(\underline{a}/\underline{y}, b/x)$, se tale b esiste (altrimenti $\mathcal{I}(f)(\underline{a})$ è arbitraria). Nella struttura così espansa, si ha che la formula

$$\forall \underline{y}(C(f(\underline{y})/x) \Leftrightarrow \exists x C)$$

è vera, per cui tale struttura espansa è modello di $\mathcal{T}$ (dopo il passo) per il Lemma 2.3.3 (i).

Passo 5 (Trasformazione in Formule Universali). Si applichino le regole di riscrittura:[20]

$$\forall x C \vee B \rightarrow \forall x(C \vee B), \qquad \forall x C \wedge B \rightarrow \forall x(C \wedge B),$$

$$B \vee \forall x C \rightarrow \forall x(B \vee C), \qquad B \wedge \forall x C \rightarrow \forall x(B \wedge C).$$

Queste regole mantengono l'equivalenza logica e danno un sistema terminante perchè ad ogni passo di riscrittura diminuisce ad esempio il numero

$$\sum \{|A_{|p}| : p \in Pos(A) \ \& \ top(A_{|p}) \equiv \vee, \wedge\}.$$

Al termine del passo 5, la teoria $\mathcal{T}$ corrente consiste di soltanto di enunciati universali in forma normale negativa (una formula *universale* è una formula del tipo $\forall \underline{x} M$, dove la *matrice* M è una formula priva di quantificatori).

Passo 6 (Trasformazione in Forma Normale Congiuntiva). L'ultimo problema da risolvere è la presenza nella teoria corrente $\mathcal{T}$ di sottoformule del tipo $D \vee (B \wedge C)$ o del tipo $(B \wedge C) \vee D$. Potremmo risolverlo, come abbiamo fatto fin qui, mediante trasformazioni non strutturali: tali trasformazioni sarebbero però in questo caso basate sulle leggi distributive e produrrebbero quindi un'esplosione esponenziale in spazio

[19] Si osservi che la sostituzione $C(f(\underline{y})/x)$ non richiede rinomine per evitare conflitti con eventuali variabili vincolate. Questo perchè quando si applica un passo di riscrittura, la C è sottoformula di un *enunciato* rettificato A appartenente alla teoria corrente $\mathcal{T}$ e quindi le $\underline{y}$ non hanno occorrenze vincolate in C.

[20] Anche qui, quando si applica un passo di riscrittura, la formula della testa della regola occorre come sottoformula di un enunciato rettificato A della teoria corrente, per cui la x non può occorrere libera in B.

degli enunciati presenti in $\mathcal{T}$. Quindi preferiamo usare trasformazioni strutturali che manterranno però solo l'equisoddisfacibilità.[21] Ognuna delle trasformazioni usate in questo passo eliminerà una sottoformula F del tipo incriminato (la terminazione seguirà banalmente dalla diminuzione del numero delle occorrenze del connettivo $\wedge$ in $\mathcal{T}$). Sia F del tipo $D \vee (B \wedge C)$ (l'altro caso è analogo) e siano $\underline{x}$ le variabili che occorrono libere in $B \wedge C$. Usiamo un *nuovo* predicato $P(\underline{x})$ dell'arietà corrispondente e aggiorniamo $\mathcal{T}$ con

$$\mathcal{T}' \;=\; \{A[\![D \vee P(\underline{x})/F]\!] \,|\, A \in \mathcal{T}\} \cup \{\forall \underline{x}(\neg P(\underline{x}) \vee B), \forall \underline{x}(\neg P(\underline{x}) \vee C)\},$$

dove la notazione $A[\![D \vee P(\underline{x})/F]\!]$ significa la sostituzione in A di tutte occorrenze di F con $D \vee P(\underline{x})$.[22] Resta da verificare che $\mathcal{T}$ e $\mathcal{T}'$ sono equi-soddisfacibili. Si noti che ogni modello di $\mathcal{T}'$ è modello dell'enunciato $\forall \underline{x}(P(\underline{x}) \supset (B \wedge C))$, quindi è anche modello di $\mathcal{T}$ per il Lemma 2.3.3 (ii). Viceversa, se $\mathcal{A}$ è modello di $\mathcal{T}$, se ne espanda la funzione interpretazione $\mathcal{I}$ al nuovo simbolo P ponendo $\mathcal{I}(P) = \{\underline{a} \in \mathbf{A}^n \,|\, \mathcal{A} \models B(\underline{a}) \wedge C(\underline{a})\}$. Nella struttura così espansa, si ha che la formula

$$\forall \underline{x}(P(\underline{x}) \Leftrightarrow (B \wedge C))$$

è vera, per cui tale struttura espansa è modello di $\mathcal{T}'$ per il Lemma 2.3.3 (i).

Al termine del passo 6, la teoria corrente $\mathcal{T}$ consiste di enunciati della forma

$$\forall \underline{x}\ (C_1 \wedge \cdots \wedge C_n)$$

dove ogni C_i è una disgiunzione di formule atomiche o atomiche negate, ossia è del tipo

$$\neg C_{i,1} \vee \cdots \vee C_{i,k_i} \vee B_{i,1} \vee \cdots \vee B_{i,l_i}.$$

Le clausole corrispondenti a $\mathcal{T}$ sono allora le n clausole

$$C_{i,1}, \ldots, C_{i,k_i} \Rightarrow B_{i,1}, \ldots, B_{i,l_i}.$$

[21] Alcuni dimostratori automatici sono in grado di valutare da soli caso per caso, mediante opportuni algoritmi efficienti, se convenga applicare le leggi distributive o meno (prendendo come criterio di riferimento la complessità delle formule in uscita).

[22] Le formule $\forall \underline{x}(\neg P(\underline{x}) \vee B), \forall \underline{x}(\neg P(\underline{x}) \vee C)$ danno un lato soltanto della definizione esplicita del predicato-etichetta P come $\forall \underline{x}(P(\underline{x}) \Leftrightarrow (B \wedge C))$. Per certi motivi, analizzati nella letteratura e legati alla simulazione dei tagli analitici nelle prove refutative, l'uso di definizioni esplicite complete risulterebbe migliore.

Quanto detto giustifica il Teorema 2.5.1; aggiungiamo soltanto che, nella pratica, è utile prevedere anche ulteriori passi che realizzino certe semplificazioni banali delle formule via via trattate.

Riassumendo, *per sapere se certi enunciati $A_1, \ldots, A_n$ implichino o meno logicamente un certo enunciato B,*[23] *occorre preprocessare il problema applicando la procedura del Teorema 2.5.1 alla teoria* $\{A_1, \ldots, A_n, \neg B\}$; sull'insieme delle clausole risultanti, agirà poi il calcolo refutativo che vedremo nel Capitolo 4, tendente a ricercare una prova della clausola vuota.

ESEMPIO 1. Facciamo i passi necessari di preprocessamento per sapere se la formula

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

implica o meno logicamente la formula

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

La (2) va negata; successivamente si ottiene l'enunciato in Forma Normale Negativa $\exists x \forall y(R(x, y) \wedge \neg Q(f(y)))$. I successivi passi dell'algoritmo che abbiamo descritto producono una teoria corrispondente alle tre clausole

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

$$\Rightarrow R(c, y) \tag{C_2}$$

$$Q(f(y)) \Rightarrow . \tag{C_3}$$

Vedremo nel Capitolo 4 che l'insieme delle tre clausole $\{C_1, C_2, C_3\}$ è davvero inconsistente (quindi la (1) ha come conseguenza logica la (2)).

[23] Questo è il formato dei problemi accettati dai dimostratori automatici correnti (se, al posto della singola tesi B, si trova una lista finita di formule, quest'ultima viene interpretata come disgiunzione).

2.6 Unificazione

Una *sostituzione* σ è una mappa

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

tale che l'insieme delle variabili x tali che $x \not\equiv \sigma(x)$ è finito. Tale insieme è detto essere il *dominio* $dom(\sigma)$ della σ, mentre il *codominio* $cod(\sigma)$ della σ è l'insieme di termini $\{\sigma(x) : x \in dom(\sigma)\}$.

Se $\{x_1, \ldots, x_n\}$ è un insieme di variabili che include 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 $\sigma\tau$, calcolata su x, dà il valore della $\hat{\tau}$ calcolata sul termine $\sigma(x)$).

Da questo si ricava subito, con una facile induzione, che per ogni espressione E si ha

$$E(\sigma\tau) \equiv (E\sigma)\tau$$

e quindi anche che la composizione di sostituzioni è associativa nel senso che vale 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 vale 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, si vede facilmente che 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 che è auto-inversa.

La seguente Proposizione (che si dimostra facilmente per induzione sulla lunghezza di p) risulterà importante per certi passaggi tecnici nel seguito del presente testo:

Proposizione 2.6.1 *Siano σ una sostituzione, t, s dei termini e $p \in Pos(t)$ una posizione. Risulta allora che*

(i) $(t_{|p})\sigma \equiv (t\sigma)_{|p}$;

(ii) $(t[s]_p)\sigma \equiv (t\sigma)[s\sigma]_p$.

Inoltre, se $p \in Pos(t\sigma)$, ma $p \notin Pos(t)$, allora esistono q, r tali che $p = qr$ e tali che $t_{|q}$ è una variabile.

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; se valgono simultaneamente le relazioni $\sigma \geq \tau$ e $\tau \geq \sigma$, allora σ e τ differiscono solo per una rinomina (ossia si può provare facilmente che esiste in tal caso una rinomina ρ tale che $\sigma\rho = \tau$).

Definizione 2.6.2 *Un problema di unificazione è un insieme di coppie di termini, che viene scritto 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')*[24] *di* (U) *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.

Teorema 2.6.3 *Dato un problema di unificazione*

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

è possibile determinare in modo effettivo se esso ammette o meno soluzioni e, in caso positivo, è possibile calcolarne un upg.

Descriviamo qui di seguito un algoritmo che realizza quanto prescritto dal Teorema 2.6.3. L'algoritmo manipola *insiemi* di equazioni e si inizializza alle equazioni del problema originario (U). Ad ogni passo viene eseguita non deterministicamente una delle istruzioni previste dalla Tabella 2.1 (finchè non ci sono più istruzioni da eseguire o finchè

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

non si raggiunge lo stato di fallimento).[25] Si noti che usiamo il simbolo '+' per denotare l'unione disgiunta di insiemi: questo significa che, se nella Tabella 2.1 compare un'istruzione del tipo $S + \{s \stackrel{?}{=} t\}/S \cup S'$, essa va eseguita rimuovendo l'equazione $s \stackrel{?}{=} t$, mettendo al suo posto le equazioni S' e infine eliminando nell'unione insiemistica le eventuali ripetizioni.

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

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

Si noti che $dom(\mu)$ è *disgiunto da* $Var(cod(\mu))$: questo prova che μ è una sostituzione *idempotente*, ossia che vale $\mu\mu = \mu$.

Giustifichiamo ora l'algoritmo esposto nella Tabella 2.1, mostrando che esso realizza quanto richiesto dal Teorema 2.6.3. Innanzitutto, ogni esecuzione dell'algoritmo termina; per convincersene, basta associare all'insieme corrente S di equazioni la misura di complessità data dalla terna

$$\langle n_1(S), n_2(S), n_3(S) \rangle,$$

dove:

- $n_1(S)$ è il numero delle variabili[26] x che sono in forma risolta in S (x è in forma risolta in S sse in S c'è una sola occorrenza di x che è in un'equazione del tipo $x \stackrel{?}{=} t$);[27]
- $n_2(S)$ è la lunghezza di S (ossia il numero $\sum\{|t| + |s| : t \stackrel{?}{=} s \in S\}$);
- $n_3(S)$ è il numero delle equazioni mal orientate di S (ossia il numero delle equazioni di S del tipo $t \stackrel{?}{=} x$, dove t non è una variabile).

Si vede subito che, ad ogni passo di esecuzione, la terna associata all'insieme delle equazioni correnti diminuisce nell'ordine lessicografico (da sinistra) delle terne di interi positivi.[28]

[25] Poichè vedremo che (comunque lo si esegua) l'algoritmo termina in stato di fallimento o calcola un upg, l'ordine di applicazione delle istruzioni è ininfluente (a meno di una rinomina) sul risultato.

[26] Si noti che l'algoritmo non introduce mai variabili che non siano già presenti nel problema in ingresso.

[27] Quindi, in particolare, x non occorre in t.

[28] Per informazioni sui prodotti lessicografici, si veda il Capitolo 3.

Tabella 2.1: Le regole dell'algoritmo di Unificazione

$$\frac{S + \{t \stackrel{?}{=} t\}}{S} \quad \text{(Semplificazione)}$$

$$\frac{S + \{f(t_1, \ldots, t_n) \stackrel{?}{=} f(u_1, \ldots, u_n)\}}{S \cup \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\}} \quad \text{(Decomposizione)}$$

$$\frac{S + \{t \stackrel{?}{=} x\}}{S \cup \{x \stackrel{?}{=} t\}} \quad \text{(Orientamento)}$$

$$\frac{S + \{x \stackrel{?}{=} t\}}{S(t/x) \cup \{x \stackrel{?}{=} t\}} \quad \text{(Sostituzione)}$$

$$\frac{S + \{f(t_1, \ldots, t_n) \stackrel{?}{=} g(u_1, \ldots, u_m)\}}{\text{FAIL}} \quad \text{(Conflitto)}$$

$$\frac{S + \{x \stackrel{?}{=} t\}}{\text{FAIL}} \quad \text{(Rilevamento di Occorrenza)}$$

- L'istruzione (Orientamento) si applica qualora t non sia una variabile.
- L'istruzione (Sostituzione) si applica se t non contiene occorrenze di x e x occorre in almeno una delle rimanenti equazioni.
- L'istruzione (Conflitto) si applica se $f \not\equiv g$.
- L'istruzione (Rilevamento di Occorrenza) si applica se $t \not\equiv x$ ma x occorre in t.

Ad ogni passo di esecuzione dell'algoritmo, l'insieme delle sostituzioni σ che risolvono l'insieme delle equazioni correnti resta invariato. Ciò è immediato per le istruzioni di Semplificazione, Decomposizione, Orientamento, Conflitto e Rilevamento di Occorrenza.[29] Nel caso dell'istruzione di Sostituzione, dobbiamo verificare che le σ che risolvono $S_1 := S + \{x \stackrel{?}{=} t\}$ sono le stesse di quelle che risolvono $S_2 := S(t/x) \cup \{x \stackrel{?}{=} t\}$. Fissiamo una σ che risolve S_1 o S_2: per tale σ abbiamo per forza $x\sigma \equiv t\sigma$. Consideriamo la sostituzione $\theta : x \mapsto t$ (sicchè $S(t/x)$ è $S\theta$). Ora, la sostituzione composta $\theta\sigma$ manda x in $t\sigma$ e ogni $z \not\equiv x$ in $z\sigma$: perciò abbiamo $\theta\sigma = \sigma$. Allora è chiaro che σ unifica S sse unifica $S\theta$, cioè σ risolve S_1 sse risolve S_2.

Resta solo da provare che, in caso di terminazione senza fallimento, la sostituzione μ data da

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

corrispondente allo stato terminale dell'algoritmo

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

è upg di quest'ultimo. Che μ sia soluzione di (U_f) è ovvio in quanto per ogni $i = 1, \ldots, k$, si ha che y_i non occorre in t_i. Sia θ una soluzione di (U_f): proviamo semplicemente che $\theta = \mu\theta$. Ovviamente θ e $\mu\theta$ agiscono nello stesso modo sulle variabili che non siano $y_1, \ldots, y_k$; d'altra parte, per ogni $i = 1, \ldots, k$, abbiamo

$$y_i\mu\theta \equiv t_i\theta \equiv y_i\theta$$

dove l'ultima uguaglianza proviene dal fatto che θ è soluzione di (U_f).

ESEMPIO 2. Consideriamo il problema di unificazione:

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

Applicando l'istruzione di Decomposizione otteniamo:

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

[29] Nel caso del Rilevamento di Occorrenza ('Occur Check' in inglese), si noti che se t non è una variabile e x occorre in t, per ogni sostituzione σ si avrà sempre $|t\sigma| > |x\sigma|$, per cui i termini $t\sigma$ e $x\sigma$ non saranno mai uguali.

applicando l'istruzione di Orientamento 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, ottenendo

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

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

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

per Sostituzione otteniamo

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

ci manca solo un passo di Orientamento per ottenere il risultato finale

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

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

L'algoritmo che abbiamo esposto per il calcolo degli upg è *esponenziale in spazio*: il problema sta nell'istruzione di Sostituzione, che può far esplodere la lunghezza delle equazioni correnti se esaustivamente applicata.[30] Tuttavia, segnaliamo brevemente che la scelta di opportuni tipi di dati rende l'unificazione trattabile: i termini vanno rappresentati non come alberi, ma come grafi aciclici diretti ('dag'), in cui distinte occorrenze di un identico sottotermine possono corispondere allo stesso nodo del grafo. Identificando in un opportuno dag tutte le occorrenze della stessa variabile nel problema di ingresso e identificando in corso di esecuzione i nodi corrispondenti a sottotermini già unificati, si può ottenere un algoritmo che lavora in spazio lineare e tempo quadratico.

[30] Si consideri ad esempio il problema di unificazione

$$x_1 \stackrel{?}{=} f(x_0, x_0), \; x_2 \stackrel{?}{=} f(x_1, x_1), \; \ldots, \; x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1}).$$

Con ulteriori raffinamenti, si può arrivare anche ad una *complessità di tipo lineare* in tempo.

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 2.6.4 *Un problema di matching è un insieme di coppie di termini, che viene scritto 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 viene applicata solo ai membri sinistri del problema (M) di partenza. Il matching può essere ricondotto all'unificazione nel modo seguente: innanzitutto possiamo supporre 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 è dato dalle istruzioni non deterministiche della Tabella 2.2: si noti che ad ogni applicazione di una regola, la lunghezza totale dell'insieme delle disequazioni correnti diminuisce, finchè non si raggiunge lo stato di fallimento o uno stato del tipo

$$y_1 \stackrel{?}{\preceq} t_1, \ldots, y_k \stackrel{?}{\preceq} t_k$$

con le $y_1, \ldots, y_k$ distinte. Se si raggiunge un tale stato, la soluzione del problema è data dalla sostituzione

$$y_1 \mapsto t_1, \ldots, y_k \mapsto t_k.$$

Si noti che, a differenza che per l'unificazione (dove l'unicità della soluzione vale solo a meno di una rinomina), qui la soluzione è proprio *unica* (almeno per quanto riguarda la sua azione sulle varibili presenti nel problema trattato, che sono poi le uniche che contano).

Tabella 2.2: Le regole dell'algoritmo di Matching

$$\frac{S + \{f(t_1, \ldots, t_n) \stackrel{?}{\preceq} f(u_1, \ldots, u_n)\}}{S \cup \{t_1 \stackrel{?}{\preceq} u_1, \ldots, t_n \stackrel{?}{\preceq} u_n\}}$$

$$\frac{S + \{f(t_1, \ldots, t_n) \stackrel{?}{\preceq} g(u_1, \ldots, u_m)\}}{\text{FAIL}} \quad (\text{se } f \not\equiv g);$$

$$\frac{S + \{f(t_1, \ldots, t_n) \stackrel{?}{\preceq} x\}}{\text{FAIL}}$$

$$\frac{S + \{x \preceq t, x \preceq s\}}{\text{FAIL}} \quad (\text{per } s \not\equiv t).$$

2.7 Nota bibliografica

Per ulteriori informazioni sui linguaggi del primo ordine e sulla loro semantica, si consulti un qualsiasi testo introduttivo alla Logica Matematica: ad esempio [81] utilizza la stessa definizione di verità (basata su espansioni del linguaggio, anzichè su assegnamenti infinitari) che noi abbiamo dato nel paragrafo 2.2.

Le procedure di riduzione in forma normale congiuntiva, in forma normale prenessa/antiprenessa e la skolemizzazione sono usualmente illustrate nei manuali di Logica. Per una accurata comparazione fra trasformazioni strutturali e non, si veda [7] ([36] fornisce anche risultati sperimentali in proposito). Ulteriori ottimizzazioni, rispetto all'algoritmo esposto nel paragrafo 2.5, si possono trovare in [73].

La problematica dell'unificazione si fa solitamente risalire a Herbrand [45]; il primo algoritmo (accompagnato da una dimostrazione di correttezza e di terminazione) è dovuto a Robinson [79]. Algoritmi che lavorano in tempo lineare sono stati successivamente proposti da [60], [74]. Dal punto di vista della classificazione di complessità, l'unificazione è un problema P-completo, mentre il matching è di classe NC (si veda [35]).

Infine, segnaliamo che l'area della E-unificazione (ossia dell'unificazione modulo una teoria equazionale E) è attualmente molto attiva: essa interviene ad esempio nella AC-Paramodulazone e nella E-riscrittura, settori su cui daremo qualche informazione nelle Note Bibliografiche dei Capitoli 4 e 5. Nella E-unificazione l'esistenza di upg non è garantita, nè è garantita la decidibilità stessa dei problemi di unificazione, tutto dipende dalla teoria E in questione. Siekmann [82] ha proposto una interessante classificazione delle teorie equazionali in base al loro comportamento rispetto all'E-unificazione. La teoria dell'E-unificazione è suscettibile di approcci algebrici (si veda ad esempio [42]) ed ha immediati collegamenti con problemi di decisione (modulo E) per frammenti positivi del linguaggio. Si veda il recente lavoro di rassegna [5] per tutto questo e, più in generale, per lo stato dell'arte nel settore.

Capitolo 3

Ordinamenti e vincoli

In questo Capitolo introdurremo le principali tecniche di ordinamento dei termini; tali tecniche sono uno strumento essenziale nella dimostrazione automatica, ad esempio per utilizzare le equazioni come regole di riscrittura, per definire vincoli di ottimizzazione all'applicazione delle regole di inferenza e per sviluppare un'opportuna teoria delle ridondanze nei processi di derivazione.

3.1 Generalità sugli ordinamenti

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 $\geq$ risulta essere riflessiva oltre che transitiva). L'ordine stretto $(P, >)$ è detto *terminante* qualora non esistano catene infinite decrescenti

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

di elementi di P. Un esempio di ordine stretto terminante è dato dai numeri naturali con la relazione di 'maggiore in senso stretto' (si noti che in tale esempio la relazione $>$ 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, si definisca la relazione $>$ sull'insieme $P_1 \times P_2$ mediante

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

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.[1] Lasciamo al lettore la facile verifica del fatto che $(P_1 \times P_2, >)$ è un ordine stretto e che è terminante/totale nel caso che $(P_1, >_1)$ e $(P_2, >_2)$ siano terminanti/totali a loro volta.

Prodotti lessicografici su liste: sia $(P, >)$ un ordine stretto e sia P^* l'insieme delle liste finite di elementi di P. Si definisca la relazione $>^*$ su P^* nel modo seguente: si ponga $(x_1, \ldots, x_n) >^* (y_1, \ldots, y_m)$ sse

$$n > m \vee (n = m \wedge \exists i < n\ (x_1 = y_1 \wedge \cdots \wedge x_i = y_i \wedge x_{i+1} > y_{i+1})).$$

Ossia si confrontano dapprima la lunghezza e poi (finchè è necessario) le componenti da sinistra a destra. Per verificare che $(P^*, >^*)$ è un ordine stretto (terminante o totale, nel caso in cui $(P, >)$ lo sia in partenza), si osservi dapprima che $>^*$ è il prodotto lessicografico di $>_l$ e di $\bigcup_i >^i$, dove $>_l$ è la relazione che vale fra due liste p e q sse p è più lunga di q e $>^i$ è il prodotto lessicografico di $>$ con se stesso i-volte. Infine, si osservi che la proprietà di essere un ordine stretto (terminante) si conserva per prodotti lessicografici e per unioni di relazioni che operano su insiemi disgiunti.

Ordinamento di multiinsiemi: sia $(P, >)$ un ordine stretto terminante e sia $Mul(P)$ l'insieme dei multiinsiemi finiti su P.[2] 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 x con un multiinsieme $\{y_1, \ldots, y_n\}$ di elementi tutti minori di x). 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 $M, N \in Mul(P)$

$$M > N \qquad \Leftrightarrow \qquad M >_1^+ N,$$

[1] Il prodotto lessicografico si estende facilmente al caso di $n \geq 2$ componenti.

[2] Si veda il paragrafo 2.5 per la definizione di multiinsieme.

otteniamo che $(Mul(P), >)$ è ancora un ordine stretto terminante (totale, qualora $(P, >)$ fosse già totale in partenza). Facciamo le necessarie verifiche.[3]

La transitività è ovvia per costruzione ($>$ è la chiusura transitiva di $>_1$) e l'irriflessività seguirà dalla proprietà di terminazione.

Per stabilire quest'ultima, ricorriamo all'ordine stretto ausiliario $(P', >')$ che si ottiene aggiungendo a P un minimo elemento $\bot$ e un massimo elemento $\top$ ($\top$ e $\bot$ devono essere nuovi, cioè non devono appartenere già a P); chiaramente $(P', >')$ è ancora terminante. Supponiamo che per assurdo esista una catena infinita di multiinsiemi su P tali che

$$M_1 >_1 M_2 >_1 M_3 >_1 \cdots$$

e costruiamo un albero infinito T a diramazione finita i cui nodi siano etichettati con elementi (via via descrescenti lungo ogni ramo) di P'. Se riusciamo nell'impresa, otteniamo subito una contraddizione per il Lemma di König 2.3.1. T è costruito come unione di alberi finiti $T_0 \subset T_1 \subset \cdots$. Le etichette (diverse da $\bot$) delle foglie di T_i costituiranno il multiinsieme M_i. Definiamo i T_i per induzione. T_0 è l'albero che contiene la sola radice ϵ, etichettata con $\top$. Se $M_1 = \{z_1, \ldots, z_k\}$, T_1 contiene, oltre alla radice, k nodi foglia etichettati con $z_1, \ldots, z_k$. Supponiamo di aver già costruito T_i e costruiamo T_{i+1}. Per farlo, siano $M_i = M' \cup \{x\}$ e $M_{i+1} = M' \cup \{y_1, \ldots, y_n\}$ $(n \geq 0)$ con $x > y_1, \ldots, x > y_n$. Aggiungiamo, sotto una qualsiasi foglia etichettata con x, n nuove foglie etichettate con $y_1, \ldots, y_n$ più una nuova foglia etichettata con $\bot$. In questo modo ci garantiamo che $T_i \subset T_{i+1}$ (si noti che l'inclusione è propria anche se $n = 0$). Questo conclude la definizione della successione di alberi $\{T_i\}_i$ e quindi anche la dimostrazione della proprietà di terminazione.

Si supponga ora che $(P, >)$ sia totale e siano $M_1, M_2 \in Mul(P)$. Dimostriamo che vale $M_1 > M_2$ o $M_2 > M_1$ o $M_1 = M_2$ per induzione sulla somma s delle cardinalità di M_1 e M_2.[4] Se s è 0, allora $M_1 = M_2$ sono entrambi il multiinsieme vuoto. Se $s > 0$, sia $x \in M_i$ un

[3] Abbiamo definito l'ordinamento $>$ fra multiinsiemi utilizzando la chiusura transitiva della relazione $>_1$; è possibile equivalentemente definire $>$ in modo diretto, ponendo: $M > N$ sse esistono $X, Y \in Mul(P)$ tali che $X \neq \emptyset$, $X \subseteq M$, $N = (M - X) \cup Y$ e $\forall y \in Y \exists x \in X (y < x)$. Ovviamente, abbiamo posto: i) $X \subseteq Y$ sse $\forall x \in P(X(x) \leq Y(x))$ e ii) $(M - X)(x) := M(x) \div X(x)$ (il simbolo $\div$ indica la sottrazione troncata a zero).

[4] Ovviamente la cardinalità di un multiinsieme M è la somma degli $M(x) \neq 0$ al variare di $x \in P$.

elemento massimale di $M_1 \cup M_2$. Se $x \notin M_j$ $(j \neq i)$, abbiamo $M_i > M_j$. Altrimenti, eliminiamo un'occorrenza di x sia da M_1 che da M_2, ottenendo dei multiinsiemi M_1', M_2' di cardinalità minore: avremo $M_1 > M_2$ o $M_2 > M_1$ o $M_1 = M_2$ a seconda che sia $M_1' > M_2'$ o $M_2' > M_1'$ o $M_1' = M_2'$.

3.2 Ordini di riduzione

Sia $\mathcal{L}$ 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 l'ordine stretto $>$ è un *ordine di riscrittura* qualora soddisfi le due condizioni seguenti, dette rispettivamente condizione di *compatibilità* e di *stabilità*:

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

(ii) 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 vale $f(x_1, \ldots, x_n) > x_i$ per ogni $f \in \mathcal{F}_n$.

Diamo ora due famiglie di ordini di semplificazione (che risulteranno essere anche ordini di riduzione per esempio nei linguaggi finiti) molto usate nei dimostratori automatici correnti; tali famiglie hanno anche la proprietà molto rilevante di essere ordini stretti *totali sui termini ground*. Per comodità di lettura, lasceremo però al prossimo paragrafo tutte le verifiche del caso.

Qui di seguito assumiamo che sia dato un ordine totale stretto $>_p$ (detto di *precedenza*) fra i simboli di funzione del nostro linguaggio $\mathcal{L}$.

Definizione 3.2.1 *L'ordinamento $s >_{lpo} t$ ('lexicographic path order', abbreviato LPO)*[5] *indotto dalla precedenza $>_p$ su $T_\mathcal{L}$ è così definito, per induzione su $|s| + |t|$ (cioè sulla somma delle lunghezze di s e t). Vale $s >_{lpo} t$ qualora si verifichi uno dei casi seguenti:*

[5] Questo tipo di ordinamento fa parte di una famiglia più ampia, detta famiglia RPO ('recursive path orders').

(LPO1) $s \equiv f(s_1, \ldots, s_n)$ *e per qualche* $i = 1, \ldots, n$ *vale che* $s_i >_{lpo} t$ *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 1. Usando la precedenza $a >_p s >_p 0$, possiamo verificare che[6]

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

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

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).

Si può provare che, nonostante la complicazione 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:

[6] 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.

- deve esistere $d > 0$ tale che $d = w(x)$ per ogni variabile x (cioè il peso delle variabili è sempre lo stesso numero reale strettamente 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 valere $f >_p g$ per ogni altro simbolo di funzione o di costante g di $\mathcal{L}$.

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 3.2.2 *L'ordinamento $s >_{kbo} t$ ('Knuth-Bendix order', abbreviato KBO) 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* $f \in \mathcal{F}_1$[7] *e per qualche variabile* x;

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

(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 2. 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).

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

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

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z) \qquad (y + z) \cdot x = (y \cdot x) + (z \cdot x)$$

in cui il termine minore contiene più occorrenze delle stessa variabile del termine maggiore (cosa che non è consentita dagli ordinamenti del tipo KBO).

Se la relazione di precedenza (e, per i KBO, anche la funzione peso ammissibile) è data anche sui simboli di predicato, le definizioni 3.2.1 e 3.2.2 possono essere estese nel modo ovvio alle *formule atomiche*: basta, a tal proposito, trattare i simboli di predicato come se fossero simboli di funzione nelle Definizioni 3.2.1 e 3.2.2. Ad esempio, se $f >_p P >_p Q$, avremo $Q(f(x), y) >_{lpo} P(x) >_{lpo} Q(x, x)$.

3.3 Il teorema di Kruskal

In questo paragrafo verificheremo che gli ordinamenti LPO e KBO (sui termini di un linguaggio *finito*, cioè contenente solo un numero finito di simboli di funzione) sono ordini di riduzione totali sui termini ground. Per farlo, utilizzeremo un risultato generale, interessante di per sè, che asserisce che nei linguaggi finiti gli ordini di semplificazione sono sempre terminanti (quindi sono sempre ordini di riduzione).

Il seguente Lemma è di facile verifica:

Lemma 3.3.1 *Sia $>$ un ordine di semplificazione e sia $\geq$ la chiusura riflessiva di $>$;*

(i) *per ogni termine t e per ogni $p \in Pos(t)$, si ha $t \geq t_{|p}$;*

(ii) *se $t_1 \geq s_1, \ldots, t_n \geq s_n$, allora per ogni $f \in \mathcal{F}_n$, si ha $f(t_1, \ldots, t_n) \geq f(s_1, \ldots, s_n)$.*

Un *preordine* $(P, \leq)$ è un insieme P dotato di una relazione $\leq$ riflessiva e transitiva. Il prodotto (cartesiano) di n preordini

$$(P_1, \leq_1), \ldots, (P_n, \leq_n)$$

è il preordine $(P, \leq)$, dove $P := P_1 \times \cdots \times P_n$ e $\leq$ è la relazione definita per componenti mediante

$$(a_1, \ldots, a_n) \leq (b_1, \ldots, b_n) \quad \text{sse} \quad a_i \leq b_i \text{ per ogni } i = 1, \ldots, n.$$

Una successione infinita

$$a_1, a_2, a_3, \ldots$$

in un preordine $(P, \leq)$ è detta *buona* sse esistono $i < j$ tali che $a_i \leq a_j$; è detta *stabilmente buona* sse ogni sua sottosuccessione

$$a_{i_1}, a_{i_2}, a_{i_3}, \ldots$$

(con $i_1 < i_2 < i_3 \cdots$) è buona.

Lemma 3.3.2 *Se la successione $a_1, a_2, a_3, \ldots$ (in un dato preordine $(P, \leq)$) è stabilmente buona, da essa si può estrarre una sottosuccessione*

$$a_{i_1}, a_{i_2}, a_{i_3}, \ldots$$

crescente, ossia tale che $a_{i_1} \leq a_{i_2} \leq a_{i_3} \cdots$.

Dim. Chiamiamo finale un indice m per cui non esiste nessun altro indice $n > m$ tale che $a_m \leq a_n$. Se ci fossero infiniti indici finali $m_1 < m_2 < m_3 \cdots$, allora la relativa sottosuccessione $a_{m_1}, a_{m_2}, a_{m_3}, \ldots$ non sarebbe buona. Quindi esiste $p \geq 1$ tale ogni $q \geq p$ non è finale. Sia $i_1 := p$. Supponiamo che $i_1, \ldots, i_k$ tali che $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_k}$ siano già stati definiti. Siccome $i_k \geq p$ non è finale, esiste $i_{k+1} > i_k$ tale che $a_{i_k} \leq a_{i_{k+1}}$. Così si completa per induzione la definizione della sottosuccessione crescente voluta. ⊣

Lemma 3.3.3 *Siano dati n preordini $(P_1, \leq_1), \ldots, (P_n, \leq_n)$ e sia $(P, \leq)$ il loro prodotto. Se le n successioni*

$$a_{1,1}, a_{2,1}, a_{3,1}, \ldots$$

$$a_{1,2}, a_{2,2}, a_{3,2}, \ldots$$

$$\ldots$$

$$a_{1,n}, a_{2,n}, a_{3,n}, \ldots$$

di $(P_1, \leq_1), \ldots, (P_n, \leq_n)$ sono stabilmente buone, tale è la successione di $(P, \leq)$

$$(S) \qquad (a_{1,1}, \ldots, a_{1,n}),\ (a_{2,1}, \ldots, a_{2,n}),\ (a_{3,1}, \ldots, a_{3,n}),\ \ldots.$$

Dim. Per induzione su n; se $n = 1$, l'asserto è ovvio. Se $n > 1$, possiamo supporre che la successione

$$(S') \qquad (a_{1,2}, \ldots, a_{1,n}),\ (a_{2,2}, \ldots, a_{2,n}),\ (a_{3,2}, \ldots, a_{3,n}),\ \ldots$$

sia stabilmente buona. Sia $i_1 < i_2 \cdots$ una successione crescente di indici; si tratta di provare che la corrispondente sottosuccessione di (S)

$$(b_{1,1}, \ldots, b_{1,n}),\ (b_{2,1}, \ldots, b_{2,n}),\ (b_{3,1}, \ldots, b_{3,n}),\ \ldots$$

è buona (dove $b_{k,j} := a_{i_k,j}$). Estraiamo, utilizzando il Lemma 3.3.2, dalla successione

$$b_{1,1}, b_{2,1}, b_{3,1}, \ldots$$

una sottosuccessione crescente, sia essa

$$c_{1,1} \leq_1 c_{2,1} \leq_1 c_{3,1} \leq_1 \ldots.$$

La corrispondente sottosuccessione di

$$(b_{1,2}, \ldots, b_{1,n}),\ (b_{2,2}, \ldots, b_{2,n}),\ (b_{3,2}, \ldots, b_{3,n}),\ \ldots$$

sia essa

$$(c_{1,2}, \ldots, c_{1,n}),\ (c_{2,2}, \ldots, c_{2,n}),\ (c_{3,2}, \ldots, c_{3,n}),\ \ldots$$

è buona, perchè è sottosuccessione della (S'). Quindi esistono $i < j$ tali che

$$(c_{i,2}, \ldots, c_{i,n}) \leq (c_{j,2}, \ldots, c_{j,n}).$$

In conclusione, siccome $c_{i,1} \leq_1 c_{j,1}$, abbiamo proprio la diseguaglianza

$$(c_{i,1}, \ldots, c_{i,n}) \leq (c_{j,1}, \ldots, c_{j,n}).$$

che prova l'asserto. ⊣

Teorema 3.3.4 *(di Kruskal) Sia $>$ un ordine di semplificazione su un linguaggio finito $\mathcal{L}$ e sia $\underline{x}$ un insieme finito di variabili. Consideriamo il preordine $(P, \leq)$ ottenuto restringendo la chiusura riflessiva di $<$[8] all'insieme P dei termini di $\mathcal{L}$ in cui occorrono al più le variabili $\underline{x}$. Allora ogni successione*

$$t_1, t_2, t_3, \ldots$$

di elementi di P è buona.

[8] Qui $<$ è ovviamente la relazione conversa di $>$, cioè vale $t < u$ sse $u > t$.

Dim. Supponiamo per assurdo che esista una successione non buona in P. Produciamo una successione non buona 'minimale' come segue. Sia t_1 un termine per cui esiste una successione non buona che comincia con t_1 e tale che per ogni successione non buona $t'_1, t'_2, \ldots$ si abbia $|t_1| \leq |t'_1|$.[9] Sia t_2 un termine per cui esiste una successione non buona che comincia con t_1, t_2 e tale che per ogni successione non buona $t_1, t'_2, t'_3, \ldots$ si abbia $|t_2| \leq |t'_2|$. Continuando così, produciamo la successione non buona

$$(T) \qquad t_1, t_2, t_3, \ldots$$

minimale voluta.

Siccome ci sono solo finiti simboli di funzione in $\mathcal{L}$ e siccome i termini di P contengono al più le variabili $\underline{x}$, esiste una sottosuccessione di (T)

$$(T') \qquad t_{i_1}, t_{i_2}, t_{i_3}, \ldots$$

tale che $t_{i_j} \equiv f(s_{j,1}, \ldots, s_{j,n})$ per lo stesso simbolo di funzione f che supponiamo di arietà n. Affermiamo che le successioni

$$s_{1,1}, s_{2,1}, s_{3,1}, \ldots$$

$$\ldots$$

$$s_{1,n}, s_{2,n}, s_{3,n}, \ldots$$

sono tutte stabilmente buone. Infatti, consideriamo $j = 1, \ldots, n$ e consideriamo una sottosuccessione

$$s_{k_1,j}, s_{k_2,j}, s_{k_3,j}, \ldots$$

che supponiamo non buona per assurdo. Per la minimalità della (T), la successione

$$t_1, \ldots, t_{i_{k_1}-1}, s_{k_1,j}, s_{k_2,j}, s_{k_3,j}, \ldots$$

è buona; quindi (essendo $t_1, t_2, t_3, \ldots$ e $s_{k_1,j}, s_{k_2,j}, s_{k_3,j}, \ldots$ entrambe non buone), esistono $k = 1, \ldots, i_{k_1} - 1$ e $l \geq 1$ tali che $t_k \leq s_{k_l,j} \leq t_{i_{k_l}}$ (si veda il Lemma 3.3.1 (i)), contraddizione perchè $k < i_{k_1} \leq i_{k_l}$.

Quindi le successioni

$$s_{1,1}, s_{2,1}, s_{3,1}, \ldots$$

[9] Si ricordi dal paragrafo 2.3 che $|t|$ indica la lunghezza del termine t.

$$\ldots$$

$$s_{1,n}, s_{2,n}, s_{3,n}, \ldots$$

sono tutte stabilmente buone. Per il Lemma 3.3.3, esistono degli indici $k < l$ tali che $(s_{k,1}, \ldots, s_{k,n}) \leq (s_{l,1}, \ldots, s_{l,n})$. Per il Lemma 3.3.1 (ii), si ha allora

$$t_{i_k} \equiv f(s_{k,1}, \ldots, s_{k,n}) \leq f(s_{l,1}, \ldots, s_{l,n}) \equiv t_{i_l},$$

assurdo perchè $i_k < i_l$. ⊣

Teorema 3.3.5 *Ogni ordine di semplificazione sui termini di un linguaggio finito è un ordine di riduzione.*

Dim. Sia

$$t_1 > t_2 > t_3 > \cdots$$

una successione infinita strettamente descrescente. Osserviamo che per ogni i, tutte le variabili che occorrono in t_{i+1} occorrono anche in t_i: infatti, se x occorre in t_{i+1} ma non in t_i, allora la sostituzione $\sigma : x \mapsto t_i$ (per il Lemma 3.3.1 e per le altre proprietà degli ordini di semplificazione) porta all'assurdo $t_i \equiv t_i\sigma > t_{i+1}\sigma$ perchè $t_{i+1}\sigma$ contiene t_i come sottotermine. Quindi, in tutta la successione occorre solo un numero finito di variabili (quelle già presenti in t_1). Per il Teorema 3.3.4, esistono degli indici $i < j$ con $t_i \leq t_j$, contro al fatto che $t_i > t_j$ (si ricordi che $>$ è irriflessiva, oltre che transitiva). ⊣

Possiamo ora verificare le proprietà rilevanti degli ordinamenti introdotti nel paragrafo precedente.

Teorema 3.3.6 *L'ordinamento $>_{lpo}$ indotto sui termini di un linguaggio $\mathcal{L}$ da una relazione di precedenza totale $>_p$ è un ordine di semplificazione che è totale sui termini ground. Quindi, se $\mathcal{L}$ è finito, tale ordinamento è anche un ordine di riduzione.*

Dim. Dimostriamo la *transitività*; siano s, t, u termini tali che $s >_{lpo} t >_{lpo} u$ e operiamo per induzione su $|s|+|t|+|u|$. Si tratta di esaminare tutti i casi possibili.

Caso 1. La relazione $s >_{lpo} t$ è stabilita mediante (LPO1), quindi $s \equiv f(s_1, \ldots, s_n)$ e vale $s_i \geq_{lpo} t$ per qualche i. Per ipotesi induttiva, abbiamo $s_i \geq_{lpo} u$ e quindi $s >_{lpo} u$.

Caso 2. La relazione $s >_{lpo} t$ è stabilita mediante (LPO2), quindi $s \equiv f(s_1, \ldots, s_n)$, $t \equiv g(t_1, \ldots, t_m)$, $f >_p g$ e inoltre vale $s >_{lpo} t_i$ per ogni $i = 1, \ldots, m$. Abbiamo tre sottocasi, a seconda del motivo per cui vale la relazione $t >_{lpo} u$.

Sottocaso 2.1. La relazione $t >_{lpo} u$ è stabilita mediante (LPO1), quindi $t \equiv g(t_1, \ldots, t_m)$ e vale $t_i \geq_{lpo} u$ per qualche i. Siccome avevamo anche $s >_{lpo} t_i$, per ipotesi induttiva segue $s >_{lpo} u$.

Sottocaso 2.2. La relazione $t >_{lpo} u$ è stabilita mediante (LPO2), quindi $u \equiv h(u_1, \ldots, u_l)$, $g >_p h$ e inoltre $t >_{lpo} u_i$ per ogni $i = 1, \ldots, l$. Segue $f >_p h$ e $s >_{lpo} u_i$ per ogni $i = 1, \ldots, l$ per ipotesi induttiva da $s >_{lpo} t >_{lpo} u_i$; quindi $s >_{lpo} u$ vale grazie ancora a (LPO2).

Sottocaso 2.3. La relazione $t >_{lpo} u$ è stabilita mediante (LPO3), quindi $u \equiv g(u_1, \ldots, u_n)$, e per qualche $i = 1, \ldots, n$ vale che $t_1 \equiv u_1, \ldots, t_{i-1} \equiv u_{i-1}, t_i >_{lpo} u_i, t >_{lpo} u_{i+1}, \ldots, t >_{lpo} u_n$. La relazione $s >_{lpo} u$ vale grazie a (LPO2): infatti per $k \leq i$ abbiamo $t >_{lpo} u_i$ (grazie a (LPO1)) e quindi per ipotesi induttiva $s >_{lpo} t >_{lpo} u_i$, mentre per $k > i$ è sufficiente invocare l'ipotesi induttiva direttamente.

Caso 3. La relazione $s >_{lpo} t$ è stabilita mediante (LPO3), quindi $s \equiv f(s_1, \ldots, s_n)$, $t \equiv f(t_1, \ldots, t_n)$ e per qualche $i = 1, \ldots, n$ vale che $s_1 \equiv t_1, \ldots, s_{i-1} \equiv t_{i-1}, s_i >_{lpo} t_i, s >_{lpo} t_{i+1}, \ldots, s >_{lpo} t_n$. Abbiamo ancora tre sottocasi, a seconda del motivo per cui vale la relazione $t >_{lpo} u$.

Sottocaso 3.1. La relazione $t >_{lpo} u$ è stabilita mediante (LPO1), quindi vale $t_j \geq_{lpo} u$ per qualche j. Siccome abbiamo $s >_{lpo} t_j$ (direttamente per $j > i$ o per (LPO1) se $j \leq i$), per ipotesi induttiva si ricava $s >_{lpo} u$.

Sottocaso 3.2. La relazione $t >_{lpo} u$ è stabilita mediante (LPO2), quindi $u \equiv g(u_1, \ldots, u_l)$, $f >_p g$ e inoltre $t >_{lpo} u_j$ per ogni $j = 1, \ldots, l$. Allora la relazione $s >_{lpo} u$ vale anch'essa per (LPO2) perchè da $s >_{lpo} t >_{lpo} u_j$ segue subito $s >_{lpo} u_j$ per ipotesi induttiva.

Sottocaso 3.3. Supponiamo infine che le relazioni $s >_{lpo} t >_{lpo} u$ siano entrambe stabilite grazie a (LPO3). Quindi $t \equiv f(t_1, \ldots, t_n)$, $u \equiv f(u_1, \ldots, u_n)$ ed esiste $j = 1, \ldots, n$ tale che $t_l \equiv u_l$ (per $l < j$), $t_j >_{lpo} u_j$, $t >_{lpo} u_l$ (per $l > j$). Ci sono 3 casi a seconda che $i < j$ o $i = j$ o $i > j$. Sia $i < j$: per $k < i$ si ha $s_k \equiv t_k \equiv u_k$, mentre per $k = i$, si ha $s_i >_{lpo} t_i \equiv u_i$ (quindi $s_i >_{lpo} u_i$) e infine per $k > i$ abbiamo $s >_{lpo} u_k$ per transitività da $s >_{lpo} t$ e $t >_{lpo} u_k$ (quest'ultima relazione vale o direttamente, se $k > j$, o si deduce se $k \leq j$ per (LPO1) da $t_k \geq_{lpo} u_k$). Nel caso $i = j$, abbiamo $s_k \equiv t_k \equiv u_k$ per $k < i$, $s_i >_{lpo} u_i$ per transitività e $s >_{lpo} u_k$ per transitività da $s >_{lpo} t >_{lpo} u_k$ nel caso

$k > i$. Nel caso $i > j$, abbiamo $s_k \equiv t_k \equiv u_k$ per $k < j$,$s_j \equiv t_j >_{lpo} u_j$ per $k = j$ e infine se $k > j$, $s >_{lpo} u_k$ vale per transitività da $s >_{lpo} t$ e $t >_{lpo} u_k$. In tutti e tre i casi $s >_{lpo} u$ vale sempre per (LPO3).

Quanto detto termina la verifica della proprietà di transitività. Vediamo le rimanenti condizioni.

Mediante (LPO1) è possibile stabilire facilmente per induzione che vale $t >_{lpo} t_{|p}$ per ogni $p \in Pos(t)$, $p \neq \epsilon$ (*proprietà del sottotermine*). Grazie a questa proprietà è banale stabilire l'*irriflessività* $t \not>_{lpo} t$ per induzione su $|t|$: basta infatti osservare che nessuna delle (LPO1)-(LPO3) è applicabile (si noti che per (LPO1), se fosse $t \equiv f(t_1, \ldots, t_n)$ e $t_i >_{lpo} t$ avremmo $t_i >_{lpo} t_i$ per la transitività appena stabilita).

Per la condizione di *stabilità*, siano s, t tali che valga $s >_{lpo} t$ e sia σ una sostituzione. La relazione $s\sigma >_{lpo} t\sigma$ si stabilisce facilmente per induzione su $|s| + |t|$, analizzando i tre casi (LPO1)-(LPO3).

Per la condizione di *compatibilità*, siano $f \in \mathcal{F}_n$ e siano dati dei termini $s_1, s_2, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ tali che $s_1 > s_2$. Allora

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

vale per (LPO3) e per la proprietà del sottotermine.

Quest'ultima proprietà (o una banale osservazione diretta tramite (LPO1)) garantisce anche che $>_{lpo}$ è un *ordine di semplificazione*.

Resta da verificare che $>_{lpo}$ è *totale sui termini ground* (qui è essenziale l'ipotesi che la relazione di precedenza $>_p$ sia totale a sua volta). Siano $s \equiv f(s_1, \ldots, s_n)$ e $t \equiv g(t_1, \ldots, t_m)$ termini ground distinti.[10] Lavoriamo per induzione su $|s| + |t|$. Per ipotesi induttiva, s è confrontabile con $t_1, \ldots, t_m$ e deve essere $s >_{lpo} t_j$ per ogni j (se no, $t >_{lpo} s$). Analogamente, si avrà $t > s_j$ per ogni j. Se $f \not\equiv g$, avremo $s >_{lpo} t$ o $t >_{lpo} s$ a seconda che $f >_p g$ o che $g >_p f$. Se $f \equiv g$, ci sarà un minimo j tale che s_j e t_j sono differenti; a seconda che $s_j >_{lpo} t_j$ o che $t_j >_{lpo} s_j$, avremo $s >_{lpo} t$ o $t >_{lpo} s$. ⊣

Avendo dato nei dettagli la dimostrazione del Teorema precedente, lasciamo al lettore la verifica (laboriosa, ma non particolarmente difficile) dell'analoga proprietà degli ordinamenti KBO:

Teorema 3.3.7 *L'ordinamento $>_{kbo}$ indotto sui termini di un linguaggio $\mathcal{L}$ da una funzione peso ammissibile w e da una relazione di precedenza totale $>_p$ è un ordine di semplificazione che è totale sui termini*

[10] I casi $n = 0$, $m = 0$ tengono conto delle costanti.

ground. Quindi, se $\mathcal{L}$ è finito, tale ordinamento è anche un ordine di riduzione.

La verifica della proprietà di terminazione in entrambi i Teoremi 3.3.6 e 3.3.7 si basa sul Teorema di Kruskal che richiede, per essere applicato, che il linguaggio sia finito. Se il linguaggio non è finito, esistono banali controesempi alla suddetta proprietà di terminazione: ad esempio, per gli ordinamenti LPO, se il linguaggio contiene infinite costanti C e la relazione di precedenza, ristretta a C, non è terminante, è chiaro che la proprietà di terminazione non vale. È tuttavia possibile fare delle estensioni dei teoremi che abbiamo provato anche al caso di linguaggi infiniti, basandosi sulla formulazione originaria estesa del Teorema di Kruskal. Senza addentrarci ad illustrare tali estensioni, menzioniamo qui una semplice osservazione che funziona nel caso (che ci servirà nel Capitolo 5) in cui $\mathcal{L}$ contenga infinite costanti e finiti simboli di funzione di arietà maggiore o uguale a 1. Sia quindi $\mathcal{L}$ uguale a $\mathcal{L}_0 + C$, dove $\mathcal{L}_0$ è finito e $C = \{d_0, d_1, d_2, \dots\}$ è un insieme infinito numerabile di (nuove) costanti. Consideriamo l'ordinamento LPO indotto da una relazione di precedenza totale $>_p$ sui simboli di $\mathcal{L}$ che sia terminante.[11] La dimostrazione del Terema 3.3.4 richiede solo la seguente piccola modifica (mentre le dimostrazioni dei successivi Teoremi 3.3.5 e 3.3.6 non richiedono cambiamenti): data una successione non buona

$$(T) \qquad t_1, t_2, t_3, \dots$$

di termini contenenti solo le variabili $\underline{x}$, possiamo ancora affermare che esiste una sottosuccessione di (T)

$$t_{i_1}, t_{i_2}, t_{i_3}, \dots$$

tale che $t_{i_j} \equiv f(s_{j,1}, \dots, s_{j,n})$ per lo stesso simbolo di funzione f. Se così non fosse, infatti, potremmo estrarre dalla (T) una sottosuccessione di costanti

$$d_{i_1}, d_{i_2}, d_{i_3}, \dots$$

contro al fatto che la (T) non è buona e che la relazione $>_p$ è terminante e totale.

[11] Un analogo ragionamento vale anche per gli ordinamenti KBO, utilizzando un'adeguata funzione peso.

3.4 Vincoli simbolici

La caratteristica principale a cui è dovuta l'efficienza del calcolo che presenteremo nel prossimo Capitolo è l'uso di inferenze soggette a vincoli simbolici.

Fissiamo un linguaggio $\mathcal{L}$ e un ordine di riduzione che sia totale sui termini ground di $\mathcal{L}$ (di norma, pensiamo ad un LPO o ad un KBO). Un *vincolo atomico* è una equazione o una disequazione fra due termini t, u, ossia è una scrittura del tipo $t \stackrel{?}{=} u$ o del tipo $t \stackrel{?}{>} u$. Un *vincolo* è un insieme finito di vincoli atomici, che viene scritto congiuntivamente nella forma:

$$V \qquad t_1 \stackrel{?}{=} u_1 \;\&\; \cdots \;\&\; t_n \stackrel{?}{=} u_n \;\&\; s_1 \stackrel{?}{>} v_1 \;\&\; \cdots \;\&\; s_m \stackrel{?}{>} v_m$$

Diciamo che le n equazioni $t_1 \stackrel{?}{=} u_1 \;\&\; \cdots \;\&\; t_n \stackrel{?}{=} u_n$ costituiscono la *parte equazionale* V_e del vincolo V e che le m disequazioni $s_1 \stackrel{?}{>} v_1 \;\&\; \cdots \;\&\; s_m \stackrel{?}{>} v_m$ costituiscono la *parte stretta* V_s del vincolo V.

Introduciamo due differenti nozioni di soddisfacibilità del vincolo V, cominciando dalla più ovvia. Diciamo che il vincolo V è *attualmente soddisfacibile* qualora esista una sostituzione *ground* σ tale che

$$t_1\sigma \equiv u_1\sigma \;\&\; \cdots \;\&\; t_n\sigma \equiv u_n\sigma \;\&\; s_1\sigma > v_1\sigma \;\&\; \cdots \;\&\; s_m\sigma > v_m\sigma.$$

Poichè è bene utilizzare nozioni che risultino stabili rispetto ad estensioni del linguaggio (spesso infatti nella pratica della dimostrazione automatica vengono aggiunti nuovi assiomi in tempi diversi che possono coinvolgere nuovi simboli di funzione), risulta utile introdurre anche un'altra meno restrittiva nozione di soddisfacilità. Per *estensione* del linguaggio $\mathcal{L}$ e dell'ordine di riduzione $>$ che opera su $T_{\mathcal{L}}$ intendiamo un linguaggio $\mathcal{L}'$ contenente tutti i simboli di funzione e di predicato di $\mathcal{L}$ più altri ancora ed un ordine di riduzione $>'$ (sempre totale sui termini ground) che coincida con $>$ sui termini di $\mathcal{L}$.[12] Diciamo allora che il vincolo V è *soddisfacibile in segnature estese* qualora esistano un'estensione $\mathcal{L}'$ di $\mathcal{L}$, un'estensione $>'$ di $>$ ed una sostituzione σ a

[12] Si dà per sottointeso, in questa definizione, che $>'$ appartenga alla stessa famiglia di $>$, ossia che sia ancora un LPO, KBO, ecc. se lo era $>$. In tali casi si assume implicitamente anche che $>'$ sia indotto da una precedenza (e, nel caso dei KBO, anche da una funzione peso) che semplicemente estenda la vecchia precedenza (risp. la vecchia funzione peso) ai nuovi simboli di $\mathcal{L}'$.

valori nei termini ground di $\mathcal{L}'$, tali che

$$t_1\sigma \equiv u_1\sigma \;\&\; \cdots \;\&\; t_n\sigma \equiv u_n\sigma \;\&\; s_1\sigma >' v_1\sigma \;\&\; \cdots \;\&\; s_m\sigma >' v_m\sigma.$$

Come vedremo nel prossimo Capitolo, l'insoddisfacibilità di un vincolo serve a bloccare l'esecuzione di un'inferenza inutile. Le due nozioni precedenti di soddisfacibilità sono sì decidibili per gli ordinamenti più comuni, ma i relativi problemi sono NP-completi. Quindi il vantaggio di bloccare un'inferenza (che potrebbe a sua volta produrne altre a catena, tutte inutili), si può perdere a causa della complessità del problema di risolvere il relativo vincolo. Per questo motivo, è bene avere a disposizione *nozioni blande* di soddisfacibilità di un vincolo che hanno il vantaggio di essere trattabili in tempo polinomiale e di operare sufficientemente bene nei casi concreti. Una nozione blanda si può formulare ad esempio così: diciamo che il vincolo V è *pseudo-soddisfacibile* qualora valga la condizione

$$s_1\mu \not\leq v_1\mu \;\&\; \cdots \;\&\; s_m\mu \not\leq v_m\mu,$$

dove μ è upg del problema di unificazione dato dalla parte equazionale V_e di V.[13]

I vincoli possono essere utilizzati per definire nuovi tipi di ordini di riduzione molto potenti. Se $>$ è un ordine di riduzione totale sui simboli ground di un linguaggio $\mathcal{L}$, definiamo (per $s, t \in T_{\mathcal{L}}$) $s \succ t$ sse i vincoli $t \overset{?}{>} s$ e $t \overset{?}{=} s$ sono entrambi insoddisfacibili (in senso attuale o esteso). Altrimenti detto, $s \succ t$ vale sse per ogni sostituzione ground σ si ha che $s\sigma > t\sigma$. Si noti che l'ordine di riduzione che così si ottiene è più forte dell'ordine di riduzione $>$ di partenza, come testimonia l'Esempio 3 che presenteremo più oltre.

A conclusione del paragrafo, studiamo un algoritmo recente che testa *la soddisfacibilità dei vincoli in segnature estese relativamente a ordini di riduzione del tipo LPO indotti da una precedenza totale* $>_p$. Esponiamo l'algoritmo nella forma che ci sembra più semplice da spiegare e da giustificare. Useremo anche ora regole di trasformazione non deterministiche, come nel caso dell'unificazione, con una importante differenza però. Nel caso dell'unificazione, due diverse esecuzioni esaustive delle regole portavano comunque allo stesso risultato, mentre nell'algoritmo che proponiamo ora il fallimento di un'esecuzione esaustiva delle

[13] La condizione di pseudo-soddisfacibilità va interpretata ricordando che $s \leq t$ (per due termini s, t) significa '$s \equiv t$ oppure $s < t$'.

regole non significa di per sè l'insoddisfacibilità del vincolo in ingresso: per stabilire quest'ultima occorre verificare che *tutte* le esecuzioni esaustive possibili portano a fallimento. Si tratta quindi di un esempio di non-determinismo 'don't know' (in contrasto con il non-determinismo 'don't care' dell'algoritmo di unificazione).

Dato un vincolo V che si vuole testare per la soddisfacibilità, lo si manipola secondo le istruzioni della Tabella 3.1 seguente, al fine di ottenere un vincolo completamente analizzato (si veda poco oltre per la relativa definizione). Un vincolo V contiene un *ciclo* sse contiene vincoli atomici stretti del tipo $x_0 \stackrel{?}{>} t_1, x_1 \stackrel{?}{>} t_2, \ldots, x_{n-1} \stackrel{?}{>} t_n$, dove in $t_1, \ldots, t_n$ occorrono rispettivamente le variabili $x_1, \ldots, x_{n-1}, x_0$: chiaramente un vincolo che contiene un ciclo è insoddisfacibile perchè gli LPO sono ordini di semplificazione.

Un vincolo atomico stretto $s \stackrel{?}{>} t$ occorrente in V è detto *analizzato in* V, qualora si verifichi una delle seguenti condizioni:

- s è una variabile o t è una variabile;
- $t \equiv g(t_1, \ldots, t_m)$, $top(s) >_p g$ e $\{s \stackrel{?}{>} t_1, \ldots, s \stackrel{?}{>} t_n\} \subseteq V$;
- $t \equiv f(t_1, \ldots, t_n)$, $s \equiv f(s_1, \ldots, s_n)$ ed esiste i tale che

$$\{s_1 \stackrel{?}{=} t_1, \ldots, s_{i-1} \stackrel{?}{=} t_{i-1}, s_i \stackrel{?}{>} t_i, s \stackrel{?}{>} t_{i+1}, \ldots, s \stackrel{?}{>} t_n\} \subseteq V;$$

- $s \equiv f(s_1, \ldots, s_n)$, $top(t) \geq_p f$ ed esiste i tale che $s_i \stackrel{?}{=} t \in V$ oppure che $s_i \stackrel{?}{>} t \in V$;
- $s \not\equiv c$, per una costante c tale che $top(t) \geq_p c$.

Un vincolo atomico equazionale $s \stackrel{?}{=} t$ è *banale* sse $s \equiv t$. Un vincolo V è *completamente analizzato* sse

(1) V consta solo di vincoli atomici equazionali banali o di vincoli atomici stretti che sono analizzati in V;

(2) se V contiene i vincoli atomici stretti $s \stackrel{?}{>} x$ e $x \stackrel{?}{>} t$, allora $s \stackrel{?}{>} t \in V$;

(3) V non contiene cicli.

Tabella 3.1: Test di Soddisfacibilità per Vincoli LPO

(i) $\dfrac{V}{V \cup \{s \overset{?}{>} t_1, \ldots, s \overset{?}{>} t_n\}}$
se $s \overset{?}{>} g(t_1, \ldots, t_m) \in V$ e $top(s) >_p g$;

(ii) $\dfrac{V}{V \cup \{s_1 \overset{?}{=} t_1, \ldots, s_{i-1} \overset{?}{=} t_{i-1}, s_i \overset{?}{>} t_i, s \overset{?}{>} t_{i+1}, \ldots, s \overset{?}{>} t_n\}}$
(per un $i = 1, \ldots, n$), se $f(s_1, \ldots, s_n) \overset{?}{>} f(t_1, \ldots, t_n) \in V$;

(iii) $\dfrac{V}{V \cup \{s_i \overset{?}{=} t\}}$
(per un $i = 1, \ldots, n$), se $f(s_1, \ldots, s_n) \overset{?}{>} t \in V$ e $top(t) \geq_p f$;

(iv) $\dfrac{V}{V \cup \{s_i \overset{?}{>} t\}}$
(per un $i = 1, \ldots, n$), se $f(s_1, \ldots, s_n) \overset{?}{>} t \in V$ e $top(t) \geq_p f$;

(v) $\dfrac{V}{FAIL}$
se $c \overset{?}{>} t \in V$, per una costante c tale che $top(t) \geq_p c$;

(vi) $\dfrac{V}{V\mu}$ se μ è upg di V_e;

(vii) $\dfrac{V}{FAIL}$ se V_e non è unificabile;

(viii) $\dfrac{V}{FAIL}$ se V contiene un ciclo;

(ix) $\dfrac{V}{V \cup \{s \overset{?}{>} t\}}$ se $s \overset{?}{>} u \in V$ e $u \overset{?}{>} t \in V$, per qualche u.

Si noti che (2) richiede che V sia chiuso solo per applicazioni di una *forma ristretta* dell'istruzione (ix) della Tabella 3.1.

Un'*esecuzione* (dell'algoritmo della Tabella 3.1), è una successione (finita o infinita) di vincoli

$$(\delta) \qquad V_0, V_1, \ldots, V_i, \ldots$$

ciascuno dei quali è ottenuto dal precedente grazie all'applicazione di una delle istruzioni della Tabella 3.1.

Un'esecuzione (δ) è *esaustiva* sse è finita e inoltre o termina in stato di fallimento oppure il vincolo con cui finisce è completamente analizzato.

Risulta che un vincolo V_0 è soddisfacibile in segnature estese (relativamente ad ordinamenti del tipo LPO indotti da relazioni di precedenza totali) sse esiste *una* esecuzione esaustiva che non fallisce e che comincia con V_0:[14] lasceremo la dimostrazione di questo fatto (come pure la discussione sulla terminazione del processo di ricerca di un'esecuzione esaustiva) al prossimo paragrafo e vediamo ora invece un esempio.

ESEMPIO 3. Proviamo che, data la precedenza $f >_p g$, il vincolo

$$V \qquad g(g(x,y), g(x,y)) \stackrel{?}{>} g(f(x), f(y))$$

non è soddisfacibile in senso esteso (quindi neppure in senso attuale) nell'ordinamento $>_{lpo}$ indotto da $>_p$.[15] Applicando le regole (iii) o (iv) a V, si produce un vincolo del tipo $g(x,y) \stackrel{?}{\geq} g(f(x), f(y))$, che si vede subito fallire comunque si prosegua. Non resta che applicare la (ii) per $i = 1$ (per $i = 2$, si produrrebbe subito il problema non unificabile $g(x,y) \stackrel{?}{=} f(x)$), passando a

$$(a) \;\; g(x,y) \stackrel{?}{>} f(x), \quad (b) \;\; g(g(x,y), g(x,y)) \stackrel{?}{>} f(y).$$

[14] Le regole non sono invece sufficienti ad assicurare la soddisfacibilità attuale del vincolo: se ad esempio il linguaggio $\mathcal{L}$ contiene il solo termine ground c, un vincolo come $\{x \stackrel{?}{>} y\}$ (considerato completamente analizzato dal nostro algoritmo) non sarebbe attualmente soddisfacibile. Per l'attuale soddisfacibilità occorrono regole ulteriori che dipendono da come è fatta la precedenza $>_p$ su cui il nostro LPO è basato.

[15] Si osservi invece che lo stesso vincolo è pseudo-soddisfacibile.

Essendo $f >_p g$, la condizione (b) può essere analizzata solo con la regola (iv) (la (iii) produce il problema non unificabile $g(x,y) \stackrel{?}{=} f(y)$), ottenendo $g(x,y) \stackrel{?}{>} f(y)$. Di nuovo, solo le regole (iii) e (iv) sono applicabili, per di più solo per $i = 1$ (per $i = 2$, si va a fallire con $y \stackrel{?}{\geq} f(y)$). Si ottiene quindi $x \stackrel{?}{\geq} f(y)$. La (a), analizzata sempre secondo le regole (iii) o (iv), produce $x \stackrel{?}{\geq} f(x)$ (che fallisce) oppure $y \stackrel{?}{\geq} f(x)$. Ora, le condizioni

$$x \stackrel{?}{\geq} f(y), \quad y \stackrel{?}{\geq} f(x)$$

portano comunque a fallimento in tutti e quattro i casi (ossia, sia interpretando $\stackrel{?}{\geq}$ come $\stackrel{?}{>}$ sia interpretandolo come $\stackrel{?}{=}$), a causa dell'insorgere di cicli o di problemi non unificabili.

3.5 Un test di soddisfacibilità

Proviamo innanazitutto che un'esecuzione dell'algoritmo della Tabella 3.1 può essere infinita solo per motivi banali.

Un'esecuzione

$$(\delta) \qquad V_0, V_1, \ldots, V_i, \ldots$$

è *non-ridondante* sse per ogni $i \geq 0$ si ha che $V_{i+1} \neq V_i$.

Proposizione 3.5.1 *Un'esecuzione non-ridondante è finita ed esistono solo finite esecuzioni non-ridondanti.*

Dim. Si noti innanzitutto che le istruzioni della Tabella 3.1 non introducono mai variabili che non siano già presenti nel vincolo cui si applicano. Un'applicazione dell'istruzione (vi) è detta *banale* sse V_e contiene solo equazioni banali sse l'upg di V_e è l'identità.[16] In un'esecuzione non-ridondante $V_0, V_1, \cdots$ non sono possibili applicazioni banali della (vi) e, siccome ogni applicazione non banale della (vi) riduce il

[16] Ripercorrendo le regole che non producono fallimento dell'algoritmo di unificazione (si veda la Tabella 2.1), si prova facilmente che, se la premessa di una di tali regole contiene un'equazione del tipo $t \stackrel{?}{=} u$ per $t \not\equiv u$, anche la conclusione ne deve contenere una. Perciò l'upg finale è l'identità sse il problema in ingresso è banale.

numero delle variabili presenti nel vincolo corrente, sono possibili solo finite applicazioni non banali della (vi). Sia V_k la conclusione dell'ultima di esse. Proviamo che l'esecuzione

$$(\delta') \qquad V_k, V_{k+1}, \ldots, V_i, \ldots$$

è finita. Siccome la (vi) non si applica più in (δ'), per come sono fatte le altre istruzioni, nella (δ') possono comparire solo sottotermini che siano già presenti in V_k (sia N il loro numero). Si noti ora che ogni applicazione di un'istruzione diversa dalla (vi) provoca immediato fallimento o aumenta (in un'esecuzione non ridondante) la cardinalità dell'insieme dei vincoli correnti. Tale cardinalità nella (δ') non può superare $2N^2$, per cui la (δ') è finita.

Il fatto che esistano solo finite applicazioni non ridondanti segue dal Lemma di König 2.3.1 e dal fatto che ad ogni dato vincolo V si possono applicare solo finite istruzioni della Tabella 3.1. ⊣

Nella dimostrazione, siamo dovuti ricorrere al lemma di König perchè dal ragionamento fatto non emergeva chiaramente nessun limite predeterminato alla lunghezza di un'esecuzione non ridondante. In realtà tale limite si può trovare raffinando l'argomentazione: ad esempio, se rappresentiamo i termini come grafi aciclici diretti, si può facilmente vedere che in un'esecuzione non ridondante vengono mantenuti, oltre al dag che rappresenta i termini, solo due insiemi I_1, I_2 di coppie di nodi del dag corrispondenti ai termini s, t per cui si è appurato che $s \overset{?}{>} t$ o, rispettivamente, che $s \overset{?}{=} t$. Ad ogni esecuzione di un'istruzione, vengono identificati dei nodi nel dag (con l'istruzione (vi)) oppure vengono semplicemente aggiunte nuove coppie di nodi a I_1 o a I_2, sicchè il tutto richiede spazio (e di fatto anche tempo) polinomiale.[17] Così, alla luce del seguente Teorema 3.5.5, si può provare che il nostro algoritmo è di classe NP (come c'era da aspettarsi, visto che il problema della soddisfacibilità di vincoli LPO in segnature estese è noto essere, in quanto tale, un problema NP-completo).

Lemma 3.5.2 *Se V_0 è soddisfacibile in segnature estese, allora esiste un'esecuzione esaustiva e non ridondante che comincia con V_0 e che non fallisce.*

[17] Si osservi che sono possibili grossi miglioramenti nell'implementazione dell'algoritmo di soddisfacibilità, rispetto all'esposizione che ne abbiamo dato (ad esempio, si potrebbe prevedere la cancellazione - e non il semplice accumulo - dei vincoli atomici via via analizzati). La formulazione che abbiamo scelto ha tuttavia il merito di rendere le dimostrazioni del presente paragrafo più semplici.

Dim. Sia σ una sostituzione ground che soddisfa V_0. Se V_0 non è già di per sè completamente analizzato, c'è qualche istruzione della Tabella 3.1 che non porta a fallimento e che è applicabile (in modo non ridondante) a V_0. Si usino la σ e la definizione 3.2.1 per determinare un'istruzione V_0/V_1 tale che σ soddisfa ancora V_1[18] (nel caso dell'istruzione (vi), non sarà proprio la σ a soddisfare V_1, ma una σ' tale che $\mu\sigma' = \sigma$). Si prosegua così, fino ad ottenere un'esecuzione esaustiva. $\dashv$

[Apriamo un inciso per fissare alcune informazioni che utilizzeremo molto più oltre nel paragrafo 5.4. Se

$$(\delta) \qquad V_0, V_1, \ldots, V_n$$

è un'esecuzione dell'algoritmo della Tabella 3.1 che utilizza esattamente nei passi $i_1, \ldots, i_k$ l'istruzione (vi) con rispettivi upg $\mu_1, \ldots, \mu_k$, la coppia (V_δ, μ_δ) è definita da

$$V_\delta := V_n, \qquad \mu_\delta := \mu_1 \cdots \mu_k.$$

L'*analisi completa* del vincolo soddisfacibile V è data dall'insieme delle coppie (V_δ, μ_δ), al variare delle esecuzioni esaustive e non ridondanti che cominciano con V e che non falliscono. La dimostrazione del Lemma 3.5.2 fornisce allora la seguente informazione

Lemma 3.5.3 *La sostituzione σ soddisfa il vincolo V_0 sse esiste una coppia (V, μ) nell'analisi completa di V_0 tale che $\sigma = \mu\sigma'$ per qualche σ' che soddisfa V.*

Torniamo ora all'argomento principale di questo paragrafo, ossia alla giustificazione dell'algoritmo della Tabella 3.1.]

Proviamo che un vincolo completamente analizzato U ha soluzione in una segnatura estesa. Si consideri il linguaggio $\mathcal{L}_C$ ottenuto dal linguaggio originario $\mathcal{L}$ aggiungendo una nuova costante 0 e un nuovo simbolo di funzione unaria *succ*; si estenda la precedenza originaria $>_p$ ponendo $f >_p succ >_p 0$ per ogni vecchio simbolo di funzione f. Si definisca un ordine stretto terminante R^+ fra le variabili occorrenti in U mediante la chiusura transitiva della relazione xRy che vale fra x e y se e solo se

[18] Sembra che nella Tabella 3.1 e nella definizione di vincolo atomico stretto analizzato manchi un caso, ma non è così. Il caso sarebbe quello in cui V_0 contiene il vincolo atomico $s \stackrel{?}{>} t$, con $s \equiv f(s_1, \ldots, s_n)$, $t \equiv g(t_1, \ldots, t_m)$ e $f >_p g$. Si supponga che σ risolva $s \stackrel{?}{>} t$ in virtù di (LPO1) (ossia perchè vale $s_i\sigma \geq t\sigma$ per qualche i). In tal caso però, anche (LPO2) si applica: per ogni j, abbiamo infatti $s\sigma > s_i\sigma \geq t\sigma > t_j\sigma$.

'esiste un termine t tale che y occorre in t e tale che $x \overset{?}{>} t \in U$'.
Si noti che la R^+ è terminante perchè nel vincolo completamente analizzato U non sono presenti cicli.

La *soluzione canonica* σ di U viene calcolata induttivamente su R^+ mediante la definizione:

$$x\sigma \equiv \begin{cases} 0, & \text{se } U \text{ non contiene disequazioni del tipo } x \overset{?}{>} t; \\ succ(u), & \text{se } u \equiv max_{>}\{t\sigma \mid x \overset{?}{>} t \in U\}, \text{ altrimenti.} \end{cases}$$

Proposizione 3.5.4 *Se U è un vincolo completamente analizzato, la soluzione canonica di U soddisfa U.*

Dim. Proviamo che la σ soddisfa tutti i vincoli atomici di U. Siccome U è completamente analizzato, tutti i vincoli atomici equazionali di U saranno banali e perciò automaticamente soddisfatti dalla σ. Per ogni vincolo atomico stretto $s \overset{?}{>} t$ presente in U, verifichiamo che vale $s\sigma >_{lpo} t\sigma$ per induzione noetheriana sulla coppia:

$$\langle Var(s) \cup Var(t), |s| + |t| \rangle$$

(tali coppie sono supposte ordinate lessicograficamente, utilizzando sulle prime componenti l'estensione ai multiinsiemi della relazione R^+ utilizzata nella definizione di soluzione canonica).

Sia dunque $s \overset{?}{>} t \in U$; siccome U è completamente analizzato, se entrambi i termini s e t non sono variabili, U deve contenere i vincoli atomici del conseguente di una delle istruzioni (i), (ii), (iii), (iv). Tali vincoli atomici sono sufficienti nel loro complesso (per la Definizione 3.2.1) ad implicare $s\sigma >_{lpo} t\sigma$, una volta che si sia appurato che essi sono soddisfatti da σ. Ma questo è ovvio per i vincoli atomici equazionali (che sono sempre banali in U) e vale per ipotesi induttiva per i vincoli atomici stretti prodotti dalle (i), (ii), (iii), (iv).

Restano quindi solo da analizzare i casi in cui s o t sia una variabile.

- *Caso 1*: s è una variabile, sia essa x. Siccome $x \overset{?}{>} t \in U$, t contiene solo variabili y tali che xR^+y. Allora, però, dalla definizione di σ, si ricava $x\sigma \geq_{lpo} succ(t\sigma) >_{lpo} t\sigma$.
- *Caso 2*: s non è una variabile, ma t è una variabile, sia essa x (quindi il vincolo da studiare è $s \overset{?}{>} x$). Se x occorre in s, $s\sigma >_{lpo} x\sigma$ segue

banalmente dal fatto che gli LPO sono ordini di semplificazione. Se no, x non occorre in s; per ogni t tale che $x \stackrel{?}{>} t$ occorre in U, il vincolo $s \stackrel{?}{>} t$ appartiene a U perchè U è completamente analizzato. Ma a $s \stackrel{?}{>} t$ si applica l'ipotesi induttiva perchè vale xR^+y per ogni y che occorre in t. Quindi si ha $s\sigma >_{lpo} t\sigma$; siccome s non è una variabile e s è un termine del linguaggio originario $\mathcal{L}$, vale $top(s\sigma) >_p succ$. Per (LPO2), da questo segue che $s\sigma >_{lpo} succ(t\sigma)$ per ogni t tale che $x \stackrel{?}{>} t \in U$, quindi abbiamo anche che $s\sigma >_{lpo} x\sigma$ per la definizione di σ (si noti che se non c'è nessun $x \stackrel{?}{>} t \in U$, abbiamo comunque $s\sigma >_{lpo} x\sigma \equiv 0$). $\dashv$

Teorema 3.5.5 *Il vincolo V_0 è soddisfacibile in segnature estese sse esiste un'esecuzione esaustiva e non ridondante che comincia con V_0 e che non fallisce.*

Dim. Un lato del Teorema è assicurato dal Lemma 3.5.2. Per provare l'altro lato, consideriamo un'esecuzione esaustiva (ridondante o meno, questo non importa) che non fallisce

$$(\delta) \qquad V_0, V_1, \ldots, V_i, \ldots, V_n$$

e costruiamo una soluzione per il vincolo V_0 in una segnatura estesa. Trasformiamo preventivamente la (δ) in modo da far sì che essa contenga solo applicazioni banali dell'istruzione (vi).

Si supponga che la prima applicazione non banale avvenga al passo i-esimo, sia dunque $V_i \equiv V$ e $V_{i+1} \equiv V\mu$, dove μ è upg di V_e. Si consideri la lista di vincoli

$$(\delta') \qquad V_0\mu, V_1\mu, \ldots, V_i\mu, V_{i+1}\mu, \ldots, V_n\mu.$$

Siccome si può supporre che $dom(\mu)$ sia disgiunto dalle variabili occorrenti in $cod(\mu)$,[19] le $x \in dom(\mu)$ non occorrono in $V_{i+1} \equiv V\mu$ e quindi nemmeno in $V_{i+2}, \ldots, V_n$ perchè le regole della Tabella 3.1 non introducono nuove variabili. Perciò risulta $V_{i+1}\mu \equiv V_{i+1}, \ldots, V_n\mu \equiv V_n$.

[19] Come osservato a suo tempo, l'upg calcolato dall'algoritmo della Tabella 2.1 ha questa proprietà.

Quindi le applicazioni non banali della (vi) successive alla prima restano inalterate passando da (δ) a (δ'). Le altre istruzioni della Tabella 3.1 che non producono fallimento sono stabili per sostituzioni.[20] Quindi la (δ') è ancora un'esecuzione finita dell'algoritmo della Tabella 3.1. Si tratta ancora di un'esecuzione esaustiva, perchè V_n è rimasto inalterato.[21] La (δ') non fallisce e contiene un'esecuzione non banale dell'istruzione (vi) in meno rispetto alla (δ). Iterando il procedimento, si ottiene infine un'esecuzione esaustiva che non fallisce

$$(\varepsilon) \qquad U_0, U_1, \ldots, U_n$$

tale che: (1) la (ε) contiene solo applicazioni banali dell'istruzione (vi) (quindi, $U_0 \subseteq U_1 \subseteq \cdots \subseteq U_n$); (2) vale $(\delta)\theta \equiv (\varepsilon)$ per una certa sostituzione θ.

Dalla soddisfacibilità del vincolo U_n, segue subito la soddisfacibilità del vincolo V_0, perchè $V_0\theta \equiv U_0 \subseteq U_n$. Ma la (ε) è un'esecuzione esaustiva che non fallisce e quindi U_n è completamente analizzato, sicchè esso è soddisfacibile per la Proposizione 3.5.4. $\dashv$

[20] Con questo vogliamo dire che, se V/V' è un'istanza di un'istruzione della Tabella 3.1 che non produce fallimento e diversa dalla (vi), allora $V\sigma/V'\sigma$ è un'istanza della stessa istruzione per ogni sostituzione σ.

[21] La (δ') potrebbe essere ridondante, anche se la (δ) non lo era. Questo fatto però non interessa il presente ragionamento, teso a provare che le esecuzioni esaustive (ridondanti o meno) garantiscono la soddisfacibilità del vincolo in ingresso.

3.6 Nota bibliografica

L'estensione degli ordinamenti ai multiinsiemi è dovuta a [31].

Gli ordini di semplificazione furono definiti per la prima volta in [28], dove venne anche utilizzato il Teorema di Kruskal[22] per provarne la terminazione. Gli ordinamenti RPO furono introdotti in [29] e gli ordinamenti KBO in [54]. Utilizzando la formulazione originale non ristretta del Teorema di Kruskal [57], è possibile estendere le prove di terminazione anche a ordini di semplificazione su segnature infinite [63].

C'è una discreta letteratura sul problema della soddisfacibilità (e della relativa complessità) dei vincoli per ordinamenti RPO: si vedano ad esempio [24], [51], [66], [26], [70], [64]. La nostra esposizione si è liberamente ispirata ai miglioramenti apportati agli algoritmi precedenti da [67][23] (nello stesso articolo sono riportati anche algoritmi per la soddisfacibilità attuale di vincoli RPO). Per vincoli KBO, si vedano i recenti contributi [55], [56].

[22] Nella letteratura, il Teorema di Kruskal viene usualmente formulato non per un ordine di semplificazione qualunque, ma per un particolare ordine di semplificazione indicato con $\triangleright$; tale ordine di semplificazione è definito in termini delle regole di riscrittura $f(x_1, \ldots, x_n) \to x_i$ (una per ogni simbolo di funzione n-aria f del linguaggio). L'ordine di semplificazione $\triangleright$ è contenuto in ogni altro ordine di semplificazione che opera sul linguaggio dato, per cui la nostra formulazione è una conseguenza immediata della formulazione usuale (cfr. ad esempio [4]).

[23] Da questo lavoro abbiamo tratto anche l'Esempio 3.

Capitolo 4

Refutazione e saturazione

Nel Capitolo 2 abbiamo visto come preprocessare un problema logico e ridurlo ad un problema di insoddisfacibilità di un insieme di clausole e nel Capitolo 3 abbiamo introdotto alcuni importanti strumenti di lavoro. In questo Capitolo affronteremo l'argomento centrale di questo libro, ossia come manipolare in modo efficace un insieme di clausole per scoprirne l'eventuale inconsistenza. Il nostro strumento principale sarà il 'Superposition Calculus', introdotto a più riprese con successivi miglioramenti all'inizio degli anni '90. Il 'Superposition Calculus' interviene sulla regola di Paramodulazione imponendo dei vincoli alla sua applicabilità e adottando quindi in questo modo alcune idee affermatesi nel campo dei sistemi di riscrittura (sostanzialmente, il calcolo delle coppie critiche estese diventa il principale motore inferenziale nelle derivazioni). Nelle analisi sperimentali, i dimostratori automatici che implementano il 'Superposition Calculus' si sono rivelati di gran lunga più efficaci dei precedenti basati sulla semplice regola di Paramodulazione.

Nel paragrafo 4.1 richiameremo alcuni principi generali che sottostanno ai calcoli che manipolano clausole, mentre nel paragrafo 4.3 riporteremo una delle versioni esistenti (che sono leggermente diverse fra loro) del 'Superposition Calculus', di cui nei paragrafi successivi proveremo la completezza refutazionale e studieremo alcune ottimizzazioni.

Nel prossimo Capitolo, esploreremo in modo più accurato la

relazione diretta con i sistemi di riscrittura e gli algoritmi di completamento.

4.1 Derivazioni con clausole

Ricordiamo dal Capitolo 2 che una clausola

$$(C) \qquad \{\neg A_1, \ldots, \neg A_n, B_1, \ldots, B_m\},$$

scritta, nella notazione a sequenti, con

$$A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$$

rappresenta il singolo enunciato $(C^\forall)$ che consiste nella chiusura universale della formula

$$(A_1 \wedge \cdots \wedge A_n) \supset (B_1 \vee \cdots \vee B_m).$$

Quindi la clausola (C) sarà definita vera in una struttura $\mathcal{A}$ sse l'enunciato $(C^\forall)$ è vero in $\mathcal{A}$ ed un insieme $\mathcal{T}$ di clausole sarà *soddisfacibile* (o consistente) sse esiste un modello di $\mathcal{T}$, ossia sse esiste una struttura in cui tutte le clausole di $\mathcal{T}$ sono *simultaneamente* vere.

Come si è detto, il nostro scopo principale è di introdurre calcoli logici per manipolare insiemi di clausole e scoprirne l'insoddisfacibilità. Formalmente, chiamiamo *regola di inferenza* ad n premesse ($n \geq 1$) una $n+1$-pla di clausole che scriviamo nella forma

$$\frac{C_1, \ldots, C_n}{C}$$

La regola dovrà essere *valida*, nel senso che l'enunciato

$$(C_1^\forall \wedge \cdots \wedge C_n^\forall) \supset C^\forall$$

dovrà essere logicamente valido.[1] Inoltre, supporremo tacitamente 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

[1] Nel seguito, lasceremo sempre al lettore la facile verifica del fatto che le regole che proponiamo sono tutte valide.

infinito di regole ciascuna delle quali consta di una terna di clausole del tipo indicato nello schema stesso).

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 $\mathcal{T}$ di una clausola C_k è una lista finita di clausole

$$C_1, \ldots, C_k$$

ciascuna delle quali è o un assioma di $\mathcal{T}$ (eventualmente *rinominato*),[2] 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. 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. 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.

Una volta dimostrata la completezza refutazionale, si ha a disposizione un metodo di *saturazione*: per scoprire l'eventuale inconsistenza di $\mathcal{T}$, basta chiudere l'insieme di clausole di $\mathcal{T}$ sotto le regole di inferenza fornite dal calcolo.[3] 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).

Prima di proseguire è necessario fornire alcuni aggiustamenti ai tipi di dati che trattiamo nelle derivazioni.

Innanzitutto, ci occuperemo di linguaggi del primo ordine $\mathcal{L}$ contenenti *solo l'identità* come simbolo di predicato. Questa non è una

[2] Una rinomina di una clausola C è una clausola del tipo $C\rho$, dove ρ è una sostituzione che è una rinomina (nel senso definito nel paragrafo 2.6).

[3] Come vedremo nel paragrafo 4.6, l'esecuzione delle inferenze è solo una parte del processo di saturazione vero e proprio, che deve prevedere, per essere efficace, anche dei passi di semplificazione dell'insieme delle clausole correnti.

limitazione: è infatti possibile (usando linguaggi a due sorte) scrivere le formule atomiche del tipo $P(t_1, \ldots, t_n)$ nella forma $p(t_1, \ldots, t_n) = true$, dove p è un nuovo simbolo di funzione n-ario e $true$ una nuova costante. Questa trasformazione (frequente nella letteratura specializzata) preserva la soddisfacibilità degli insiemi di clausole e, come vedremo, rende superflua la regola di Risoluzione di Robinson.[4]

[Apriamo un inciso per precisare meglio questo punto, dando i relativi (peraltro facili) dettagli. Supponiamo che il lettore sia in grado da sè di formulare la nozione di linguaggio a due sorte e di modificare di conseguenza le nozioni di struttura e di verità di un enunciato in una struttura. Nel nostro caso specifico, costruiamo da un linguaggio ad una sorta $\mathcal{L}_1$ (contenente simboli di predicato) un linguaggio a due sorte $\mathcal{L}_2$ (senza simboli di predicato) nel modo seguente. $\mathcal{L}_2$ ha, in aggiunta all'unica sorta di $\mathcal{L}_1$ che chiamiamo X, un'altra sorta che chiamiamo Y; i simboli di funzione di $\mathcal{L}_1$ sono anche simboli di funzione di $\mathcal{L}_2$, dove hanno per codominio e per dominio sempre la sorta X e, rispettivamente, le sue potenze; per ogni simbolo di predicato n-ario P di $\mathcal{L}_1$, $\mathcal{L}_2$ ha un simbolo di funzione n-ario p di dominio X^n e di codominio Y. Infine, $\mathcal{L}_2$ ha anche una costante $true$ di codominio Y. Ad ogni $\mathcal{L}_1$-formula atomica associamo una $\mathcal{L}_2$-formula atomica A^2 nel modo seguente: se $top(A)$ è l'uguaglianza, A^2 è A, altrimenti se $A \equiv P(t_1, \ldots, t_n)$, A^2 è $p(t_1, \ldots, t_n) = true$. La traduzione si estende in modo ovvio alle clausole; proviamo che, dato un insieme S di $\mathcal{L}_1$-clausole, si ha che S è soddisfacibile sse $S^2 = \{C^2 : C \in S\}$ lo è.

Da un lato, data una $\mathcal{L}_1$-struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$, si costruisca la $\mathcal{L}_2$-struttura $\mathcal{A}_2 = \langle \mathbf{A_1}, \mathbf{A_2}, \mathcal{I}_2 \rangle$ ponendo $\mathbf{A_1} := \mathbf{A}$ e $\mathbf{A_2} := \{0, 1\}$; $\mathcal{I}_2$ agisce come $\mathcal{I}$ sui simboli di funzione di sorta X e per i simboli di funzione di codominio Y, si pone $\mathcal{I}_2(p)(\underline{a}) := 1$ sse $\underline{a} \in \mathcal{I}(P)$. Poniamo anche $\mathcal{I}_2(true) := 1$. Risulta subito che $\mathcal{A} \models S$ sse $\mathcal{A}_2 \models S^2$.

Dall'altro lato, data una $\mathcal{L}_2$-struttura $\mathcal{A}_2 = \langle \mathbf{A_1}, \mathbf{A_2}, \mathcal{I}_2 \rangle$, costruiamo una $\mathcal{L}_1$-struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ come segue. Abbiamo $\mathbf{A} := \mathbf{A_1}$, inoltre $\mathcal{I}$ agisce come $\mathcal{I}_2$ sui simboli di funzione e per ogni simbolo di predicato n-ario P abbiamo $\mathcal{I}(P) := \{\underline{a} \in \mathbf{A}^n : \mathcal{I}_2(p)(\underline{a}) = \mathcal{I}_2(true)\}$. Anche in questo caso, risulta subito che $\mathcal{A} \models S$ sse $\mathcal{A}_2 \models S^2$.]

Identificheremo i *letterali positivi* del tipo $s = t$ con 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).

[4] La trasformazione viene fatta solo per motivi di compattezza ed eleganza di esposizione; nelle implementazioni, invece, Risoluzione e Fattorizzazione Destra vengono di solito mantenute come regole a parte.

Per quanto riguarda i *letterali negativi* del tipo $s \neq t$,[5] li identificheremo (per distinguerli dai corrispondenti letterali positivi) con il multiinsieme $\{s, s, t, t\}$. Così, *le clausole vengono identificate con multiinsiemi aventi per elementi multiinsiemi di termini del tipo* $\{s, t\}$ *o del tipo* $\{s, s, t, t\}$. Più esplicitamente, la clausola

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

viene identificata con il multiinsieme di multiinsiemi

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

In questo modo, potremo definire adeguatamente i vincoli di massimalità fra i letterali all'interno di una clausola.

4.2 Vincoli di massimalità in una clausola

Supponiamo fissato *un linguaggio* $\mathcal{L}$ *e un ordine di riduzione* $>$ *totale sui termini ground* di $\mathcal{L}$ (ad esempio un LPO o un KBO). Utilizzeremo *vincoli disgiuntivi*, ossia condizioni (equivalenti a) disgiunzioni di vincoli ordinari:[6] un vincolo disgiuntivo è soddisfacibile (in uno dei sensi visti nel Capitolo precedente) sse uno dei suoi disgiunti lo è. Ad esempio, la scrittura $\{s, t\} \overset{?}{>} \{u, v\}$ esprime il vincolo (disgiuntivo) che dice che il multiinsieme $\{s, t\}$ è maggiore del multiinsieme $\{u, v\}$ secondo l'estensione dell'ordine di riduzione $>$ ai multiinsiemi; in parole esplicite, la scrittura $\{s, t\} \overset{?}{>} \{u, v\}$ significa la disgiunzione delle condizioni scritte nelle tre righe seguenti:[7]

$$\begin{array}{lll}
(s \overset{?}{>} t & \& & ((s \overset{?}{>} u \;\&\; s \overset{?}{>} v) \;\vee\; (s \overset{?}{=} u \;\&\; t \overset{?}{>} v) \;\vee\; (s \overset{?}{=} v \;\&\; t \overset{?}{>} u))) \vee \\
\vee\, (t \overset{?}{>} s \;\& & ((t \overset{?}{>} u \;\&\; t \overset{?}{>} v) \;\vee\; (t \overset{?}{=} u \;\&\; s \overset{?}{>} v) \;\vee\; (t \overset{?}{=} v \;\&\; s \overset{?}{>} u))) \vee \\
\vee\, (s \overset{?}{=} t \;\& & ((s \overset{?}{>} u \;\&\; s \overset{?}{>} v) \;\vee\; (s \overset{?}{=} u \;\&\; s \overset{?}{>} v) \;\vee\; (s \overset{?}{=} v \;\&\; s \overset{?}{>} u))).
\end{array}$$

[5] Qui, ovviamente, $s \neq t$ sta per $\neg(s = t)$.

[6] Nel seguito di questo Capitolo, parlando di 'vincoli' intenderemo sempre 'vincoli disgiuntivi'.

[7] Si tenga presente, nel comprendere la presente definizione, che le soluzioni dei vincoli sono sostituzioni ground (a valori nei termini ground della segnatura corrente o di una segnatura estesa) e che i termini ground sono sempre confrontabili secondo l'ordine di riduzione che si sceglie.

Analogamente, $\{s,t\} \stackrel{?}{\geq} \{u,v\}$ vorrà dire

$$\{s,t\} \stackrel{?}{>} \{u,v\} \ \vee \ \{s,t\} \stackrel{?}{=} \{u,v\},$$

dove questo secondo disgiunto sta a sua volta per

$$(s \stackrel{?}{=} u \ \& \ t \stackrel{?}{=} v) \ \vee \ (s \stackrel{?}{=} v \ \& \ t \stackrel{?}{=} u).$$

Per esprimere vincoli di massimalità di letterali all'interno di una clausola, dobbiamo prima saper confrontare i letterali fra loro. Ciò viene fatto nel modo seguente, per casi.

Confronto di letterali positivi: dati due letterali positivi $s = t$ e $t = u$, i vincoli $s = t \stackrel{?}{>} u = v$ e $s = t \stackrel{?}{\geq} u = v$ stanno rispettivamente per $\{s,t\} \stackrel{?}{>} \{u,v\}$ e per $\{s,t\} \stackrel{?}{\geq} \{u,v\}$.

Confronto di letterali negativi: dati due letterali negativi $s \neq t$ e $t \neq u$, i vincoli $s \neq t \stackrel{?}{>} u \neq v$ e $s \neq t \stackrel{?}{\geq} u \neq v$ stanno ancora rispettivamente per $\{s,t\} \stackrel{?}{>} \{u,v\}$ e per $\{s,t\} \stackrel{?}{\geq} \{u,v\}$ (qui si vede facilmente che non fa differenza infatti confrontare ad esempio $\{s,t\}$ con $\{u,v\}$ piuttosto che confrontare $\{s,s,t,t\}$ con $\{u,u,v,v\}$).

Confronto di un letterale positivo e di un letterale negativo: siano dati un letterale negativo $s \neq t$ e un letterale positivo $u = v$. Posto che $\{s,s,t,t\} \stackrel{?}{=} \{u,v\}$ è banalmente insoddisfacibile, qualunque sostituzione ground si applichi, restano due casi da considerare. Il vincolo $s \neq t \stackrel{?}{>} u = v$ verrà espresso dalla condizione $\{s,s,t,t\} \stackrel{?}{>} \{u,v\}$, ossia da

$$(s \stackrel{?}{\geq} t \ \& \ s \stackrel{?}{\geq} u \ \& \ s \stackrel{?}{\geq} v) \quad \vee \quad (t \stackrel{?}{\geq} s \ \& \ t \stackrel{?}{\geq} u \ \& \ t \stackrel{?}{\geq} v),$$

dove ricordiamo che, ad esempio, $s \stackrel{?}{\geq} t$ sta per $(s \stackrel{?}{>} t) \vee (s \stackrel{?}{=} t)$. Invece, il vincolo $u = v \stackrel{?}{>} s \neq t$, verrà espresso dalla condizione $\{u,v\} \stackrel{?}{>} \{s,s,t,t\}$, ossia da

$$(u \stackrel{?}{\geq} v \ \& \ u \stackrel{?}{>} s \ \& \ u \stackrel{?}{>} t) \quad \vee \quad (v \stackrel{?}{\geq} u \ \& \ v \stackrel{?}{>} s \ \& \ v \stackrel{?}{>} t).$$

Se ora vogliamo esprimere il vincolo che il *letterale positivo* $u = v$ *sia massimale nella clausola*

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

basterà ricorrere a
"$u = v \overset{?}{>} s_i \neq t_i \ (\forall\, i = 1, \ldots, n)$ & $u = v \overset{?}{\geq} u_j = v_j \ (\forall\, j = 1, \ldots, m)$."
Similmente, se vogliamo esprimere il vincolo che $u = v$ sia *strettamente massimale* nella suddetta clausola, ricorreremo a
"$u = v \overset{?}{>} s_i \neq t_i \ (\forall\, i = 1, \ldots, n)$ & $u = v \overset{?}{>} u_j = v_j \ (\forall j = 1, \ldots, m)$."

Analoghe definizioni si possono dare per i letterali negativi. Se vogliamo esprimere il vincolo che il *letterale negativo* $s = t$ *sia massimale nella clausola*

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

basterà ricorrere a
"$s \neq t \overset{?}{\geq} s_i \neq t_i \ (\forall\, i = 1, \ldots, n)$ & $s \neq t \overset{?}{>} u_j = v_j \ (\forall\, j = 1, \ldots, m)$."
Similmente, se vogliamo esprimere il vincolo che $s \neq t$ sia *strettamente massimale* nella suddetta clausola, ricorreremo a
"$s \neq t \overset{?}{>} s_i \neq t_i \ (\forall\, i = 1, \ldots, n)$ & $s \neq t \overset{?}{>} u_j = v_j \ (\forall j = 1, \ldots, m)$."

4.3 Il calcolo S

Le regole di inferenza che vedremo nel presente e nel prossimo paragrafo avranno la seguente forma:

$$\frac{C_1 \quad \cdots \quad C_n}{C \ \| \ V}$$

dove V è un vincolo. Per applicare una di tali regole, *occorre preventivamente mostrare che il relativo vincolo è soddisfacibile.*[8]

Qui facciamo riferimento alla nozione di soddisfacibilità *attuale* dei vincoli (ossia alla soddisfacibilità rispetto al linguaggio $\mathcal{L}$ e all'ordine di riduzione che abbiamo fissato): con questa interpretazione della nozione di soddisfacibilità dei vincoli, *il calcolo* $\mathcal{S}$ *che introdurremo è refutazionalmente completo*, come vedremo nel paragrafo 4.5.

[8] Detto altrimenti, sono istanze di una tale regola solo le $n + 1$-ple di clausole $(C_1, \ldots, C_n, C)$ tali che V è soddisfacibile.

Va tuttavia osservato, però, che è possibile (e anzi necessario) usare nelle implementazioni delle nozioni blande di soddisfacibilità che siano computazionalmente trattabili. Da un punto di vista matematico, queste scelte sono pienamente giustificate: ogni nozione di soddisfacibilità *che sia implicata dalla soddisfacibilità attuale* non fa perdere la completezza refutazionale ed è quindi accettabile[9] (l'unico problema che può insorgere con queste scelte blande è la derivazione di clausole non strettamente indispensabili a scoprire le inconsistenze).

Ad esempio, la nozione di soddisfacibilità blanda comunemente utilizzata è la seguente. Siccome i vincoli che compaiono nelle regole che vedremo sono tutti del tipo $s_1 \stackrel{?}{=} s_2$ & V', dove V' consta di congiunzioni di espressioni del tipo

$$u \stackrel{?}{>} v, \qquad u = u' \stackrel{?}{>} v = v',$$
$$u = u' \stackrel{?}{\geq} v = v', \qquad u \neq u' \stackrel{?}{>} v = v',$$
$$u = u' \stackrel{?}{>} v \neq v',$$

tali espressioni sono considerate soddisfatte come vincoli, qualora si abbia rispettivamente

$$u\mu \not\leq v\mu, \qquad \{u\mu, u'\mu\} \not\leq \{v\mu, v'\mu\},$$
$$\{u\mu, u'\mu\} \not< \{v\mu, v'\mu\}, \qquad \{u\mu, u\mu, u'\mu, u'\mu\} \not\leq \{v\mu, v'\mu\},$$
$$\{u\mu, u'\mu\} \not\leq \{v\mu, v\mu, v'\mu, v'\mu\},$$

dove μ è upg del problema di unificazione $s_1 \stackrel{?}{=} s_2$. Qui, ovviamente, le condizioni $\{u\mu, u'\mu\} \not\leq \{v\mu, v'\mu\}$, $\{u\mu, u'\mu\} \not< \{v\mu, v'\mu\}$, ecc. fanno riferimento all'estensione ai multiinsiemi di termini dell'ordine di riduzione sui termini del linguaggio. Si tenga presente che, trattandosi di termini non ground, non è detto che due termini (e quindi nemmeno due multiinsiemi di termini) siano confrontabili fra loro: per questo non si possono sostituire $\not\leq$ con $>$ e $\not<$ con $\geq$ (cosa che invece si può fare per multiinsiemi di termini ground).

Le regole del calcolo $\mathcal{S}$ sono specificate (con i relativi vincoli) nella Tabella 4.1. L'uso dei vincoli realizza una serie di intuizioni su come re-

[9] Al limite, si potrebbero addirittura ignorare i vincoli (considerandoli cioè sempre soddisfacibili): le regole del calcolo sarebbero comunque valide.

Tabella 4.1: Regole di Inferenza di $\mathcal{S}$

Sovrapposizione Destra

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

(se $s_{|p}$ non è una variabile)

Sovrapposizione Sinistra

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

(se $s_{|p}$ non è una variabile)

Risoluzione per l'Uguaglianza

$$\frac{s = s', \Gamma \Rightarrow \Delta}{\Gamma\mu \Rightarrow \Delta\mu \;\parallel\; VE1}$$

Fattorizzazione per l'Uguaglianza

$$\frac{\Gamma \Rightarrow s = t, s' = t', \Delta}{\Gamma\mu, (t = t')\mu \Rightarrow (s = t')\mu, \Delta\mu \;\parallel\; VE2}$$

Nelle prime due regole, μ è supposto essere upg di $s_{|p} \stackrel{?}{=} l$;
nelle seconde due regole, μ è supposto essere upg di $s \stackrel{?}{=} s'$.
I vincoli $VD, VS, VE1, VE2$ sono così specificati:

VD $\quad s_{|p} \stackrel{?}{=} l$ & $l \stackrel{?}{>} r$ & $s \stackrel{?}{>} t$ & $s = t \stackrel{?}{>} l = r$ &
$l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta'$ &
$s = t$ è strettamente massimale in $\Gamma \Rightarrow s = t, \Delta$.

VS $\quad s_{|p} \stackrel{?}{=} l$ & $l \stackrel{?}{>} r$ & $s \stackrel{?}{>} t$ &
$l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta'$ &
$s \neq t$ è massimale in $\Gamma, s = t \Rightarrow \Delta$.

$VE1$ $\quad s \stackrel{?}{=} s'$ & $s \neq s'$ è massimale in $\Gamma, s = s' \Rightarrow \Delta$.

$VE2$ $\quad s \stackrel{?}{=} s'$ & $s \stackrel{?}{>} t$ &
$s = t$ è massimale in $\Gamma \Rightarrow s = t, s' = t', \Delta$.

stringere l'applicazione delle regole. Queste intuizioni consistono nell'utilizzare per le inferenze *solo termini massimali di letterali massimali*,[10] a preferenza dei rimanenti. Inoltre, nelle Regole di Sovrapposizione, l'identità è trattata in modo *non simmetrico*: il parallelo con i sistemi di riscrittura ordinata è immediato, perchè l'equazione $s[r]_p = t$ che compare nel conseguente delle due Regole di Sovrapposizione altri non è che una *coppia critica estesa*[11] delle equazioni $s = t$ e $l = r$ (orientate con $s \to t$ e $l \to r$, mediante la sostituzione ground che risolve il vincolo).

La Regola di Fattorizzazione per l'Uguaglianza svolge un ruolo analogo alla regola di Fattorizzazione Destra impiegata nei calcoli basati sulla Risoluzione ed è necessaria per ottenere la completezza refutazionale se si trattano clausole che contengono più di un letterale positivo.

La presenza della Regola di Fattorizzazione per l'Uguaglianza (che opera solo su letterali positivi) giustifica il fatto che nelle Regole di Sovrapposizione si richiedono per i letterali positivi vincoli di massimalità stretta, mentre per i letterali negativi solo vincoli di massimalità semplice.

[I vincoli che intervengono nelle regole del Calcolo $\mathcal{S}$ si possono semplificare ed esplicitare svolgendo le definizioni. Ad esempio, la definizione che abbiamo dato nel paragrafo 4.2 della relazione $\{s,t\} \overset{?}{>} \{u,v\}$ consiste in una disgiunzione di tre casi; siccome, quando tale definizione è usata per confrontare due letterali positivi o due letterali negativi fra loro nel vincolo di applicazione di una regola, è presente nel vincolo stesso anche la condizione $s \overset{?}{>} t$, è chiaro che solo il primo caso dà una condizione soddisfacibile ed è pertanto l'unico caso che va tenuto in considerazione. Analoghe (e ancor maggiori) semplificazioni si possono avere nel confrontare un letterale positivo e uno negativo: ad esempio, in presenza di $s \overset{?}{>} t$, la condizione $s \neq t \overset{?}{>} u = v$ si riduce a $s \overset{?}{\geq} u \;\&\; s \overset{?}{\geq} v$ e la condizione $s = t \overset{?}{>} u \neq v$ si riduce a $s \overset{?}{>} u \;\&\; s \overset{?}{>} v$. Quindi, anche trascurando di trarre vantaggio dall'informazione $s \overset{?}{>} t$ nel confronto di due letterali positivi/negativi fra loro,

[10] Spesso nei dimostratori automatici i letterali massimali delle clausole memorizzate vengono marcati con un asterisco (o con un doppio asterisco se il primo dei due termini della relativa equazione è maggiore del secondo). Questo metodo di mantenere i dati è efficace, tuttavia va osservato che la massimalità dovrebbe essere ri-testata dopo l'applicazione dell'upg richiesto dalle regole di inferenza (l'applicazione di tale upg potrebbe, infatti, far cadere *a posteriori* la massimalità che valeva).

[11] Si veda il paragrafo 5.2 per la relativa definizione.

è chiaro che i vincoli $VD, VS, VE1, VE2$ possono tutti essere espressi nella forma $s_1 \stackrel{?}{=} s_2 \,\&\, V'$ dove V' consta di congiunzioni di vincoli del tipo

$$u \stackrel{?}{>} v, \quad u \stackrel{?}{\geq} v, \quad \{u, u'\} \stackrel{?}{>} \{v, v'\}, \quad \{u, u'\} \stackrel{?}{\geq} \{v, v'\}.$$

Approfitteremo di queste semplificazioni nel trattare gli esempi del prossimo paragrafo.]

4.4 Esempi

Vediamo ora l'utilizzo concreto del calcolo $\mathcal{S}$ e sviluppiamo contemporaneamente anche importanti complementi alla nostra trattazione. Per valutare appieno l'efficacia del Calcolo $\mathcal{S}$, si faccia attenzione, negli esempi che verranno presentati, non solo alla comprensione delle derivazioni proposte, ma anche al relativamente ristretto numero di inferenze possibili.

Le regole di Sovrapposizione Destra, di Sovrapposizione Sinistra, di Risoluzione per l'Uguaglianza e di Fattorizzazione per l'Uguaglianza saranno contrassegnate, rispettivamente, con le sigle SpR, SpL, EqR ed EqF; una dicitura del tipo $[SpL\ 1.1; 3.1]$ significherà che, per ottenere la clausola corrente, è stata applicata la regola di Sovrapposizione Sinistra al primo letterale della clausola 1 e al primo letterale della clausola 3.

ESEMPIO 1. Vogliamo dimostrare che in un semigruppo il prodotto di due elementi a, b cancellativi a sinistra è cancellativo a sinistra. Le ipotesi di questo problema sono le seguenti (ricordiamo che un semigruppo è un insieme dotato di una operazione binaria associativa '$\cdot$' che scriviamo in notazione infissa):

$$\forall x \forall y \forall z\ ((x \cdot y) \cdot z = x \cdot (y \cdot z))$$
$$\forall x \forall y\ (x \cdot a = y \cdot a \supset x = y)$$
$$\forall x \forall y\ (x \cdot b = y \cdot b \supset x = y)$$

La tesi è la seguente:

$$\forall x \forall y\ (x \cdot (a \cdot b) = y \cdot (a \cdot b) \supset x = y).$$

Skolemizzando otteniamo le cinque clausole:

1. $\Rightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z)$
2. $x \cdot a = y \cdot a \Rightarrow x = y$
3. $x \cdot b = y \cdot b \Rightarrow x = y$
4. $\Rightarrow s_1 \cdot (a \cdot b) = s_2 \cdot (a \cdot b)$
5. $s_1 = s_2 \Rightarrow$

dove s_1, s_2 sono nuove costanti. Per dedurre nel calcolo $\mathcal{S}$ la clausola vuota, usiamo l'ordinamento $>_{lpo}$ indotto dalla relazione di precedenza

$$\cdot >_p b >_p a >_p s_1 >_p s_2.$$

Una prima applicazione della Regola di Sovrapposizione Sinistra genera la clausola:

$$6.\ \ x \cdot b = v \cdot (w \cdot b) \Rightarrow x = v \cdot w \quad [SpL\ 1.1; 3.1]$$

Il sistema ha generato il vincolo

$$y \cdot b \stackrel{?}{=} (v \cdot w) \cdot z\ \&\ (v \cdot w) \cdot z \stackrel{?}{>} v \cdot (w \cdot z)\ \&\ y \cdot b \stackrel{?}{>} x \cdot b\ \&\ y \cdot b \stackrel{?}{\geq} x\ \&\ y \cdot b \stackrel{?}{\geq} y.$$

Il vincolo è soddisfacibile e la sua parte equazionale ha come upg la sostituzione

$$y \mapsto v \cdot w, \qquad z \mapsto b.$$

Questo upg viene applicato ai membri destri delle due equazioni (si noti che la prima è stata rinominata)

$$(v \cdot w) \cdot z = v \cdot (w \cdot z), \qquad y \cdot b = x \cdot b$$

generando la nuova equazione (coppia critica estesa delle due precedenti, secondo la terminologia dei sistemi di riscrittura)

$$v \cdot (w \cdot b) = x \cdot b.$$

L'upg viene poi applicato ai letterali rimanenti delle due clausole 1 e 3 coinvolte nella Sovrapposizione Sinistra (di fatto, c'è un solo letterale, ossia il letterale $x = y$ della 3), dando come risultato la clausola 6.

Per proseguire, usiamo ancora la regola di Sovrapposizione Sinistra:

$$7.\ \ x \cdot b = s_2 \cdot (a \cdot b) \Rightarrow x = s_1 \cdot a \quad [SpL\ 4.1; 6.1]$$

Qui abbiamo il vincolo soddisfacibile

$$s_1 \cdot (a \cdot b) \stackrel{?}{=} v \cdot (w \cdot b)\ \&\ s_1 \cdot (a \cdot b) \stackrel{?}{>} s_2 \cdot (a \cdot b)\ \&$$

$$\&\ v \cdot (w \cdot b) \stackrel{?}{>} x \cdot b\ \&\ v \cdot (w \cdot b) \stackrel{?}{\geq} x\ \&\ v \cdot (w \cdot b) \stackrel{?}{\geq} v \cdot w.$$

L'upg è la sostituzione $v \mapsto s_1$, $w \mapsto a$. La sovrapposizione in radice delle due equazioni $s_1 \cdot (a \cdot b) = s_2 \cdot (a \cdot b)$ e $v \cdot (w \cdot b) = x \cdot b$ genera la nuova equazione $s_2 \cdot (a \cdot b) = x \cdot b$ e quindi la clausola 7 applicando l'upg al letterale rimanente $x = v \cdot w$.

Per proseguire, si usa ancora la Regola di Sovrapposizione Sinistra:

$$8.\ \ u \cdot (v \cdot b) = s_2 \cdot (a \cdot b) \Rightarrow u \cdot v = s_1 \cdot a \quad [SpL\ 1.1; 7.1]$$

È stato generato il vincolo soddisfacibile

$$(u \cdot v) \cdot w \stackrel{?}{=} x \cdot b\ \&\ (u \cdot v) \cdot w \stackrel{?}{>} u \cdot (v \cdot w)\ \&\ x \cdot b \stackrel{?}{>} s_2 \cdot (a \cdot b)\ \&$$
$$\&\ x \cdot b \stackrel{?}{\geq} x\ \&\ x \cdot b \stackrel{?}{\geq} s_1 \cdot a.$$

L'upg è la sostituzione $x \mapsto u \cdot v$, $w \mapsto b$. La sovrapposizione in radice delle due equazioni $(u \cdot v) \cdot w = u \cdot (v \cdot w)$ e $x \cdot b = s_2 \cdot (a \cdot b)$ genera la nuova equazione $u \cdot (v \cdot b) = s_2 \cdot (a \cdot b)$ e quindi la clausola 8 applicando l'upg al letterale rimanente $x = s_1 \cdot a$.

Applichiamo ora la Regola di Risoluzione per l'Uguaglianza alla clausola 8, ottenendo

$$9.\ \ \Rightarrow s_2 \cdot a = s_1 \cdot a \quad [EqR\ 8.1]$$

Per applicare la regola è stato generato il vincolo soddisfacibile

$$u \cdot (v \cdot b) \stackrel{?}{=} s_2 \cdot (a \cdot b)\ \&\ s_2 \cdot (a \cdot b) \stackrel{?}{\geq} u \cdot v\ \&\ s_2 \cdot (a \cdot b) \stackrel{?}{\geq} s_1 \cdot a,$$

è stato trovato l'upg $u \mapsto s_2$, $v \mapsto a$, il quale è stato applicato al letterale $u \cdot v = s_1 \cdot a$ per produrre la clausola 9.

Una nuova applicazione della Regola di Sovrapposizione Sinistra produce la clausola

$$10.\ \ x \cdot a = s_2 \cdot a \Rightarrow x = s_1 \quad [SpL\ 9.1; 2.1]$$

Per applicare la regola, è stato generato il vincolo soddisfacibile

$$y \cdot a \stackrel{?}{=} s_1 \cdot a\ \&\ s_1 \cdot a \stackrel{?}{>} s_2 \cdot a\ \&\ y \cdot a \stackrel{?}{>} x \cdot a\ \&\ y \cdot a \stackrel{?}{\geq} x\ \&\ y \cdot a \stackrel{?}{\geq} y.$$

con upg $y \mapsto s_1$. La sovrapposizione in radice delle due equazioni $s_1 \cdot a = s_2 \cdot a$ e $y \cdot a = x \cdot a$ genera la nuova equazione $s_2 \cdot a = x \cdot a$ e quindi la clausola 10 applicando l'upg al letterale rimanente $x = y$.

Un'applicazione della Regola di Risoluzione per l'Uguaglianza produce la clausola

$$11.\ \ \Rightarrow s_2 = s_1 \quad [EqR\ 10.1]$$

Qui è stato generato il vincolo soddisfacibile

$$x \cdot a \stackrel{?}{=} s_2 \cdot a\ \&\ s_2 \cdot a \stackrel{?}{\geq} x\ \&\ s_2 \cdot a \stackrel{?}{\geq} s_1,$$

è stato trovato l'upg $x \mapsto s_2$, il quale è stato applicato al letterale $x = s_1$ per produrre la clausola 11.

La dimostrazione si conclude generando la clausola $s_2 = s_2 \Rightarrow$ e poi la clausola vuota dalle clausole 11 e 5, mediante Sovrapposizione Sinistra e Risoluzione per l'Uguaglianza.

Come si è detto, le regole del calcolo basato sulla Risoluzione di Robinson sono automaticamente conglobate nel calcolo $\mathcal{S}$, 'traducendo' le formule atomiche del tipo $P(t_1, \ldots, t_n)$ in equazioni del tipo $p(t_1, \ldots, t_n) = true$ (per ragioni di efficienza, la costante *true* deve essere resa piccola nell'ordinamento dei termini). In dettaglio, le regole di Sovrapposizione Sinistra e di Fattorizzazione per l'Uguaglianza, se applicate a letterali aventi in radice simboli di predicato, danno rispettivamente le Regole di *Risoluzione Ordinata* e di *Fattorizzazione Destra Ordinata*.[12] Tali regole costituiscono il calcolo $\mathcal{R}$ della Tabella 4.2: da quanto si è detto e dalla completezza refutazionale di $\mathcal{S}$ (Teorema 4.5.1), segue immediatamente che il calcolo $\mathcal{R}$ è *refutazionalmente completo per i linguaggi senza identità*.[13]

Per la comprensione del significato dei vincoli nella Tabella 4.2, si tenga sempre presente che un letterale positivo del tipo $P(t_1, \ldots, t_n)$ vale come il multiinsieme $\{p(t_1, \ldots, t_n), true\}$, ossia di fatto (siccome si fa sì che *true* sia minimo nell'ordinamento dei termini) come il multiinsieme $\{p(t_1, \ldots, t_n)\}$. Analogamente, un letterale negativo del tipo $\neg P(t_1, \ldots, t_n)$ vale come il multiinsieme $\{p(t_1, \ldots, t_n), p(t_1, \ldots, t_n)\}$.

ESEMPIO 2. Riprendiamo in esame le tre clausole dell'Esempio 1 del paragrafo 2.5:

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

[12] Strettamente parlando, occorrerebbe inserire nel conseguente di entrambe le regole un letterale negativo del tipo $true \neq true$: tale letterale tuttavia potrà essere rimosso in seguito dalla Regola di Risoluzione per l'Uguaglianza (quando il relativo vincolo sarà soddisfacibile), oppure potrà essere rimosso all'istante mediante una banale regola di riduzione del tipo di quelle che vedremo nel paragrafo 4.6.

[13] Segnaliamo che la Regola di Fattorizzazione Sinistra

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

(dove μ è upg di $A \stackrel{?}{=} B$), viene talvolta implementata (o meglio data per opzionale) nei dimostratori automatici correnti, anche se essa non sarebbe strettamente necessaria per ottenere la completezza refutazionale del calcolo.

Otteniamo la seguente prova della clausola vuota in due passaggi:

$$\begin{array}{ll} 4.\ R(x, s(x)) \Rightarrow & [Res\ 1.2; 3.1] \\ 5.\ \Rightarrow & [Res\ 2.1; 4.1] \end{array}$$

dove la regola di Risoluzione è stata contrassegnata con la sigla Res.

Tabella 4.2: Regole di Inferenza di $\mathcal{R}$

Risoluzione Ordinata

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

Fattorizzazione Destra Ordinata

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

dove μ è upg di $A \stackrel{?}{=} B$ e i vincoli VR, VF sono così specificati:

VR $\quad A \stackrel{?}{=} B$ & $\neg B$ è massimale in $B, \Gamma' \Rightarrow \Delta'$ &
A è strettamente massimale in $\Gamma \Rightarrow A, \Delta$.

VF $\quad A \stackrel{?}{=} B$ & A è massimale in $\Gamma \Rightarrow \Delta, A, B$.

ESEMPIO 3. Il seguente esempio dimostra che la regola di Fattorizzazione Destra (contrassegnata con la sigla Fac) 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. $\Rightarrow R(x, f(x))$ $[Fac\ 1.1; 1.2]$
4. $R(x, f(x)) \Rightarrow$ $[Res\ 3.1; 2.1]$
5. $\Rightarrow$ $[Res\ 3.1; 4.1]$

ESEMPIO 4. (Cavalieri e Furfanti) Risolviamo il seguente quesito 'alla Smullyan':[14]

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. 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

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

[14] L'universo di Herbrand (ossia l'insieme dei termini ground del linguaggio considerato) in questo e nel successivo esempio è finito. In tale situazione, metodi di analisi basati su tecniche di logica proposizionale (come la procedura DPLL) possono risultare più efficaci.

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

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

Analogamente, da (II) otteniamo la formula

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

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

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

Non ci resta che provare ora a vedere cosa succede aggiungendo agli enunciati che abbiamo descritto (in tre esperimenti diversi) le 'congetture' $\neg L(a)$, $\neg L(b)$, $\neg L(c)$. Nel primo caso il dimostratore automatico in cui abbiamo inserito i dati segnala 'proof found', il che vuol dire che certamente a non è il lupo. Negli altri due casi si ha il messaggio 'completion found', il che significa che il processo di saturazione termina senza produrre la clausola vuota: quindi non c'è alcuna garanzia che il lupo non sia proprio b o, rispettivamente, c.

ESEMPIO 5. 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 sta per 'Italia', s per 'Svizzera', a per 'Austria', ecc.). Inoltre ci servono tre 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 $\forall x \forall y(C(x,y) \supset C(y,x))$; 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

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

Se i dati sono consistenti (ossia se il dimostratore automatico in cui li abbiamo inseriti dà il messaggio 'completion found') la colorazione

esiste, altrimenti (ossia se dà il messaggio 'proof found') la colorazione non esiste.

Nel nostro calcolo, sia la regola di Risoluzione che la regola di Fattorizzazione Destra risultano essere soggette a vincoli (quelli indicati nella Tabella 4.2). Questi vincoli fanno sì che solo i letterali che soddisfano il vincolo di massimalità siano utilizzabili per queste inferenze. Ciò porta a considerevoli risparmi nel processo di saturazione e aumenta i casi di terminazione.

ESEMPIO 6. Consideriamo a questo proposito l'esempio dell'insieme di clausole

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

Siccome $P(f(x)) > P(x)$, non risulta applicabile nessuna inferenza: questo insieme di clausole è già saturo. Se non si utilizzano vincoli, invece, il processo di saturazione diverge.

Il meccanismo dei vincoli, nel caso in cui il linguaggio contenga simboli di predicato diversi dall'identità, può trarre qualche vantaggio (senza costi aggiuntivi di complessità) qualora si faccia un uso moderato della soddisfacibilità attuale dei vincoli stessi. Supponiamo ad esempio che il linguaggio contenga un simbolo di predicato n-ario P e usiamo un ordinamento LPO basato su una precedenza che dia a P precedenza maggiore rispetto a tutti gli altri simboli di funzione e di predicato. È chiaro allora che un letterale L del tipo $P(t_1, \ldots, t_n)$ o del tipo $\neg P(t_1, \ldots, t_n)$ soddisfa sempre (in senso attuale) il vincolo

$$L \stackrel{?}{>} L'$$

rispetto ad un letterale *qualunque* L' in cui P non occorre. Questo fatto ovviamente restringe in modo significativo le inferenze applicabili alle clausole C in cui il predicato P occorre (tali clausole saranno utilizzabili ad esempio solo per regole di Risoluzione e Fattorizzazione Destra coinvolgenti letterali del tipo $P(t_1, \ldots, t_n)$ o $\neg P(t_1, \ldots, t_n)$).

ESEMPIO 7. Diamo un esempio che richiede l'uso della Regola di Fattorizzazione per l'Uguaglianza. Consideriamo le clausole

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

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.	$x = x, x = x \Rightarrow$	$[Res\ 10.2; 11.1]$
13.	$x = x \Rightarrow$	$[EqR\ 12.1]$
14.	$\Rightarrow$	$[EqR\ 13.1]$

ESEMPIO 8. 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

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

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:

$$C(x,y) \quad \Leftrightarrow \quad (F(p(x),p(y)) \vee F(p(x),m(y)) \vee \\ \vee F(m(x),p(y)) \vee F(m(x),m(y))).$$

Tabella 4.3: Regole di Inferenza Aggiuntive di $\mathcal{S}^+$

Sovrapposizione Sinistra (con Selezione)

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

(se $s_{|p}$ non è una variabile)

Risoluzione per l'Uguaglianza (con Selezione)

$$\frac{(s = s')^+, \Gamma \Rightarrow \Delta}{\Gamma\mu \Rightarrow \Delta\mu \;\; \| \;\; VE1^+}$$

Nella prima regola, μ è upg di $s_{|p} \stackrel{?}{=} l$ e nella seconda, μ è upg di $s \stackrel{?}{=} s'$; i vincoli VS^+ e $VE1^+$ sono così specificati:

VS^+ $\quad$ $s_{|p} \stackrel{?}{=} l$ & $l \stackrel{?}{>} r$ & $s \stackrel{?}{>} t$ & $l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta'$.

$VE1^+$ $\quad$ $s \stackrel{?}{=} s'$.

In certi casi (come in 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*. Supponiamo di avere a disposizione un criterio 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 indicate nella Tabella 4.3 (in tale Tabella il letterale negativo marcato è contrassegnato con $(-)^+$).

Si osservi che nella premessa sinistra $\Gamma' \Rightarrow \Delta', l = r$ della regola di Sovrapposizione Sinistra (con Selezione), 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').

Proveremo nel prossimo paragrafo che 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 che consiste 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).

Per ottenere ulteriori problemi da sottoporre al calcolatore, si possono sfogliare testi ed eserciziari di algebra elementare: i dimostratori automatici odierni che implementano il calcolo $\mathcal{S}$ sono infatti in grado di provare facilmente ad esempio che un gruppo di periodo due è abeliano, che un anello booleano è commutativo ed ha caratteristica due, che il complemento è unico nelle algebre di Boole, ecc. In rete, sono anche disponibili ampi campionari di esempi significativi (come la libreria TPTP).

4.5 Completezza refutazionale

Il presente paragrafo è interamente dedicato alla dimostrazione del seguente

Teorema 4.5.1 *(di Completezza Refutazionale) Il calcolo $\mathcal{S}$ è refutazionalmente completo.*

Per affrontare la dimostrazione, anticipiamo alcune nozioni sui sistemi di riscrittura che vedremo meglio nel paragrafo 5.1.

Una *regola di riscrittura ground* è una coppia di termini ground scritti nella forma

$$l \to r.$$

Se R è un insieme di regole di riscrittura ground e se t, u sono termini ground scriviamo $t \to_R^p u$ per esprimere il fatto che esite una regola $l \to r$ appartenente a R tale che

$$t_{|p} \equiv l \quad \text{e} \quad u \equiv t[r]_p$$

(spesso ometteremo l'indicazione esplicita di p e/o di R in $t \to_R^p u$ qualora siano chiare dal contesto). Risulta chiaro che l'insieme dei termini ground costituisce un sistema di riscrittura astratto (nel senso del paragrafo 2.4) rispetto alla relazione $\to_R$.

Se esiste un ordine di riduzione $>$ tale che vale $l > r$ per ogni $l \to r \in R$, tale sistema di riscrittura astratto è certamente terminante (perchè l'applicazione iterata della proprietà di compatibilità degli ordini di riduzione assicura che $t \to_R u$ implica $t > u$). In tale situazione, la convergenza dovrà essere testata tramite la convergenza locale (per il Lemma di Newman 2.4.12); il seguente Lemma (che è un caso molto particolare di un risultato importante che vedremo nel prossimo Capitolo) dà un semplice criterio in proposito. Se R è un sistema di riscrittura ground, una *coppia critica* di R è una coppia di termini ground del tipo $(r, l[r']_p)$ per cui $l \to r$ e $l_{|p} \to r'$ sono entrambe regole (diverse fra loro) di R.

Lemma 4.5.2 *Se il sistema di riscrittura ground R è terminante e non ha coppie critiche, R è convergente.*

Dim. Si supponga che valga $t \to_R^{p_1} s_1$ e che valga $t \to_R^{p_2} s_2$ per certi termini ground t, s_1, s_2. Questo significa che esistono due regole $l_i \to r_i \in R$ $(i = 1, 2)$ tali che $t_{|p_i} \equiv l_i$ e che $s_i \equiv t[r_i]_{p_i}$. Ci sono due casi, a seconda che p_1, p_2 siano posizioni parallele o meno.

Se p_1, p_2 non sono parallele, abbiamo ad esempio $p_2 = p_1 q$ per una certa q. Allora $l_2 \equiv t_{|p_2} \equiv (t_{|p_1})_{|q} \equiv (l_1)_{|q}$ (per il Lemma 2.3.2); siccome R non ha coppie critiche, segue $l_1 \equiv l_2, r_1 \equiv r_2$ (e anche $q = \epsilon$ perchè nessun termine è uguale ad un suo sottotermine proprio), da cui $s_1 \equiv s_2$.

Se p_1, p_2 sono posizioni parallele, per il Lemma 2.3.2 si ha:

$$t[r_1]_{p_1} \to (t[r_1]_{p_1})[r_2]_{p_2} \equiv (t[r_2]_{p_2})[r_1]_{p_1} \leftarrow t[r_2]_{p_2},$$

il che dimostra la confluenza di s_1 e s_2. $\dashv$

Riprendiamo la dimostrazione del Teorema di Completezza Refutazionale. Anticipiamo alcune idee relativamente alla tecnica (detta di 'model generation') che verrà impiegata. Il Teorema di Completezza Refutazionale dice che, dato un insieme $\mathcal{T}$ di clausole, $\mathcal{T}$ è inconsistente sse esiste una derivazione in $\mathcal{S}$ da $\mathcal{T}$ della clausola vuota. Un lato di tale teorema è banale e segue dal fatto che le regole di $\mathcal{S}$ sono tutte valide. Per l'altro lato, supponiamo che non esista una derivazione in $\mathcal{S}$ da $\mathcal{T}$ della clausola vuota. Sia S l'insieme delle clausole C per cui esiste una derivazione in $\mathcal{S}$ da $\mathcal{T}$ di C. Siccome $S \supseteq \mathcal{T}$, basterà trovare un modello di S. S non contiene la clausola vuota ed è *chiuso rispetto alle regole di* $\mathcal{S}$, nel senso che S contiene la conclusione di ogni regola di inferenza di $\mathcal{S}$ le cui premesse (eventualmente rinominate) appartengono a S. Quindi, per dimostrare il Teorema di Completezza Refutazionale, basta provare che *ogni insieme di clausole S non contenente la clausola vuota e chiuso rispetto alle regole di $\mathcal{S}$ ammette un modello.*

Come primo passo in tale direzione, introduciamo un ordinamento per le clausole *ground*, che ci sarà utile. Ricordiamo che una clausola ground C

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

rappresenta il multiinsieme di multiinsiemi (cfr. il paragrafo 4.1)

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

quindi le clausole ground sono ordinate dalla relazione d'ordine stretto (che chiamiamo ancora $>$ per semplicità) ottenuta considerando la doppia estensione ai multiinsiemi dell'ordine di riduzione $>$ che opera sui termini.

Essendo questo ordinamento sulle clausole ground totale e terminante, lo utilizzeremo per introdurre un sistema di riscrittura ground R, facendo un'induzione transfinita sulle clausole ground che sono istanze di clausole in S. R risulterà essere convergente e pertanto la R-congiungibilità definirà una relazione di equivalenza sull'universo di Herbrand dei termini ground di $\mathcal{L}$. Il modello di S cercato sarà allora ottenuto introducendo sull'insieme quoziente l'ovvia interpretazione di Herbrand $\mathcal{I}$ ($\mathcal{I}$ sarà fatta in modo da interpretare ogni termine ground sulla sua stessa classe di equivalenza).

Vediamo ora la realizzazione nei dettagli di questo programma. Fissiamo, come detto, un insieme di clausole S che sia chiuso rispetto alle

regole del calcolo $\mathcal{S}$ e che non contenga la clausola vuota. Sia $g(S)$ l'insieme delle clausole che sono istanze ground di clausole in S. Per ogni $C \in g(S)$, definiamo per induzione transfinita su C un sistema di riscrittura ground $R_{\leq C}$.

Useremo la seguente notazione: se R è un sistema di riscrittura ground e, se C è la clausola ground

$$t_1 = u_1, \ldots, t_n = u_n \Rightarrow s_1 = v_1, \ldots, s_m = v_m,$$

la relazione $R \models C$ significherà[15]

"se $(t_1 \leftrightarrow_R^* u_1 \,\&\, \cdots \,\&\, t_n \leftrightarrow_R^* u_n)$ allora $(s_1 \leftrightarrow_R^* v_1 \text{ o } \cdots \text{ o } s_m \leftrightarrow_R^* v_m)$"

(se $n = 0$ e $m = 1$, scriveremo direttamente $R \models s_1 = v_1$ piuttosto che $R \models \Rightarrow s_1 = v_1$ o che $s_1 \leftrightarrow_R^* v_1$).

Supponiamo che $R_{\leq D}$ sia già stato definito per ogni $D \in g(S)$ con $D < C$ e sia $R_{<C} = \bigcup\{R_{\leq D} : D \in g(S), D < C\}$; abbiamo che $R_{\leq C}$ coincide con $R_{<C}$ a meno che non si verifichino le seguenti cinque condizioni, nel qual caso si avrà che $R_{\leq C} := R_{<C} \cup \{l \to r\}$:

(0) C è del tipo $\Gamma \Rightarrow l = r, \Delta$;

(1) $R_{<C} \not\models C$;

(2) l è in forma normale in $R_{<C}$;

(3) $l > r$ e $l > t$ (per ogni t che occorre in Γ) e $\{l, r\} > \{u, v\}$ (per ogni letterale positivo $u = v$ che compare in Δ);[16]

(4) $R_{<C} \not\models r = v$ per ogni letterale positivo del tipo $l = v$ che compare in Δ.

Nel caso che le condizioni (0)-(4) siano verificate si dice che la clausola C è *produttiva* e che produce la regola di riscrittura $l \to r$.

Sia $R := \bigcup\{R_{\leq C} : C \in g(S)\}$.

[15] Ricordiamo dal paragrafo 2.4 che la relazione $\leftrightarrow_R^*$ è la minima relazione di equivalenza che contiene $\to_R$. Ricordiamo anche che, nell'ipotesi di convergenza di R, la relazione $\leftrightarrow_R^*$ si testa mediante il confronto delle forme normali (Proposizione 2.4.10).

[16] Quest'ultima condizione individua univocamente il letterale positivo $l = r$ nella clausola C. Si osservi anche che, assumendo $l > r$, le condizioni

"$l > t$ (per ogni t che occorre in Γ) e $\{l, r\} > \{u, v\}$ (per ogni $u = v \in \Delta$)"

semplicemente affermano che il letterale positivo $l = r$ è strettamente massimale nella clausola ground $\Gamma \Rightarrow l = r, \Delta$ (cfr. quanto detto in chiusura del paragrafo 4.3).

Lemma 4.5.3 (i) *I sistemi di riscrittura ground R e $R_{<C}$ sono tutti convergenti;*

(ii) *per ogni $C \in g(S)$, se $R_{<C} \models C$, allora $R \models C$.*

Dim. (i) I sistemi R e $R_{<C}$ sono terminanti per la condizione (3). Quindi basta provare che essi soddisfano la condizione del Lemma 4.5.2, cioè che, se $l \to r$ è una regola di R, non esiste nessuna regola $l' \to r'$ di R (diversa dalla $l \to r$ stessa) con l' che è sottotermine di l.[17] Quando $l \to r$ è prodotta da una clausola C, l è irriducibile dalle regole generate in precedenza per la condizione (2). Quindi $l' \to r'$ deve essere prodotta da una clausola $D :\equiv \Gamma' \Rightarrow l' = r', \Delta'$ con $D > C$. Ma l' è termine massimale in D per la (3) e da $D > C$ segue che l' non può essere sottotermine proprio di l (se no, si avrebbe $D < C$). Nemmeno può essere $l \equiv l'$ perchè l' è irriducibile (per la (2)) in $R_{<D}$ (che contiene già la regola $l \to r$). Quindi l'asserto (i) del Lemma è dimostrato.

(ii) Sia $C \equiv \Gamma \Rightarrow \Delta$ tale che $R_{<C} \models C$. Se $R_{<C} \models u = v$ per un letterale positivo $u = v \in \Delta$, allora $R \models u = v$ perchè $R \supseteq R_{<C}$. Altrimenti, $R_{<C} \not\models u = v$ per ogni letterale positivo $u = v \in \Delta$. Proviamo che nessuna regola di $R \setminus R_{<C}$ può ridurre un termine t che occorre in Γ (da questo segue che la R-normalizzazione dei termini che occorrono in Γ si può fare solo con le regole di $R_{<C}$, quindi da $R_{<C} \not\models u = v$ segue $R \not\models u = v$, per ogni letterale negativo $u \neq v$ di C). In effetti, le regole di $R \setminus R_{<C}$ sono prodotte da clausole $D \geq C$. Tali clausole saranno del tipo $\Gamma' \Rightarrow l = r, \Delta'$ e produrranno regole $l \to r$. Affichè l possa ridurre un termine t che occorre in Γ, occorre che sia $t \geq l$; tuttavia abbiamo $\Gamma \Rightarrow \Delta \leq \Gamma' \Rightarrow l = r, \Delta'$. Siccome le equazioni dei letterali negativi 'contano doppio' e siccome $l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta'$ (per la (3)), deve essere $l > t$ (altrimenti si avrebbe $\Gamma \Rightarrow \Delta > \Gamma' \Rightarrow l = r, \Delta'$), contraddizione. ⊣

Lemma 4.5.4 *Se la clausola*

$$C \equiv \Gamma \Rightarrow l = r, \Delta$$

produce la regola $l \to r$, allora $R \not\models \Gamma \Rightarrow \Delta$.

Dim. Per la (1), $R_{<C} \not\models \Gamma \Rightarrow \Delta$ e quindi, essendo $R \supseteq R_{<C}$, vale $R \models u = v$ per ogni letterale negativo $u \neq v$ tale che $u = v \in \Gamma$.

[17] Se questo vale per R, vale *a fortiori* per gli $R_{<C}$.

Sia, per assurdo, $\Delta \equiv u = v, \Delta_0$ con $R \models u = v$ (per la (3), abbiamo $\{l, r\} > \{u, v\}$). Siccome $R_{<C} \not\models \Gamma \Rightarrow u = v, \Delta_0$, avremo $u \not\equiv v$ (sia per esempio $u > v$) e $R_{<C} \not\models u = v$; quindi per normalizzare u e v allo stesso termine nel sistema convergente R, serve una regola $l' \to r' \in R$ prodotta da una clausola

$$D \equiv \Gamma' \Rightarrow l' = r', \Delta'$$

con $D \geq C$. Tale regola sarà applicata in un certo passo di riscrittura ad un termine $u' \leq u$. Abbiamo $l \geq u \geq u' \geq l'$, ma non può essere $l > l'$ (perchè il letterale positivo $l' = r'$ è massimale in D e perciò $l > l'$ implicherebbe $D < C$). Perciò risulta $l \equiv u \equiv l'$. Siccome $l \equiv l'$ è $R_{<D}$-irriducibile per la (2), deve essere per forza $C \equiv D$. Quindi, riassumendo,

$$D \equiv C \equiv (\Gamma \Rightarrow l = r, l = v, \Delta_0),$$

con $l > r > v$ e $R \models l = v$. Siccome $l \to r \in R$, abbiamo $R \models r = v$. Ma R è convergente per il Lemma 4.5.3 (i) e per ridurre r e v alla stessa forma normale, servono regole appartenenti a $R_{<C}$ (le altre hanno una testa l'' con $l'' \geq l$, perciò non vanno bene). Quindi $R_{<C} \models r = v$, contro la condizione (4). $\dashv$

Continuiamo ora la dimostrazione del Teorema di Completezza Refutazionale. Ricordiamo che S è un insieme di clausole chiuso rispetto alle regole del calcolo $\mathcal{S}$ (ma non contenente la clausola vuota), mentre $g(S)$ è l'insieme delle istanze ground di S. Proveremo il seguente

Lemma 4.5.5 *Per ogni $C \in g(S)$, vale che $R \models C$.*

Da questo Lemma segue immediatamente l'asserto del Teorema di Completezza Refutazionale: si consideri infatti l'insieme **A** dato dai termini ground divisi per la relazione di equivalenza

$$t \simeq v \quad \text{sse} \quad t \leftrightarrow_R^* v.$$

Per ogni simbolo di funzione n-ario f, si ponga

$$\mathcal{I}(f)([t_1], \ldots, [t_n]) := [f(t_1, \ldots, t_n)].$$

Siccome $\simeq$ è chiaramente una congruenza, $\mathcal{I}(f)$ è ben definita; inoltre risulta subito, mediante una facile induzione, che vale $\mathcal{I}(t) = [t]$ per

ogni termine ground t. Allora, però, la struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ risulta essere modello di S per il Lemma 4.5.5.[18]

Dim. del Lemma 4.5.5. Se l'enunciato del Lemma non vale, esiste C minima (rispetto all'ordinamento delle clausole ground di $g(S)$) per cui si ha $R \not\models C$. C non è la clausola vuota, pertanto ci sarà un termine s massimale in C (ossia avremo $s \geq t$ per ogni altro termine t che occorre in C). Produrremo (in contrasto con la minimalità di C) una clausola $C' \in g(S)$ tale che $C' < C$ e $R \not\models C'$.[19] Procediamo per casi, a seconda di dove occorre il termine massimale s di C.

Caso 1. s compare solo in letterali positivi di C e C è del tipo $\Gamma \Rightarrow s = s, \Delta$: questo caso è impossibile in quanto $R \not\models C$.

Caso 2. s compare solo in letterali positivi di C e C è del tipo

$$C \equiv \Gamma \Rightarrow s = t, \Delta$$

con $s > t$; possiamo anche supporre che valga $\{s, t\} \geq \{u, v\}$ per ogni altro letterale positivo $u = v \in \Delta$ (se non è così, basta scegliere un altro letterale positivo di Δ). Siccome s è massimale e non compare in Γ, abbiamo $s > u$ per ogni u che compare in Γ. Poichè per ipotesi $R \not\models C$, C non ha prodotto la regola $s \to t$: questo può avvenire solo perchè una delle condizioni (1)-(4) della definizione di clausola produttiva non sussiste. Distinguiamo quindi quattro casi, a seconda di quale di tali condizioni non sussiste.

Sottocaso 2.1. La condizione (1) non sussiste: questo è impossibile, perchè per il Lemma 4.5.3, $R \not\models C$ implica $R_{<C} \not\models C$.

Sottocasi 2.3 e 2.4. Qui abbiamo $\Delta \equiv s = t', \Delta_0$ e $R_{<C} \models t = t'$[20] (con $t \geq t'$, perchè $s = t$ è massimale in Δ). Abbiamo quindi che C è del tipo

$$\Gamma \Rightarrow s = t, s = t', \Delta_0.$$

[18] Per la precisione, si usa il Lemma 2.2.3, osservando che per ogni $a = [t] \in \mathbf{A}$, abbiamo $\mathcal{I}(\bar{a}) = a = [t] = \mathcal{I}(t)$. Quindi, se $\tilde{C} \in S$, per stabilire che $\mathcal{A} \models \tilde{C}^{\forall}$ basta provare che vale $\mathcal{A} \models \tilde{C}\sigma$ per ogni istanza ground $\tilde{C}\sigma$ di C. Ma $\tilde{C}\sigma \in g(S)$, per cui $\mathcal{A} \models \tilde{C}\sigma$ segue subito da $R \models \tilde{C}\sigma$: si noti infatti che, se $t = v$ è un'equazione ground, si ha $\mathcal{A} \models t = v$ sse $\mathcal{I}(t) = \mathcal{I}(v)$ sse $[t] = [v]$ sse $t \simeq v$ sse $t \leftrightarrow_R^* v$.

[19] Questa tecnica è talvolta chiamata 'tecnica di riduzione del controesempio'.

[20] Questo caso include anche il caso (3), quando $s = t$ occorre più volte in Δ. Si noti che le altre condizioni di (3) valgono invece, perchè le abbiamo già appurate.

Ma C è un'istanza ground $C \equiv \tilde{C}\sigma$ di una clausola $\tilde{C} \in S$ che sarà del tipo[21]

$$\tilde{\Gamma} \Rightarrow \tilde{s} = \tilde{t}, \tilde{s}' = \tilde{t}', \tilde{\Delta}_0.$$

Il fatto che $C \equiv \tilde{C}\sigma$ prova che il vincolo relativo alla Regola di Fattorizzazione per l'Uguaglianza è soddisfatto da σ, per cui S conterrà anche la clausola

$$\tilde{\Gamma}\mu, \tilde{t}\mu = \tilde{t}'\mu \Rightarrow \tilde{s}\mu = \tilde{t}'\mu, \tilde{\Delta}_0\mu,$$

dove μ è upg di $\tilde{s} \stackrel{?}{=} \tilde{s}'$. Siccome anche σ è soluzione dello stesso problema di unificazione, avremo $\sigma = \mu\theta$ per una certa sostituzione θ, per cui $g(S)$ conterrà l'istanza ground

$$\Gamma, t = t' \Rightarrow s = t', \Delta_0.$$

Siccome $R_{<C} \models t = t'$, avremo anche $R \models t = t'$. Quindi

$$R \not\models \Gamma, t = t' \Rightarrow s = t', \Delta_0,$$

che è una clausola più piccola di C (perchè il letterale positivo $s = t$ è stato sostituito dal letterale negativo $t \neq t'$ e $s > t \geq t'$).

Sottocaso 2.2. Le condizioni (3) e (4) sussistono, ma s è riducibile in $R_{<C}$. Allora esiste una clausola $D \in g(S)$, $D < C$

$$D \equiv \Gamma' \Rightarrow l = r, \Delta'$$

che ha prodotto la regola $l \to r$ che riduce s in una certa posizione p. Siccome vale la (3) sia per C che per D, da $D < C$ segue $\{s, t\} \geq \{l, r\}$; ma non può essere $(s = t) \equiv (l = r)$ (perchè R contiene la regola $l \to r$ e $R \not\models C$), per cui avremo forzatamente

$$(*) \qquad \{s, t\} > \{l, r\}.$$

Abbiamo che C e D sono istanze ground di clausole in S, per cui $C \equiv \tilde{C}\sigma$ e $D \equiv \tilde{D}\sigma$ con $\tilde{C}, \tilde{D} \in S$ (ricorrendo ad opportune rinomine,

[21] È essenziale, in passaggi di questo tipo, che le clausole siano state definite in termini di multiinsiemi e non di insiemi semplicemente: se no, la situazione si complica perchè, istanziando le clausole, alcuni letterali potrebbero venire forzatamente identificati (la $\tilde{C}$, ad esempio, non potrebbbe più essere rappresentata nella forma $\tilde{\Gamma} \Rightarrow \tilde{s} = \tilde{t}, \tilde{s}' = \tilde{t}', \tilde{\Delta}_0$ con $\tilde{\Gamma}\sigma \equiv \Gamma, \tilde{\Delta}_0\sigma \equiv \Delta_0, \tilde{s}\sigma \equiv s, \tilde{t}\sigma \equiv t, \tilde{s}'\sigma \equiv s, \tilde{t}'\sigma \equiv t'$).

possiamo far sì che $\tilde{C}$ e $\tilde{D}$ abbiano variabili disgiunte, per cui la σ si può supporre che sia la stessa sostituzione nei due casi). Avremo

$$\tilde{C} \equiv \tilde{\Gamma} \Rightarrow \tilde{s} = \tilde{t}, \tilde{\Delta}$$

$$\tilde{D} \equiv \tilde{\Gamma}' \Rightarrow \tilde{l} = \tilde{r}, \tilde{\Delta}'.$$

Sappiamo che $\tilde{l}\sigma \equiv l \equiv s_{|p} \equiv (\tilde{s}\sigma)_{|p}$, quindi ci sono due possibilità (che analizzeremo separatamente):[22] 1) $p \in Pos(\tilde{s})$ e $\tilde{s}_{|p}$ non è una variabile; 2) esistono p', q con $p'q = p$, tali che $p' \in Pos(\tilde{s})$ e $\tilde{s}_{|p'}$ è una variabile.

Sotto-sottocaso 2.2.1. In questo caso, $s_{|p} \equiv (\tilde{s}_{|p})\sigma \equiv \tilde{l}\sigma$,[23] per cui il problema di unificazione $\tilde{s}_{|p} \stackrel{?}{=} \tilde{l}$ ha upg μ e risulta $\mu\theta = \sigma$, per una certa sostituzione θ. Poichè i relativi vincoli sono soddisfatti da σ,[24] S contiene la clausola

$$\tilde{\Gamma}\mu, \tilde{\Gamma}'\mu \Rightarrow (\tilde{s}\mu)[\tilde{r}\mu]_p = \tilde{t}\mu, \tilde{\Delta}\mu, \tilde{\Delta}'\mu$$

ottenuta per Sovrapposizione Destra da $\tilde{C}$ e $\tilde{D}$. Istanziando con la sostituzione θ, abbiamo che $g(S)$ contiene la clausola

$$\Gamma, \Gamma' \Rightarrow s[r]_p = t, \Delta, \Delta'.$$

Per il Lemma 4.5.4, $R \not\models \Gamma' \Rightarrow \Delta'$; inoltre sappiamo già che $R \not\models C$, cioè che $R \not\models \Gamma \Rightarrow s = t, \Delta$. Siccome $l \to r \in R$, da $R \not\models s = t$, segue $R \not\models s[r]_p = t$ (perchè $s_{|p} \equiv l$). Quindi, in conclusione

$$R \not\models \Gamma, \Gamma' \Rightarrow s[r]_p = t, \Delta, \Delta'.$$

Questa clausola è più piccola di $C \equiv \Gamma \Rightarrow s = t, \Delta$, perchè $\{s, t\} > \{s[r]_p, t\}$, perchè $l = r$ è letterale massimale in D e per la $(*)$.

Sotto-sottocaso 2.2.2. Qui abbiamo che esiste una variabile x ed esistono delle posizioni p', q tali che $p = p'q$ e $\tilde{s}_{|p'} \equiv x$. Consideriamo la sostituzione ground σ' così definita

$$x\sigma' :\equiv (x\sigma)[r]_q, \quad y\sigma' :\equiv y\sigma \text{ (per } y \not\equiv x).$$

Si osservi che $l \equiv (\tilde{s}\sigma)_{|p} \equiv ((\tilde{s}\sigma)_{|p'})_{|q} \equiv ((\tilde{s})_{|p'}\sigma)_{|q} \equiv (x\sigma)_{|q}$.[25] Siccome $l \to r \in R$, abbiamo $z\sigma \to_R^* z\sigma'$ per ogni variabile z. Perciò, siccome

[22] Si veda la Proposizione 2.6.1.
[23] In questi passaggi si usa la Proposizione 2.6.1(i).
[24] Questo è dovuto alla $(*)$ e al fatto che la (3) vale sia per C che per D.
[25] Abbiamo usato il Lemma 2.3.2 in questi passaggi.

$R \not\models C \equiv \tilde{C}\sigma$, avremo anche $R \not\models \tilde{C}\sigma'$. Tuttavia $\tilde{C}\sigma > \tilde{C}\sigma'$ (perchè $\tilde{C}\sigma'$ si ottiene da $\tilde{C}\sigma$ per riscrittura mediante R).[26] Abbiamo anche in questo caso trovato una clausola $\tilde{C}\sigma' \in g(S)$ minore della C e tale che $R \not\models \tilde{C}\sigma'$.

Caso 3. s compare in un letterale negativo $s \neq s$ di C, cioè C è del tipo

$$\Gamma, s = s \Rightarrow \Delta.$$

Esisterà una clausola $\tilde{C} \in S$, sia essa

$$\tilde{\Gamma}, \tilde{s} = \tilde{s}' \Rightarrow \tilde{\Delta},$$

tale che $\tilde{C}\sigma \equiv C$ per una certa σ ground. Siccome $\tilde{s}\sigma \equiv s \equiv \tilde{s}'\sigma$, il problema di unificazione $\tilde{s} \stackrel{?}{=} \tilde{s}'$ è risolubile e ammette un upg, sia esso μ; avremo anche $\mu\theta = \sigma$, per una certa θ. Siccome s è massimale in C, il letterale negativo $s \neq s$ è pure massimale in C, perciò è possibile applicare la Regola di Risoluzione per l'Uguaglianza a $\tilde{C}$ (il relativo vincolo è soddisfatto dalla σ), ottenendo la clausola

$$\tilde{\Gamma}\mu \Rightarrow \tilde{\Delta}\mu$$

che apparterrà ad S ed avrà come istanza ground (tramite la θ) la clausola $C' \equiv \Gamma \Rightarrow \Delta$, minore della C e tale che $R \not\models C'$.

Caso 4. s compare in letterali negativi $s \neq t$ di C con $s > t$ (e in nessun letterale negativo del tipo $s \neq s$). Sia $s \neq t$ massimale fra i letterali negativi in cui s compare; allora C è del tipo

$$\Gamma, s = t \Rightarrow \Delta$$

e $s \neq t$ è letterale massimale in C.[27] Siccome $R \models C$, abbiamo in particolare che $R \models s = t$, per cui s sarà riducibile (in una certa posizione p) mediante una regola $l \to r$ prodotta da una certa clausola $D \in g(S)$ del tipo

$$\Gamma' \Rightarrow l = r, \Delta'.$$

[26] Tale riscrittura può avvenire in più passi (a seconda del numero della occorrenze della variabile x in $\tilde{C}$); almeno un passo deve però essere effettuato, ossia quello che rimpiazza nel letterale $s = t$, il sottotermine $s_{|p}$ con r.

[27] Questo avviene anche se in C sono presenti letterali positivi del tipo $s = t'$, per il criterio di confronto fra letterali negativi e positivi, cfr. il paragrafo 4.2.

C e D sarranno istanze (tramite la stessa sostituzione ground σ) di certe clausole (che possiamo supporre a variabili disgiunte) $\tilde{C}, \tilde{D} \in S$, siano esse

$$\tilde{C} \equiv \tilde{\Gamma}, \tilde{s} = \tilde{t} \Rightarrow \tilde{\Delta}$$

$$\tilde{D} \equiv \tilde{\Gamma}' \Rightarrow \tilde{l} = \tilde{r}, \tilde{\Delta}'.$$

Sappiamo che $\tilde{l}\sigma \equiv l \equiv s_{|p} \equiv (\tilde{s}\sigma)_{|p}$, quindi anche ora ci sono due possibilità (che analizzeremo separatamente): 1) $p \in Pos(\tilde{s})$ e $\tilde{s}_{|p}$ non è una variabile; 2) esistono p', q con $p'q = p$, tali che $p' \in Pos(\tilde{s})$ e $\tilde{s}_{|p'}$ è una variabile.

Sottocaso 4.1. In questo caso, $s_{|p} \equiv (\tilde{s}_{|p})\sigma \equiv \tilde{l}\sigma$, per cui il problema di unificazione $\tilde{s}_{|p} \stackrel{?}{=} \tilde{l}$ ha upg μ e risulta $\mu\theta = \sigma$, per una certa sostituzione θ. Poichè i relativi vincoli sono soddisfatti da σ, S contiene la clausola

$$\tilde{\Gamma}\mu, \tilde{\Gamma}'\mu, (\tilde{s}\mu)[\tilde{r}\mu]_p = \tilde{t}\mu \Rightarrow \tilde{\Delta}\mu, \tilde{\Delta}'\mu$$

ottenuta per Sovrapposizione Sinistra da $\tilde{C}$ e $\tilde{D}$. Istanziando con la sostituzione ground θ, abbiamo che $g(S)$ contiene la clausola

$$\Gamma, \Gamma', s[r]_p = t, \Rightarrow \Delta, \Delta'.$$

Per il Lemma 4.5.4, $R \not\models \Gamma' \Rightarrow \Delta'$; siccome $l \to r \in R$, da $R \models s = t$, segue $R \models s[r]_p = t$ (perchè $s_{|p} \equiv l$). Quindi, in conclusione

$$R \not\models \Gamma, \Gamma', s[r]_p = t \Rightarrow \Delta, \Delta'.$$

Ma questa clausola è minore della C perchè il letterale negativo $s \neq t$ di C è stato rimpiazzato da $\Gamma', s[r]_p = t \Rightarrow \Delta'$: abbiamo infatti che $s > s[r]_p$ per riscrittura, che $s \geq l > r$ e che $l = r$ è letterale massimale in D per la condizione (3) della definizione di clausola produttiva.[28]

Sottocaso 4.2. Qui esiste una variabile x ed esistono delle posizioni p', q tali che $p = p'q$ e $\tilde{s}_{|p'} \equiv x$. Sia σ' la sostituzione così definita

$$x\sigma' :\equiv (x\sigma)[r]_q, \quad y\sigma' :\equiv y\sigma \text{ (per } y \not\equiv x).$$

[28] Si ricordi che i termini dei letterali negativi 'contano doppio', quindi nel caso in cui $s \equiv l$ e in cui l compare in Δ', abbiamo che il letterale negativo $s \neq t$ vale come il multiinsieme $\{s, s, t, t\}$, mentre un'eventuale equazione $l = u$ (necessariamente con $l > u$, perchè $l = r$ è massimale in D) all'interno di Δ' vale solo come il multiinsieme $\{l, u\}$.

Siccome $l \to r \in R$, abbiamo $z\sigma \to_R^* z\sigma'$ per ogni variabile z (si ricordi che $l \equiv (\tilde{s}\sigma)_{|p} \equiv ((\tilde{s}\sigma)_{|p'})_{|q} \equiv ((\tilde{s})_{|p'}\sigma)_{|q} \equiv (x\sigma)_{|q}$). Perciò, siccome $R \not\models C \equiv \tilde{C}\sigma$, avremo anche $R \not\models \tilde{C}\sigma'$. Infine, per riscrittura, $\tilde{C}\sigma > \tilde{C}\sigma'$. Questo chiude l'ultimo caso rimasto e completa perciò la dimostrazione del Teorema di Completezza Refutazionale. $\dashv$

La dimostrazione che abbiamo visto può essere facilmente adattata al calcolo $\mathcal{S}^+$ (che prevede anche un meccanismo preassegnato di marcatura dei letterali negativi):

Teorema 4.5.6 *Il calcolo $\mathcal{S}^+$ è refutazionalmente completo.*

Dim. Diamo solo un cenno alle modifiche rilevanti da apportare (lasciando i facili dettagli al lettore). Innanzitutto, affinchè una clausola $C \in g(S)$ sia produttiva occorrerà (in aggiunta alle condizioni (0)-(4) di cui sopra) anche che *esista* una clausola $\tilde{C} \in S$ che *non abbia letterali negativi marcati* e per cui valga $\tilde{C}\sigma \equiv C$ (per una certa sostituzione σ). I Lemmi 4.5.3 e 4.5.4 si provano come prima; per il Lemma 4.5.5, si supponga ancora che $C \in g(S)$ sia una clausola minima tale che $R \not\models C$. Se C è istanza ground di una clausola di S che non ha letterali negativi marcati, si procede come prima. Altrimenti, esistono $\tilde{C} \in S$ ed una sostituzione σ tali che $\tilde{C}\sigma \equiv C$ e tali che $\tilde{C}$ è del tipo

$$\tilde{C} \equiv (\tilde{\Gamma}, (\tilde{s} = \tilde{t})^+ \Rightarrow \tilde{\Delta}).$$

Di conseguenza, C sarà uguale a

$$\Gamma, s = t \Rightarrow \Delta.$$

Se $s \equiv t$, si procede come nel Caso 3, mentre se $s > t$ si procede come nel Caso 4 (l'unica differenza è che ora i vincoli da verificare sono più deboli). $\dashv$

4.6 Ridondanze

Un dimostratore automatico deve saper gestire il processo di saturazione delle clausole di una teoria. 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 di saturazione.

In questo paragrafo affrontiamo questo problema, introducendo opportune nozioni di ridondanza, sia per inferenze che per clausole (sulla nozione di ridondanza per inferenze si baserà la 'forward reduction', mentre la 'backward reduction' si baserà sulla nozione di ridondanza per clausole).

Ricordiamo che all'inizio del paragrafo 4.5, abbiamo introdotto un ordinamento totale e terminante $>$ sulle clausole ground del nostro linguaggio; per farlo abbiamo utilizzato la doppia estensione ai multiinsiemi dell'ordine di riduzione che opera sui termini del linguaggio: ricordiamo infatti che la clausola ground

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

rappresenta il multiinsieme di multiinsiemi

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

Se S è un insieme di clausole, $g(S)$ indica l'insieme delle clausole che sono istanze ground di clausole in S. Se

$$\frac{C_1, \ldots, C_n}{D \parallel V}$$

è un esempio (π) di una regola di inferenza,[29] un'*istanza ground* ($\pi\sigma$) di (π) è una regola del tipo

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

[29]Per i calcoli $\mathcal{S}$ e $\mathcal{S}^+$ cui facciamo riferimento, si avrà ovviamente $n \leq 2$.

dove σ è una sostituzione ground che soddisfa il vincolo V.[30]

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

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

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

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

Possiamo ora introdurre il concetto fondamentale di derivazione (prendiamo come riferimento il calcolo $\mathcal{S}^+$,[31] quindi parlando di 'regola di inferenza' intendiamo sempre uno dei sei tipi di regole di inferenza introdotti nelle Tabelle 4.1 e 4.3). 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;

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

Si noti che la condizione (i) vale ovviamente sempre se C è conclusione di una regola di inferenza di $\mathcal{S}^+$ le cui premesse stanno in S_i. La formulazione generale di (i)-(ii) che abbiamo dato ci sarà utile più avanti quando esamineremo esempi concreti di regole di riduzione. Grazie a (i)-(ii) potremo infatti sostituire clusole correnti con clausole 'più semplici': succederà infatti che aggiungendo a S_i una certa clausola

[30] Nelle regole di inferenza di $\mathcal{S}$ e $\mathcal{S}^+$, la conclusione D è sempre del tipo $D'\mu$, dove μ è upg di un certo problema di unificazione. Siccome però la σ è soluzione dello stesso problema di unificazione e siccome μ può essere scelto idempotente (come osservato nel paragrafo 2.6), abbiamo $\sigma = \mu\sigma'$ (per una certa σ'), per cui $D'\mu\sigma \equiv D'\mu\mu\sigma' \equiv D'\mu\sigma' \equiv D'\sigma$. Questo prova che l'upg μ della conclusione delle regole di inferenza può essere ignorato, prendendo in considerazione le loro istanze.

[31] $\mathcal{S}$ è un caso particolare di $\mathcal{S}^+$, che si realizza quando nessun letterale negativo è marcato.

D (conseguenza logica di S_i) potremo poi eliminare da $S_i \cup \{D\}$ una certa C, più complessa della D, divenuta ridondante proprio grazie all'aggiunta della D.

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 4.6.1 *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 (che dimostreremo nel prossimo paragrafo) è facilitato dalla seguente peculiarità di immediata verifica relativa al calcolo $\mathcal{S}^+$: in ogni istanza ground di una regola del calcolo $\mathcal{S}^+$, *la conclusione è sempre strettamente minore di una delle premesse.*[32] 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 4.8 vedremo che con una opportuna funzione di scelta della clausola data, questa proprietà è automaticamente garantita nelle implementazioni). Tuttavia, quando un'inferenza (π) è già ridondante rispetto all'insieme delle clausole correnti S_i, *essa può essere ignorata*: in tal caso, infatti, la (π) sarà a maggior ragione ridondante anche in S_∞.

[32] Abbiamo in effetti già verificato tale semplice proprietà nel corso della dimostrazione del Teorema di Completezza Refutazionale. Si noti che, nel caso di inferenze a due premesse, la premessa massimale è sempre quella destra, per come sono disposti i vincoli.

Elenchiamo qui di seguito alcuni schemi di regole di riduzione implementate sui dimostratori automatici correnti, specificandone l'utilizzo sia in modalità 'forward' (cioè per scartare inferenze fra clausole correnti) che in modalità 'backward' (cioè per eliminare clausole correnti).

- **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**: una clausola D *sussume* la clausola C, qualora per una certa μ si abbia $D\mu \subseteq C$;[33] 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 $C_1, \ldots, C_m/C \parallel V$ 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).[34]

- **Regola MRR** ('Matching Replacement Resolution'):[35] 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[36]

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

[33] Ovviamente, se $D \equiv (\Gamma \Rightarrow \Delta)$ e $C \equiv (\Gamma' \Rightarrow \Delta')$, $D\mu \subseteq C$ significa ($\Gamma\mu \subseteq \Gamma'$ e $\Delta\mu \subseteq \Delta'$). Si noti che, trattandosi di multiinsiemi, notazioni come $\Gamma\mu \subseteq \Gamma'$ significano che ogni formula atomica che occorre in $\Gamma\mu$ occorre anche in Γ' e in un numero maggiore o uguale di volte.

[34] Si ricordi la proprietà sopra menzionata del calcolo $\mathcal{S}^+$ per cui l'istanza ground della conclusione è minore della corrispondente istanza della premessa massimale.

[35] Questa Regola è implementata ad esempio su SPASS.

[36] Questa è la versione destra della regola, ovviamente ne esiste anche una versione sinistra (del tutto simmetrica) in cui A_1 compare come letterale negativo e A_2 compare come letterale positivo.

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 $\Gamma_2 \Rightarrow \Delta_2$ (infatti, una volta aggiunta la $\Gamma_2 \Rightarrow \Delta_2$, la $\Gamma_2, A_2 \Rightarrow \Delta_2$ diventa subito ridondante).

- **Riflessività**: la clausola $s = s, \Gamma \Rightarrow \Delta$ può essere sostituita dalla clausola $\Gamma \Rightarrow \Delta$.

- **Demodulazione** (o riscrittura): supponiamo che valga $l > r$ e che la clausola unitaria $\Rightarrow l = r$ 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 regola di riscrittura $l \to r$: questo fatto significa che, in una certa posizione p, la C contiene un termine s per cui vale $s \equiv l\mu$ (per un matcher μ) e la D è ottenuta dalla C sostituendo in posizione p il termine s con $r\mu$. Se succede che vale $C\sigma > \Rightarrow l\sigma = r\sigma$ per ogni σ ground, 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 modalità 'forward' (almeno nel caso del Calcolo $\mathcal{S}$ applicato a clausole Horn): se $C_1, \ldots, C_n/C \parallel V$ è una regola di inferenza di $\mathcal{S}$ e S_i contiene la clausola $\Rightarrow l = r$ (con $l > r$) che giustifica il passo di riscrittura $C \to D$, allora per come sono fatte le regole di $\mathcal{S}$ nel caso di clausole Horn,[37] si ha che ponendo $S_{i+1} = S_i \cup \{D\}$, l'inferenza diventa ridondante e può essere ignorata.

Osserviamo infine che la Demodulazione si può applicare anche quando la condizione $l > r$ sull'equazione demodulante $\Rightarrow l = r$ è

[37] Le clausole Horn sono quelle che contengono al più un letterale positivo. Il problema nel caso non Horn sta nel fatto che può succedere che il termine s della premessa destra della Regola di Sovrapposizione Destra coincida con il termine l della premessa sinistra della stessa regola (qui facciamo riferimento alla Tabella 4.1) e con la testa della regola di riscrittura usata nella demodulazione; allora se l ha più occorrenze in un letterale positivo delle premesse, una delle quali viene demodulata, la condizione che la premessa massimale dell'inferenza (cioè quella destra) sia maggore della regola di riscrittura demodulante può fallire.

sostituita ad esempio dalla condizione più debole $l\mu > r\mu$ (qui μ è il matcher usato nella riscrittura $C \to D$).

ESEMPIO 9. ('Congruence Closure'). *Il processo di saturazione di un insieme S di clausole unitarie ground termina sempre* (una clausola è unitaria, qualora consista di un unico letterale). Infatti, se S non è già saturo, allora è possibile applicare un'inferenza a S. Vediamo i casi possibili (tenendo conto del fatto che tutte le clausole sono ground). La Regola di Fattorizzazione per l'Uguaglianza non si può mai applicare (esse infatti prevede un premessa non Horn) e la Regola di Risoluzione per l'Uguaglianza, se applicabile, produce immediatamente la clausola vuota e la terminazione del processo di saturazione. Alternativamente, va applicata una Regola di Sovrapposizione, con premessa sinistra una clausola del tipo $\Rightarrow l = r$, con $l > r$. Ma sia per la Sovrapposizione Sinistra che per la Sovrapposizione Destra, la premessa destra della regola diventa subito ridondante dopo l'applicazione della regola stessa (per come sono fatti i vincoli e per come è definito l'ordinamento delle clausole ground); quindi, applicando una Regola di Sovrapposizione, se ne può subito eliminare la premessa destra dall'insieme delle clausole correnti. L'insieme S' di clausole che si ottiene in questo modo è minore di S nell'estensione ai multiinsiemi dell'ordinamento delle clausole ground, perciò il processo di saturazione non può divergere.[38]

ESEMPIO 10. Si può estendere il risultato dell'Esempio precedente al caso di insiemi di clausole Horn ground: basta utilizzare $\mathcal{S}^+$ e marcare sempre un letterale negativo, qualora ce ne sia almeno uno. In tal modo, solo clausole unitarie positive sono utilizzabili nelle regole di Sovrapposizione e il ragionamento di cui sopra si applica senza rilevanti modifiche.

ESEMPIO 11. Abbiamo visto che la nostra teoria delle ridondanze consente l'uso in modalità 'backward' della Demodulazione solo in presenza della condizione $C\sigma > \Rightarrow l\sigma = r\sigma$: questa condizione è automaticamente soddisfatta in molti casi, ad esempio qualora il termine demodulato occorra in un letterale negativo di C oppure qualora la demodulazione non avvenga in radice. Ci si può chiedere se una diversa teoria delle ridondanze possa consentire un uso differente della Demodulazione (che è una regola di riduzione molto importante), in analogia a quanto par-

[38] Ci sono controesempi che mostrano, invece, che senza l'uso di passi di riduzione, la terminazione non è garantita.

zialmente avviene nel completamento di Knuth-Bendix per i sistemi di riscrittura ordinata. Ciò è forse possibile, segnaliamo però con un controesempio che un uso improvvido della Demodulazione può produrre effetti deleteri, fino ad inficiare la validità stessa del Teorema 4.6.1.

Consideriamo le tre clausole

$$\Rightarrow f(g(f(x))) = f(g(x)), \quad \Rightarrow g(g(x)) = g(x), \quad f(g(f(a))) = f(g(a)) \Rightarrow$$

e utilizziamo un ordine LPO basato sulla precedenza $f >_p g >_p a$. Ovviamente tale insieme di clausole è inconsistente (basta utilizzare la Regola di Risoluzione fra la prima e la terza clausola). Invece di procedere così, costruiamo (in modo contorto, mediante un uso illegale della Demodulazione) una derivazione (δ) tale che per ogni n ci sia uno stadio i_n di (δ) in cui l'insieme delle clausole correnti S_{i_n} è dato da

$$\Rightarrow f(g(f(x))) = f(g^{2^n}(x)),\ \Rightarrow g(g(x)) = g(x),\ f(g(f(a))) = f(g(a)) \Rightarrow$$

Fatto questo, avremo subito un controesempio al Teorema 4.6.1: la derivazione è infatti equa, perchè le uniche clausole persistenti sono

$$\Rightarrow g(g(x)) = g(x), \quad f(g(f(a)) = f(g(a)) \Rightarrow$$

e fra di esse non è possibile alcuna inferenza.

i_0 è lo stadio iniziale della derivazione. Se abbiamo già raggiunto i_n, raggiungiamo i_{n+1} come segue. Dapprima inferiamo per Sovrapposizione Destra la clausola

$$\Rightarrow f(g^{2^n}(g(f(x)))) = f(g(f(g^{2^n}(x))))$$

che demoduliamo, mediante le clausole $\Rightarrow f(g(f(x))) = f(g^{2^n}(x))$ e $\Rightarrow g(g(x)) = g(x)$, fino ad ottenere la clausola $\Rightarrow f(g(f(x))) = f(g^{2^{n+1}}(x))$. Poi usiamo quest'ultima e la clausola $\Rightarrow g(g(x)) = g(x)$ per demodulare la clausola $\Rightarrow f(g(f(x))) = f(g^{2^n}(x))$[39] fino a ridurla ad una clausola tautologica, che eliminiamo all'istante. Così raggiungiamo l'insieme $S_{i_{n+1}}$ voluto.

Nel paragrafo 5.2, vedremo un ulteriore esempio di applicazione di regole di riduzione a clausole unitarie.

[39] Questo è il passaggio illegale: abbiamo demodulato una clausola mediante un'altra maggiore.

4.7 Completezza con regole di riduzione

In questo paragrafo ci occupiamo esclusivamente della dimostrazione del Teorema 4.6.1.

Un insieme di clausole S è *saturo* sse ogni regola fra premesse appartenenti a S è ridondante in S (questa condizione, come si è visto, è più debole della richiesta che S sia chiuso rispetto alle regole del calcolo).

Lemma 4.7.1 *Se S è saturo, S è inconsistente sse S contiene la clausola vuota.*

Dim. Si procede come nella dimostrazione del Teorema 4.5.1. Il punto-chiave è la dimostrazione del Lemma 4.5.5: si tratta anche ora di stabilire che per ogni $C \in g(S)$ vale $R \models C$, solo che ora S è supposto solo essere saturo (e non chiuso rispetto alle regole del calcolo). In tutti i vari Casi occorrenti nella dimostrazione del Lemma 4.5.5, data una clausola $C \in g(S)$ tale che $R \not\models C$, veniva prodotta una clausola ground D tale che $D < C$, $D \in g(S)$ e tale che $R \not\models D$. Lo schema utilizzato per produrre D era il seguente: si utilizzava una regola (π) del calcolo e se ne prendeva un'istanza ground $(\pi\sigma)$ avente la C come premessa massimale e la D come conclusione. L'unica informazione che non possiamo ricavare ancora nel caso presente è l'appartenenza di D a $g(S)$. Tuttavia, se $D \notin g(S)$, questo si può verificare solo perchè la (π) è ridondante in S. Dunque

$$\{E \in g(S) \mid E < C\} \models D,$$

ma da questo segue che esiste $D' \in g(S)$, tale che $D' < C$ e $R \not\models D'$,[40] quindi questa D' può sostituire la D e il ragionamento può proseguire nello stesso modo. ⊣

Fissiamo ora una derivazione equa

$$(\delta) \qquad S_1, S_2, \ldots, S_i, \ldots$$

Lemma 4.7.2 *Se $\tilde{C} \in (S_\infty \setminus S_\omega)$, allora $\tilde{C}$ è ridondante in S_ω.*

[40] Qui R è ambiguamente usato come sistema di riscrittura ground e come struttura, ma l'ambiguità è giustificata dalla costruzione della struttura herbrandiana $\mathcal{A}$ introdotta nel paragrafo 4.5 subito dopo l'enunciato del Lemma 4.5.5: in tale struttura $\mathcal{A}$ si ha infatti $\mathcal{A} \models C$ sse $R \models C$, per ogni clausola ground C.

Dim. Sia $\tilde{C}$ una tale clausola non persistente e sia σ una sostituzione tale che $C :\equiv \tilde{C}\sigma$ è ground; si tratta di provare che

$$(I) \qquad \{E \in g(S_\omega) \,|\, E < C\} \models C.$$

Questo fatto viene provato per induzione trasfinita su C: supponiamo che la condizione valga per tutte le $D < C$ che siano istanze ground di clausole non persistenti. Siccome $\tilde{C}$ non è persistente, $\tilde{C}$ è stata rimossa da un certo S_j perchè $\tilde{C}$ era ridondante rispetto a S_j, cioè in particolare per $C \equiv \tilde{C}\sigma$ si aveva

$$(II) \qquad \{D \in g(S_j) \,|\, D < C\} \models C.$$

Ma le $D \in g(S_j), D < C$ o sono istanze di clausole persistenti o ad esse si applica l'ipotesi induttiva. In entrambi i casi abbiamo, per ogni tale D, sfruttando il fatto che $D < C$,

$$(III) \qquad \{E \in g(S_\omega) \,|\, E < C\} \models D.$$

Ma allora (I) segue immediatamente da (II) e (III). ⊣

Lemma 4.7.3 *S_1 è soddisfacibile sse S_ω è soddisfacibile.*

Dim. Siccome S_1 e S_∞ sono teorie logicamente equivalenti (perchè tutte le regole del calcolo sono valide), basterà provare che se S_ω è soddisfacibile tale è S_∞ (il viceversa è ovvio perchè $S_\omega \subseteq S_\infty$). Sia dunque $\mathcal{A}$ una struttura che è modello di S_ω; poichè S_ω è una teoria universale, la sottostruttura $\mathcal{B}$ di $\mathcal{A}$ individuata dalle interpretazioni dei termini ground è ancora un modello di S_ω e basterà stabilire che vale $\mathcal{B} \models C$ per ogni $C \in g(S_\infty)$. Avremo $C \equiv \tilde{C}\sigma$ per una certa clausola $\tilde{C}$. Se $\tilde{C}$ è persistente, $\tilde{C}\sigma \in g(S_\omega)$, se no per il Lemma 4.7.2, $\tilde{C}$ è ridondante in S_ω, quindi in particolare abbiamo $g(S_\omega) \models \tilde{C}\sigma \equiv C$, da cui $\mathcal{B} \models C$. ⊣

Il prossimo Lemma 4.7.4, combinato con i Lemmi 4.7.1, 4.7.3 visti in precedenza, dà immediatamente la prova del Teorema 4.6.1.

Lemma 4.7.4 *S_ω è saturo.*

Dim. Sia (π) una regola del calcolo avente le clausole $\tilde{C}_1, \ldots, \tilde{C}_n \in S_\omega$ come premesse e la clausola $\tilde{D}$ come conclusione. Per l'equità della

derivazione (δ), la (π) è ridondante in S_∞; dobbiamo provare una analoga proprietà di ridondanza rispetto a S_ω. Sia dunque ($\pi\sigma$) un'istanza ground della (π); per la ridondanza rispetto a S_∞, abbiamo (sia $\tilde{C}_m\sigma$ la premessa massimale della ($\pi\sigma$))

$$\{E \in g(S_\infty) \,|\, E < \tilde{C}_m\sigma\} \models \tilde{D}\sigma.$$

Ma le $E \in g(S_\infty)$ sono o istanze di clausole persistenti oppure istanze di clausole ridondanti in S_ω per il Lemma 4.7.2; in quest'ultima ipotesi si ha $\{E' \in g(S_\omega) \,|\, E' < E\} \models E$ e quindi in ogni caso

$$\{E \in g(S_\omega) \,|\, E < \tilde{C}_m\sigma\} \models \tilde{D}\sigma,$$

come richiesto. ⊣

4.8 Il ciclo della clausola data

Vediamo ora un semplice schema di implementazione a livello globale basato sul Teorema 4.6.1.

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 *'Saturation 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*.

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 modalità 'forward' per semplificare l'insieme *New* (tramite clausole in *New*, poi in *Wo* ed infine anche in *Us*).

4. Si applica la riduzione in modalità 'backward' 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 relativa 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 (in tal modo la completezza refutazionale viene preservata grazie al Teorema 4.6.1). Usualmente si sceglie la clausola 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 qualche modo convenzionale.

4.9 Il calcolo B

Sono possibili ulteriori raffinamenti del calcolo $\mathcal{S}$. Riportiamo brevemente, per completezza di esposizione, la cosiddetta versione *basica* ('basic') $\mathcal{B}$ del calcolo. Le regole di inferenza del calcolo $\mathcal{B}$ sono essenzialmente le stesse del calcolo $\mathcal{S}$ (le abbiamo riportate per comodità nella Tabella 4.4).

La differenza consiste in una diversa gestione dei vincoli: quando si utilizza una regola, *non si applica più nessun upg alla conclusione*, piuttosto, dopo aver semplicemente verificato che il vincolo è soddisfacibile, *lo si trascina e lo si accumula* con i vincoli successivi. Ciò non solo porta a vincoli sempre più stringenti (quindi più soggetti ad insoddisfacibilità, con l'effetto di bloccare sempre più applicazioni di regole

Tabella 4.4: Regole di Inferenza di $\mathcal{B}$

Sovrapposizione Destra

$$\frac{\Gamma' \Rightarrow \Delta', l = r \;\|\; V_1 \qquad \Gamma \Rightarrow s = t, \Delta \;\|\; V_2}{\Gamma, \Gamma' \Rightarrow s[r]_p = t, \Delta, \Delta' \;\|\; VD \;\&\; V_1 \;\&\; V_2}$$

(se $s_{|p}$ non è una variabile)

Sovrapposizione Sinistra

$$\frac{\Gamma' \Rightarrow \Delta', l = r \;\|\; V_1 \qquad \Gamma, s = t \Rightarrow \Delta \;\|\; V_2}{\Gamma, \Gamma', s[r]_p = t \Rightarrow \Delta, \Delta' \;\|\; VS \;\&\; V_1 \;\&\; V_2}$$

(se $s_{|p}$ non è una variabile)

Risoluzione per l'Uguaglianza

$$\frac{s = s', \Gamma \Rightarrow \Delta \;\|\; V}{\Gamma \Rightarrow \Delta \;\|\; VE1 \;\&\; V}$$

Fattorizzazione per l'Uguaglianza

$$\frac{\Gamma \Rightarrow s = t, s' = t', \Delta \;\|\; V}{\Gamma, t = t' \Rightarrow s = t', \Delta \;\|\; VE2 \;\&\; V}$$

I vincoli $VD, VS, VE1, VE2$ sono così specificati:

VD $\quad s_{|p} \stackrel{?}{=} l \;\&\; l \stackrel{?}{>} r \;\&\; s \stackrel{?}{>} t \;\&\; s = t \stackrel{?}{>} l = r \;\&$
$l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta' \;\&$
$s = t$ è strettamente massimale in $\Gamma \Rightarrow s = t, \Delta$.

VS $\quad s_{|p} \stackrel{?}{=} l \;\&\; l \stackrel{?}{>} r \;\&\; s \stackrel{?}{>} t \;\&$
$l = r$ è strettamente massimale in $\Gamma' \Rightarrow l = r, \Delta' \;\&$
$s \neq t$ è massimale in $\Gamma, s = t \Rightarrow \Delta$.

$VE1$ $\quad s \stackrel{?}{=} s' \;\&\; s \neq s'$ è massimale in $\Gamma, s = s' \Rightarrow \Delta$.

$VE2$ $\quad s \stackrel{?}{=} s' \;\&\; s \stackrel{?}{>} t \;\&$
$s = t$ è massimale in $\Gamma \Rightarrow s = t, s' = t', \Delta$.

di inferenza), ma - questo è il punto principale - di fatto impedisce alle regole di inferenza di intervenire sui sottotermini che verrebbero introdotti dagli upg via via calcolati.

Il calcolo $\mathcal{B}$ è ancora *refutazionalmente completo*: *un insieme di clausole S (senza vincoli) è inconsistente se e solo se da esso si può derivare la clausola vuota utilizzando le regole di inferenza di $\mathcal{B}$ con le modalità sopra specificate.*[41] Per dimostrarlo, occorre apportare alcune modifiche (tecnicamente fastidiose, almeno nel caso non Horn) ai metodi che abbiamo utilizzato nel paragrafo 4.5: il motivo di tali modifiche è dovuto al fatto che, siccome il processo di saturazione di S produce un insieme S' di clausole con vincoli, è chiaro che $g(S')$ dovrà essere definito in modo da contenere solo le istanze ground delle clausole di S' che soddisfano il relativo vincolo. Allora però, l'argomento utilizzato nel Sotto-sottocaso 2.2.2 e nel Sottocaso 4.2 della dimostrazione del Lemma 4.5.5 non funzione più. L'enunciato stesso del Lemma 4.5.5 andrà modificato chiedendo che valga $R \models C$ solo per le istanze ground di clausole di S' ottenute mediante sostituzioni che siano 'R-irriducibili'.

Segnaliamo infine che il calcolo $\mathcal{B}$ non è compatibile con la teoria delle ridondanze esposta nel paragrafo 4.6 e richiede una sua propria teoria delle ridondanze.

[41] L'ovvia estensione di questo risultato al caso in cui S è un insieme di clausole con vincoli invece non vale (a meno di imporre delle condizioni).

4.10 Nota bibliografica

Il primo calcolo basato su metodi di saturazione fu introdotto da J. A. Robinson in [79]; raffinamenti alla Regola di Risoluzione da lui proposta sono stati estensivamente studiati attorno agli anni '70. Per esposizioni comprensive sui calcoli con Risoluzione per linguaggi senza identità, si vedano ad esempio i testi [23], [59] e il recente lavoro di rassegna [16].

La Regola di Paramodulazione fu introdotta da Robinson e Wos in [80],[42] ma solo più tardi si arrivò a giustificarne l'uso senza i fastidiosi assiomi di funzionalità riflessiva e senza la dispendiosa paramodulazione sulle variabili[43] [21], [75]. Ulteriori miglioramenti portarono progressivamente al 'Superposition Calculus': dapprima, vincoli di ordinamento furono imposti all'applicazione della Regola, ottenendo la cosiddetta Paramodulazione Ordinata, la cui completezza refutazionale fu provata in [47]. Tali vincoli vennero poi rafforzati fino alla formulazione corrente, che richiede che la regola sia applicata solo a termini massimali di letterali massimali. La prova di completezza refutazionale per il calcolo $\mathcal{S}$, basata sul metodo di 'model generation' e sulla relativa nozione di clausola produttiva, è dovuta a Bachmair e Ganzinger [11], [12]. Esistono versioni lievemente differenti del calcolo $\mathcal{S}$ in cui ad esempio la Regola di Fattorizzazione per l'Uguaglianza è rimpiazzata da una coppia di regole (la Regola di Fattorizzazione Destra[44] e la Regola di Paramodulazione-con-Immersione), cfr. [72], [13]. Per la completezza refutazionale e la teoria delle ridondanze delle versioni 'basiche' del calcolo, si vedano [68], [69], [17],[18],[70].

La decidibilità della consistenza di un insieme finito di letterali ground (cfr. l'Esempio 9) fu osservata, all'interno di problemi più generali, già da Ackermann [1]; algoritmi veloci per la 'congruence closure' vengono forniti in [34], [65]. La terminazione del calcolo $\mathcal{S}$ nella saturazione di insiemi finiti di letterali ground si può estendere al caso in cui vengano inserite anche delle clausole non ground che assiomatizzino teorie comunemente utilizzate nella verifica del software (quali le teorie delle liste, degli arrays, ecc.), cfr. [2].

---

[42] La Regola di Paramodulazione (Destra o Sinistra) si ottiene dalla Regola di Sovrapposizione (Destra o risp. Sinistra) semplicemente ignorandone i vincoli.

[43] Si ha paramodulazione sulle variabili quando non si tiene conto della condizione '$s_{|p}$ non è una variabile' della Tabella 4.1.

[44] La sola Regola di Fattorizzazione Destra è sufficiente per la completezza refutazionale (si veda [14]), ma il calcolo che ne risulta (simile al calcolo originariamente proposto in [89]) è incompatibile per esempio con l'eliminazione di tautologie.

Per una discussione accurata relativa all'implementazione del processo di saturazione con l'utilizzo dell'algoritmo della clausola data, si veda [87].

Una delle frontiere della ricerca attuale nella dimostrazione automatica consiste nell'integrare all'interno di sistemi 'generalisti' conoscenze relative a teorie particolari, di cui siano note caratteristiche specifiche o anche procedure di decisione relativamente efficienti. Riprendendo la terminologia di [40], vi sono due diverse prospettive per ottenere tale integrazione.

Da un lato, si possono usare le conoscenze relative alle teorie particolari come *scatole bianche*, intergrandole all'interno di regole di inferenza individuali del sistema deduttivo generale. Rientrano in questa prospettiva i numerosi studi (citiamo solo [71] a titolo di esempio) sulla AC-Paramodulazione, ossia sulle modifiche da apportare alle Regola di Paramodulazione e alle sue varianti, in presenza di simboli di funzione binari che soggiacciono agli assiomi di associatività e commutatività. Ancora, si possono far rientrare in questa prospettiva studi come [15], miranti a conglobare nelle regole del 'Superposition Calculus' varie proprietà (fra cui la transitività) di simboli di relazione binari.

Dall'altro lato, si possono usare le conoscenze relative alle teorie particolari come *scatole nere*: in tal caso, le procedure di decisione particolari diventano dei moduli in sè compiuti che vengono richiamati, qualora sia necessario, dal sistema deduttivo generale (si veda a titolo esemplificativo la 'risoluzione modulo una teoria' di [83]).[45] Per un esempio di tale approccio all'interno del 'Superposition Calculus', si vedano [84], [40]. È collegata al problema dell'integrazione fra sistemi differenti anche la questione delle condizioni di eliminabilità delle inferenze impure nel calcolo $\mathcal{S}$, per cui si vedano [43], [44], [41].

A conclusione del capitolo, pensiamo di far cosa gradita segnalando risorse di rete da cui scaricare dei dimostratori automatici che possano essere utili non solo per facilitare l'apprendimento dei contenuti del presente testo, ma anche per testare le prestazioni su problemi concreti. Il progenitore degli attuali dimostratori automatici è il sistema OTTER

http://www-unix.mcs.anl.gov/AR/otter/

mentre un buon dimostratore automatico per iniziare è il sistema tedesco SPASS, che ha il vantaggio di essere disponibile sia per piattaforme

[45] In questo ambito, si usa fare un'ulteriore suddivisione fra 'partial theory reasoning' e 'total theory reasoning'. Questa distinzione si applica anche a calcoli basati per esempio sui tableaux. Per risalire alla letteratura rilevante sull'argomento, si consulti il recente articolo [85].

UniX che per piattaforme Windows

http://spass.mpi-sb.mpg.de/

Dimostratori automatici ad alte prestazioni sono il sistema E
http://www4.informatik.tu-muenchen.de/ schulz/WORK/eprover.html
e il sistema VAMPIRE

http://www.cs.man.ac.uk/ riazanoa/Vampire/

Il sistema WALDMEISTER

http://agent.informatik.uni-kl.de/waldmeister/

è specializzato nella logica equazionale, mentre il sistema SATURATE (scritto in Prolog) ha il merito di implementare in via sperimentale vari suggerimenti addizionali emersi dalla letteratura recente

http://www.mpi-sb.mpg.de/SATURATE/Saturate.html

Un'ampia lista dei sistemi disponibili (che include anche sistemi atti a trattare logiche di ordine superiore) si può trovare all'indirizzo

http://www-formal.stanford.edu/clt/ARS/systems.html

Segnaliamo infine la pagina web

http://www-formal.stanford.edu/clt/ARS/ars-db.html

che contiene informazioni utili per bibliografie e contatti con persone, gruppi di ricerca e associazioni che operano nel settore a livello internazionale.

Capitolo 5

Problemi della parola

Per il resto del presente testo ci occuperemo del caso particolare della *logica equazionale*: nella logica equazionale non vi sono predicati oltre all'identità e le clausole sono tutte *unitarie*, ossia sono composte sempre da un unico letterale. Il *problema della parola* è il tema centrale della logica equazionale e il metodo classico per affrontarlo è dato dall'*algoritmo di Knuth-Bendix* e dai suoi raffinamenti (come il completamento ordinato). In questo capitolo, ricaveremo alcuni importanti risultati tradizionali (tra cui il Teorema delle Coppie Critiche) come Corollari immediati della completezza refutazionale del calcolo $\mathcal{S}$; vedremo anche la procedura di completamento come un'istanza del metodo di saturazione esaminato nel paragrafo 4.6.

5.1 Sistemi di riscrittura

Sia $\mathcal{L}$ un linguaggio senza simboli di predicato diversi dall'uguaglianza e sia E una *$\mathcal{L}$-teoria equazionale*, ossia un insieme di $\mathcal{L}$-equazioni (che implicitamente supponiamo universalmente quantificate). Il *problema della parola (uniforme)* per E è il problema di decidere la relazione

$$E \models s = t$$

ossia di sapere se $s = t$ vale in tutti i modelli di E[1] (ad esempio, se E è la teoria dei gruppi, sappiamo dall'algebra elementare che $E \models (xy)^{-1} = y^{-1}x^{-1}$, mentre $E \not\models (xy)^{-1} = x^{-1}y^{-1}$).

[1] Anche questo problema, nalla sua formulazione generale, è indecidibile.

Definizione 5.1.1 *Sia E una $\mathcal{L}$-teoria equazionale e siano s, t termini di $\mathcal{L}$. La relazione (di 'riduzione') $s \underset{E}{\longrightarrow} t$ vale se e solo se esiste una coppia di termini (l, r) tale che $l = r \in E$, esiste $p \in Pos(s)$ ed esiste una sostituzione σ tale che*

$$\begin{aligned} s_{|p} &\equiv l\sigma \\ t &\equiv s\left[r\sigma\right]_p. \end{aligned}$$

In breve, il sottotermine di s nella posizione p è del tipo $l\sigma$ e viene rimpiazzato con il termine $r\sigma$, in virtù dell'identità $l = r$. Scriviamo talvolta $s \underset{E}{\overset{p}{\longrightarrow}} t$, per indicare anche la posizione p in cui avviene la riduzione. Si noti che la sostituzione σ che compare nella Definizione 5.1.1 altri non è che il matcher di l a $s_{|p}$.

ESEMPIO 1. Sia E dato dalle seguenti identità:

$$\begin{aligned} f\left(x, f\left(y, z\right)\right) &= f\left(f\left(x, y\right), z\right) \\ f\left(e, x\right) &= x \\ f\left(i\left(x\right), x\right) &= e \end{aligned}$$

Allora sono possibili le seguenti riscritture, a partire dal termine $f\left(i\left(e\right), f\left(e, e\right)\right)$:

$$f\left(i\left(e\right), f\left(e, e\right)\right) \underset{E}{\overset{\epsilon}{\longrightarrow}} f\left(f\left(i\left(e\right), e\right), e\right) \underset{E}{\overset{1}{\longrightarrow}} f\left(e, e\right) \underset{E}{\overset{\epsilon}{\longrightarrow}} e.$$

Dal termine e non è possibile più alcuna riduzione.

Definizione 5.1.2 *Con $\underset{E}{\overset{*}{\leftrightarrow}}$ indichiamo la più piccola relazione di equivalenza contenente $\underset{E}{\longrightarrow}$, ossia $s \underset{E}{\overset{*}{\leftrightarrow}} t$ vale se e solo se esistono $n \geq 0$ e $s \equiv s_0, \ldots, s_n \equiv t$ tali che per ogni $i = 0, \ldots, n-1$*

$$s_i \underset{E}{\longrightarrow} s_{i+1}$$

oppure

$$s_{i+1} \underset{E}{\longrightarrow} s_i.$$

Dunque, $s_0 \underset{E}{\overset{*}{\leftrightarrow}} s_n$ significa che esiste un cammino di riduzioni del tipo

$$(*) \qquad s_0 \underset{E}{\longrightarrow} s_1 \underset{E}{\longleftarrow} s_2 \underset{E}{\longleftarrow} s_3 \underset{E}{\longrightarrow} s_4 \cdots s_{n-2} \underset{E}{\longrightarrow} s_{n-1} \underset{E}{\longleftarrow} s_n$$

in cui sono mescolati modo arbitrario passi del tipo $\underset{E}{\longrightarrow}$ e del tipo $\underset{E}{\longleftarrow}$.

Poste queste definizioni, abbiamo il seguente Teorema di facile verifica:

Teorema 5.1.3 *$E \models s = t$ vale se e solo se vale $s \overset{*}{\underset{E}{\leftrightarrow}} t$.*

Dim. Diamo solo una traccia della prova. Da un lato è ovvio che $s \overset{*}{\underset{E}{\leftrightarrow}} t$ implica $E \models s = t$. Dall'altro lato, si osservi che la relazione $\overset{*}{\underset{E}{\leftrightarrow}}$ è una congruenza sull'insieme $T_{\mathcal{L}}$ dei termini del linguaggio, per cui si può costruire una $\mathcal{L}$-struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ prendendo $\mathbf{A}$ uguale a $T_{\mathcal{L}}/ \overset{*}{\underset{E}{\leftrightarrow}}$ e definendo, per ogni simbolo di funzione n-ario f,

$$\mathcal{I}(f)([t_1], \ldots, [t_n]) := [f(t_1, \ldots, t_n)].$$

Poste tali definizioni,[2] è facile provare per induzione che per ogni $a_1 = [u_1], \ldots, a_n = [u_n] \in \mathbf{A}$ e per ogni termine $t(x_1, \ldots, x_n)$, si ha $\mathcal{I}(t(\bar{a}_1, \ldots, \bar{a}_n)) = [t(u_1, \ldots, u_n)]$.

Quindi, risulta $\mathcal{A} \models s = t$ sse per ogni sostituzione σ si ha $[s\sigma] = [t\sigma]$[3] sse per ogni σ si ha che $s\sigma \overset{*}{\underset{E}{\leftrightarrow}} t\sigma$ sse vale $s \overset{*}{\underset{E}{\leftrightarrow}} t$ (perchè la relazione $\overset{*}{\underset{E}{\leftrightarrow}}$ è stabile per sostituzioni). In conclusione, $\mathcal{A}$ è modello di E e da $E \models s = t$ segue $\mathcal{A} \models s = t$ e quindi $s \overset{*}{\underset{E}{\leftrightarrow}} t$. ⊣

L'approccio mediante riscrittura diventa efficace quando (i) è sempre possibile sostituire una sequenza $\overset{*}{\underset{E}{\leftrightarrow}}$ che, come abbiamo visto, contiene passi del tipo $\underset{E}{\longleftarrow}$ e del tipo $\underset{E}{\longrightarrow}$ (mescolati fra loro senza alcun criterio predefinito), con una sequenza ('a valle') del tipo

$$s_0 \underset{E}{\longrightarrow} s_1 \underset{E}{\longrightarrow} s_2 \cdots \underset{E}{\longrightarrow} s_i \underset{E}{\longleftarrow} s_{i+1} \cdots s_{n-1} \underset{E}{\longleftarrow} s_n$$

e quando (ii) si riescono ad evitare sequenze infinite di riduzione. In tal caso, il problema della parola si risolve nel seguente modo: al fine di confrontare due termini t ed s si riscrive ciascuno di essi finché non

[2] La $\mathcal{L}$-struttura che abbiamo definito altri non è che la E-algebra libera sui generatori liberi $[x]$, dove x è una variabile.

[3] In dettaglio, vale $\mathcal{A} \models s(x_1, \ldots, x_n) = t(x_1, \ldots, x_n)$ sse per ogni $a_1, \ldots, a_n \in \mathbf{A}$ vale $\mathcal{A} \models s(\bar{a}_1, \ldots, \bar{a}_n) = t(\bar{a}_1, \ldots, \bar{a}_n)$ sse per ogni $a_1, \ldots, a_n \in \mathbf{A}$ vale $\mathcal{I}(s(\bar{a}_1, \ldots, \bar{a}_n)) = \mathcal{I}(t(\bar{a}_1, \ldots, \bar{a}_n))$ sse per ogni $u_1, \ldots, u_n \in T_{\mathcal{L}}$ vale $[s(u_1, \ldots, u_n)] = [t(u_1, \ldots, u_n)]$ sse per ogni sostituzione σ vale $[u\sigma] = [t\sigma]$.

è più possibile, e si confrontano i risultati ottenuti per vedere se sono uguali. Se lo sono, allora vale $E \models t = s$, altrimenti no. Ciò riflette la tecnica che abbiamo imparato, fin dalle scuole medie, per verificare un'identità algebrica: si semplificano entrambi i membri e si guarda se diventano uguali fra loro. Dunque, le questioni della confluenza (come evitare i picchi $\underset{E}{\longleftarrow} \circ \underset{E}{\longrightarrow}$ occorrenti in $(*)$) e della terminazione (come evitare sequenze infinite di riduzione) risultano cruciali per ottenere una procedura finitaria che decida quando due termini sono equivalenti o meno modulo una data teoria equazionale.

Definizione 5.1.4 *Una regola di riscrittura è una coppia di termini (l, r), scritta $l \to r$, tale che $Var(l) \supseteq Var(r)$ e tale che l non sia una variabile.*

Le due limitazioni della definizione precedente servono ad evitare evidenti casi di non terminazione.[4]

Una regola di riscrittura $l \to r$ ha lo stesso significato logico dell'equazione $l = r$, ma scrivendola nella forma $l \to r$ ne sottolineiamo il *significato operativo*: $l \to r$ viene usata per sostituire (all'interno di un termine che si vuole semplificare) un'istanza di l con la corrispondente istanza di r. Lo scopo di questo e del prossimo paragrafo sarà di manipolare un insieme di equazioni E in modo da ottenere un insieme di regole di riscrittura (equivalenti a E dal punto di vista logico) con le proprietà volute atte ad eliminare picchi e sequenze infinite di riduzioni.

Definizione 5.1.5 *Un* sistema di riscrittura *(TRS) R è una $\mathcal{L}$-teoria equazionale, i cui assiomi (visti come coppie ordinate) siano tutti regole di riscrittura.*

Dato un sistema di riscrittura R, è chiaro che l'insieme dei termini di un linguaggio $\mathcal{L}$ è un sistema di riscrittura astratto (che chiamiamo ancora R per semplicità) rispetto alla relazione $\underset{R}{\longrightarrow}$ introdotta nella Definizione 5.1.1; possiamo quindi liberamente utilizzare nozioni e risultati già introdotti per i sistemi di riscrittura astratti nel paragrafo 2.4 (in particolare faremo spesso uso delle Proposizioni 2.4.7, 2.4.10 e del Lemma 2.4.12).

L'utilizzo degli ordini di riduzione nella teoria dei sistemi di riscrittura è giustificato dalla seguente:

[4] Servono anche ad escludere assiomi del tipo $x = t$ (dove t è ground): si noti che ogni equazione è conseguenza logica di tali assiomi banali.

Proposizione 5.1.6 *Un sistema di riscrittura R è terminante se e solo se esiste un ordine di riduzione $>$ per cui valga $l > r$ per ogni regola $l \to r$ di R.*

Dim. Se esiste un ordine di riduzione con le caratteristiche indicate, dalle proprietà degli ordini di riduzione segue che vale $s > t$ tutte le volte che vale $s \underset{R}{\longrightarrow} t$. Per il viceversa, basta osservare che, se R è terminante, allora $\underset{R}{\longrightarrow}^{+}$ è un ordine di riduzione. ⊣

Quindi, per assicurarci la terminazione di un sistema di riscrittura R basta verificare che vale $l > r$ per ogni regola $l \to r$ di R, utilizzando uno qualsiasi degli ordini di riduzione che abbiamo incontrato nel paragrafo 3.2.

Affrontiamo ora la questione della confluenza di un sistema di riscrittura. Ricordiamo che, per il lemma di Newman 2.4.12, siccome abbiamo già a disposizione dei buoni metodi per la terminazione, possiamo concentrarci sulla confluenza locale.

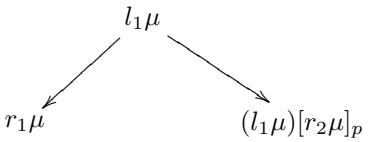
Sia R un sistema di riscrittura; la confluenza locale richiede che qualora un termine s si riscriva (in un solo passo, qui sta la differenza con la confluenza *tout court*) a due termini t_1 e t_2, questi ultimi siano riscrivibili (anche in più passi) ad uno stesso termine u. Di primo acchito, questa situazione non sembra essere controllabile in modo effettivo, invece si può stabilire che, nel caso rilevante in cui R è finito, è sufficiente testare la congiungibilità di un numero finito di coppie di termini. Tali coppie di termini sono le *coppie critiche* introdotte nella seguente:

Definizione 5.1.7 *Siano $l_1 \to r_1$ e $l_2 \to r_2$ due regole (non necessariamente distinte) di un TRS R, e supponiamo che le variabili siano state opportunamente rinominate in modo che $Var(l_1, r_1) \cap Var(l_2, r_2) = \emptyset$. Sia inoltre $p \in Pos(l_1)$ tale che $l_{1|p}$ non sia una variabile e sia μ l'unificatore più generale di $l_{1|p} \stackrel{?}{=} l_2$. Questi dati danno origine alla coppia critica*

$$\left\langle r_1\mu, (l_1\mu)\left[r_2\mu\right]_p \right\rangle.$$

Si dice in tal caso che le due regole $l_1 \to r_1$ e $l_2 \to r_2$ si sovrappongono.

Le coppie critiche corrispondono a particolari 'picchi' di riscrittura come illustrato nella seguente figura:



Qui il termine $l_1\mu$ viene riscritto a sinistra del picco in $r_1\mu$ (mediante la regola $l_1 \to r_1$ applicata in radice) e a destra del picco in $(l_1\mu)[r_2\mu]_p$ (mediante la regola $l_2 \to r_2$ applicata in posizione p).

Si noti che, se R è finito, l'insieme delle coppie critiche di R è pure finito, da cui l'importanza del seguente:[5]

Teorema 5.1.8 *Un sistema di riscrittura R è localmente confluente se e solo se le sue coppie critiche sono congiungibili. Di conseguenza, se R è terminante, R è convergente se e solo se le sue coppie critiche sono congiungibili.*

La dimostrazione di questo Teorema (che non è del tutto agevole, se svolta in modo accurato) si può trovare sui testi specializzati che trattano i Sistemi di Riscrittura: in sostanza, viene fatta un'accurata analisi per casi delle posizioni in cui avvengono le due riscritture $s \underset{R}{\longrightarrow} t_1$ e $s \underset{R}{\longrightarrow} t_2$ di un picco. Vediamo qui invece come il Teorema 5.1.8 si possa ottenere, nel caso di sistemi di riscrittura *terminanti*, come banale Corollario del Teorema 4.6.1, facendo un'ulteriore piccola assunzione (che copre tutti i casi di rilevanza pratica).

Supponiamo che R sia un sistema di riscrittura in un linguaggio $\mathcal{L}$, la cui terminazione sia garantita da un ordine di riduzione $>$ tale che valga $l > r$ per ogni regola $l \to r$ di R; in aggiunta, supponiamo che $>$ sia *totale* sui termini ground del linguaggio $\mathcal{L} + C$, dove C è un insieme infinito di nuove costanti.[6]

In queste ipotesi, proviamo direttamente che, se le coppie critiche di R sono congiungibili, allora R ha la proprietà di Church-Rosser.

[5] Nell'applicare il Teorema 5.1.8, si osservi che è inutile considerare le sovrapposizioni di una regola $l \to r$ con se stessa in radice, perchè la relativa coppia critica (r, r) è banalmente congiungibile. Invece, bisogna sempre tenere presenti possibili sovrapposizioni di una regola con se stessa in posizioni diverse dalla radice.

[6] Si ricordi quanto detto al termine del paragrafo 3.3 circa l'uso di LPO e KBO per linguaggi contenenti infinite costanti.

Dim. Siano s, t tali che $s \overset{*}{\underset{R}{\leftrightarrow}} t$ e siano $\hat{s}, \hat{t}$ delle forme normali di s e t, rispettivamente: esse esistono perchè R è terminante, si tratta di provare che $\hat{s} \equiv \hat{t}$. Rimpiazziamo le variabili presenti nei termini $\hat{s}, \hat{t}$ con delle nuove costanti di C, ottenendo rispettivamente i termini $\tilde{s}, \tilde{t}$. Per il Teorema 5.1.3, l'insieme di clausole

$$\{\tilde{s} = \tilde{t} \Rightarrow\} \cup \{\Rightarrow l = r \ : \ l \to r \in R\}$$

è inconsistente, quindi ad esso si applica il Teorema 4.6.1. Tuttavia, l'unica regola applicabile in modo non ridondante è la Regola di Risoluzione per l'Uguaglianza, per cui $\tilde{s} \equiv \tilde{t}$, come si voleva. Infatti la regola di Sovrapposizione Destra può produrre solo coppie critiche e la confluenza di tutte le coppie critiche implica automaticamente la ridondanza di tale inferenza.[7] La Sovrapposizione Sinistra non si applica, perchè $\hat{s}, \hat{t}$ (quindi anche $\tilde{s}, \tilde{t}$) sono in forma normale. ⊣

Il Teorema 5.1.8 viene utilizzato nel modo seguente. Sia data una teoria equazionale E e sia dato un ordine di riduzione $>$: supponiamo di riuscire ad *orientare* ogni equazione di E, ossia supponiamo che, per ogni $s = t \in E$, valga $s > t$ oppure $t > s$. Allora trasformiamo E nel sistema di riscrittura R che contiene, per ogni equazione $s = t \in E$, la regola $s \to t$ oppure la regola $t \to s$ (a seconda che valga $s > t$ o, rispettivamente, $t > s$). Se ora tutte le coppie critiche di R sono congiungibili (condizione che è si può testare in modo effettivo se E, e quindi anche R, è finito), sappiamo dal Teorema 5.1.8 che R è convergente. Quindi, dati due termini u e v, abbiamo che $E \models u = v$ equivale a $u \overset{*}{\underset{R}{\leftrightarrow}} v$ e a $u \downarrow v$: quest'ultima relazione è rilevabile dal confronto delle forme normali di u e v. In questa situazione, raggiungiamo così una procedura di decisione per il problema della parola in E.

ESEMPIO 2. Consideriamo la teoria

$$E \ = \ \{x + 0 = x, \ x + s(y) = s(x + y)\}.$$

Per ottenere da E un TRS, orientiamo le equazioni usando l'ordinamento LPO indotto dalla precedenza $+ >_p s$. Così ricaviamo il TRS

$$R \ = \ \{x + 0 \to x, \ x + s(y) \to s(x + y)\}.$$

[7] Si ricordi che la Demodulazione (cioè la riscrittura) si applica senza limitazioni in modalità 'forward'.

Non ci sono coppie critiche, quindi R è convergente e il problema della parola per E è decidibile mediante confronto delle R-forme normali. Così ad esempio, scopriamo che

$$E \models s(s(0)) + s(0) = s(0) + s(s(0))$$

perchè entrambi i membri hanno R-forma normale $s(s(s(0)))$. Invece

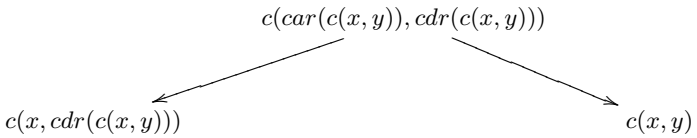
$$E \not\models x + y = y + x$$

perchè entrambi i membri sono in R-forma normale e tali R-forme normali sono distinte (in sostanza, abbiamo appurato che gli assiomi aritmetici per la somma e il successore non garantiscono di per sè la commutatività della somma).

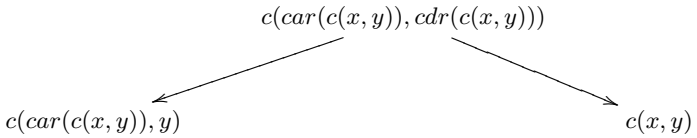
ESEMPIO 3. Consideriamo la teoria delle liste; tale teoria ha un simbolo di funzione binaria $c(ons)$ e due simboli di funzione unaria car e cdr. Gli assiomi sono

$$c(car(x), cdr(x)) = x, \quad car(c(x, y)) = x, \quad cdr(c(x, y)) = y.$$

Queste equazioni si orientano tutte da sinistra a destra secondo ogni ordine di semplificazione. Ci sono due coppie critiche; la prima è



Siccome $c(x, cdr(c(x, y))) \rightarrow c(x, y)$, tale coppia critica confluisce banalmente. Analogamente confluisce pure la seconda coppia critica



Quindi questo sistema di riscrittura è convergente.

5.2 La riscrittura ordinata

Il principale limite della teoria classica della riscrittura sta nell'impossibilità di trattare equazioni non orientabili. In effetti esistono equazioni

molto semplici e molto importanti che non sono orientabili. L'esempio più classico è dato dall'equazione

$$(C) \qquad f(x,y) \;=\; f(y,x)$$

che esprime la commutatività del simbolo binario f. Tale equazione non è orientabile secondo nessun ordine di riduzione: si ricordi infatti che un ordine di riduzione deve essere stabile per sostituzione (cioè l'applicazione di una sostituzione qualunque deve conservare l'orientamento) e la (C) è particolarmente critica in tal senso perchè la sostituzione $x \mapsto y,\ y \mapsto x$ scambia il primo membro col secondo.

La *riscrittura ordinata* dà una possibile risposta al problema. Il principio che sta alla base della riscrittura ordinata è che se un'equazione non è orientabile, sicuramente lo sono tutte le sue istanze ground, qualora si sia scelto un ordine di riduzione *totale sui termini ground* (come gli ordinamenti LPO e KBO indotti da una relazione di precedenza totale sui simboli del linguaggio). Nella riscrittura ordinata, il problema della parola viene trattato nella forma $E \models\ s = t$, dove s e t sono termini ground di una segnatura espansa con nuove costanti: ciò non lede in generalità, in quanto

$$(*) \qquad E \models\ s\,(\underline{x}) = t\,(\underline{x}) \quad \text{equivale a} \quad E \models\ s\,(\underline{c}) = t\,(\underline{c})\,,$$

se $\underline{x} = x_1, \ldots, x_n$ e $\underline{c} = c_1, \ldots, c_n$ sono delle nuove costanti distinte.

Fissiamo dunque per il resto di questo paragrafo *un linguaggio* $\mathcal{L}$ *ed una sua estensione* $\mathcal{L}_C$ che comprenda un'infinità numerabile di nuovi termini ground $c_1, c_2, \ldots$ per cui valga la $(*)$; chiamiamo $gr(\mathcal{L}_C)$ l'insieme dei termini ground di $\mathcal{L}_C$.

La scelta più ovvia per $\mathcal{L}_C$ consiste nel prendere un insieme infinito numerabile C di nuove costanti per poi porre $\mathcal{L}_C := \mathcal{L} + C$. Esistono però anche altre alternative: ad esempio, $\mathcal{L}_C$ potrebbe essere ottenuto aggiungendo a $\mathcal{L}$ una nuova costante c_0 ed un nuovo simbolo di funzione unario f, sicchè le 'nuove costanti' sarebbero codificate dai termini $c_0, f(c_0), f(f(c_0)), \ldots$. [8] Si faccia attenzione però al fatto che alcune

[8] Nel caso in cui le 'nuove costanti' sono codificate dai termini $c_0, f(c_0), f(f(c_0)), \ldots$ (dove $f, c_0 \notin \mathcal{L}$), la $(*)$ si può verificare come segue. Assumiamo che $\underline{x} = x_0, \ldots, x_n$, che $\underline{c} = c_0, \ldots, f^{n-1}(c_0)$ e che $E \not\models\ s\,(\underline{x}) = t\,(\underline{x})$; allora l'equazione $s\,(\underline{x}) = t\,(\underline{x})$ è falsa in una $\mathcal{L}$-struttura $\mathcal{A}$ che è una E-algebra libera, relativamente all'assegnamento che manda x_i nel generatore libero g_i. Siccome i g_i sono tutti distinti (altrimenti E è la teoria banale che ha tutte le equazioni possibili come conseguenze logiche e per tale teoria banale la $(*)$ è ovvia),

delle definizioni che daremo possono essere sensibili in certe circostanze delicate a come è fatta l'estensione $\mathcal{L}_C$ scelta.

Un insieme di equazioni E è *simmetrico* sse è chiuso per rinomine di equazioni ed inoltre $l = r \in E$ implica $r = l \in E$ (per ogni l, r).

Definizione 5.2.1 *Un sistema di riscrittura ordinato consiste di:*

(i) un ordine di riduzione $>$ totale sui termini ground di $\mathcal{L}_C$;

(ii) un insieme simmetrico di equazioni E fra termini di $\mathcal{L}$.

Un sistema di riscrittura ordinato verrà indicato con $(E, >)$ (o, talvolta, in modo più completo con $(\mathcal{L}, \mathcal{L}_C, E, >)$).

Definiamo ora la relazione di riscrittura nel caso dei sistemi di riscrittura ordinati. Tale relazione consente di usare nella riscrittura tutte le *istanze orientate* delle equazioni di E.

Definizione 5.2.2 *Siano t, s termini di $\mathcal{L}_C$. Definiamo $t \xrightarrow[E]{>} s$ (abbreviato con $t \xrightarrow{>} s$) se e solo se $\exists\, p \in Pos(t)$, $\exists\, (l = r) \in E$ e $\exists\, \sigma$ tali che valgano le seguenti condizioni:*

1. $l\sigma > r\sigma$,[9]
2. $t_{|p} \equiv l\sigma$,
3. $s \equiv t\,[r\sigma]_p$.

Nella pratica, la relazione $t \xrightarrow{>} s$ (che è definita per termini arbitrari di $\mathcal{L}_C$) si usa solo per *termini ground* di $\mathcal{L}_C$.[10] Il seguente Teorema si prova in modo simile al Teorema 5.1.3:

Teorema 5.2.3 *Se s, t sono termini ground di $\mathcal{L}_C$, $E \models s = t$ vale se e solo se vale $s\ (\xleftrightarrow[E]{>})^*\ t$ (qui $(\xleftrightarrow[E]{>})^*$ è la chiusura riflessiva, simmetrica e transitiva della relazione $\xrightarrow[E]{>}$).*

---

possiamo espandere l'interpretazione $\mathcal{I}$ ponendo $\mathcal{I}(c_0) = g_0$, $\mathcal{I}(f)(g_i) = g_{i+1}$ ($\mathcal{I}(f)$ è arbitraria sugli altri elementi del supporto di $\mathcal{A}$). In questo modo, si ottiene una $\mathcal{L}_C$-struttura che falsifica l'enunciato $s(\underline{c}) = t(\underline{c})$.

[9] Diciamo in tal caso che $l\sigma \to r\sigma$ è un'istanza orientata dell'equazione $l = r$.

[10] Si osservi che, se t è ground e $t \xrightarrow{>} s$, allora anche s è ground: infatti, l'istanza orientata $l\sigma \to r\sigma$ dell'equazione di E usata nel passo di riscrittura deve essere ground ($l\sigma$ è ground perchè è un sottotermine di t e $r\sigma$ è ground perchè se no, per la proprietà di stabilità degli ordini di riduzione, non potremmo avere $l\sigma > r\sigma$).

Dim. Da un lato è ovvio che $s \ (\xleftrightarrow[E]{>})^* \ t$ implica $E \models s = t$. Dall'altro lato, la relazione $(\xleftrightarrow[E]{>})^*$ è una congruenza sull'insieme dei termini ground di $\mathcal{L}_C$, per cui si può costruire una $\mathcal{L}_C$-struttura $\mathcal{A} = \langle \mathbf{A}, \mathcal{I} \rangle$ prendendo $\mathbf{A}$ uguale a $gr(\mathcal{L}_C)/(\xleftrightarrow[E]{>})^*$ e definendo, per ogni simbolo di funzione n-aria f,

$$\mathcal{I}(f)([t_1], \ldots, [t_n]) := [f(t_1, \ldots, t_n)].$$

Siccome ogni istanza ground di un assioma di E è orientabile in un senso o nell'altro (o banalmente consiste di due termini uguali fra loro), abbiamo $\mathcal{A} \models E$. Ma allora da $E \models s = t$, segue $\mathcal{A} \models s = t$ e quindi $s \ (\xleftrightarrow[E]{>})^* \ t$. ⊣

Anche ora, l'insieme $gr(\mathcal{L}_C)$ munito della relazione $\xrightarrow{>}$ è un sistema di riscrittura astratto (automaticamente terminante); diremo quindi che $(E, >)$ è ground di Church-Rosser, ground confluente, ground localmente confluente, ecc. a seconda che $(gr(\mathcal{L}_C), \xrightarrow{>})$ sia rispettivamente di Church-Rosser, confluente, localmente confluente, ecc., come sistema di riscrittura astratto.

Giungiamo finalmente alla nozione di *coppia critica estesa*, che non è altro che il risultato dell'applicazione di una Regola di Sovrapposizione Destra a equazioni di E:

Definizione 5.2.4 *Siano* $(l = r) \in E$, $(s = t) \in E$ *con* $Var(l, r) \cap Var(s, t) = \emptyset$. *Se* $s_{|p}$ *non è una variabile e* $s_{|p}$ *è unificabile con* l *mediante unificatore più generale* μ, *la coppia*

$$\left\langle t\mu, (s\mu)\,[r\mu]_p \right\rangle$$

è detta coppia critica estesa (ottenuta per sovrapposizione di $l = r$ *su* $s = t$*),*[11] *qualora il vincolo*

$$s\mu \overset{?}{>} t\mu \ \&\ l\mu \overset{?}{>} r\mu$$

sia soddisfacibile nella segnatura $\mathcal{L}_C$.

[11] Si noti che, mancando le restrizioni imposte dalla Definizione 5.1.4, non ci si può più esimere dal considerare sovrapposizioni di una equazione con se stessa in radice: questo perchè se $l' = r'$ è una copia rinominata di $l = r$, la relativa coppia critica estesa sarà non più $r = r$, bensì $r(\underline{x}, \underline{y}) = r(\underline{x}, \underline{y}')$, dove $\underline{y} = Var(r) \setminus Var(l)$.

Per riformulare il Teorema 5.1.8 delle coppie critiche, ci serviremo della seguente nozione:

Definizione 5.2.5 *Diciamo che due $\mathcal{L}$-termini u_1, u_2 sono ground-congiungibili (in simboli $u_1 \downarrow_g u_2$) se e solo se per ogni sostituzione ground σ (a valori in $gr(\mathcal{L}_C)$) vale che $u_1\sigma \downarrow u_2\sigma$, ossia se e solo se per ogni tale σ esiste $w \in gr(\mathcal{L}_C)$ tale che:*

$$u_1\sigma \overset{> \, *}{\longrightarrow} w \overset{* \, <}{\longleftarrow} u_2\sigma.$$

Teorema 5.2.6 *$(E, >)$ ha la proprietà di Church-Rosser ground se e solo se tutte le coppie critiche estese di $(E, >)$ sono ground-congiungibili.*

Dim. La proprietà di Church-Rosser ground implica banalmente la ground congiungibilità di tutte le coppie critiche estese: le istanze ground di queste ultime infatti sono conseguenze logiche di E e quindi si possono applicare il Teorema 5.2.3 e la Definizione 2.4.5. Viceversa, supponiamo che valga la ground congiungibilità delle coppie critiche estese e proviamo la condizione di Church-Rosser ground. Siano s, t termini ground di $\mathcal{L}_C$ tali che valga $s \ (\underset{E}{\overset{>}{\longleftrightarrow}})^* \ t$ e siano $\tilde{s}, \tilde{t}$ loro rispettive forme normali (tali forme normali esistono perchè la terminazione è automaticamente garantita nella riscrittura ordinata). Come nella dimostrazione del Teorema 5.1.8, basta osservare che all'insieme inconsistente di clausole

$$\{\tilde{s} = \tilde{t} \Rightarrow\} \cup \{\Rightarrow l = r \ : \ l = r \in E\}$$

solo la Regola di Risoluzione per l'Uguaglianza si applica in modo non ridondante, per cui $\tilde{s} \equiv \tilde{t}$.

Infatti, la Regola di Sovrapposizione Sinistra non si applica perchè $\tilde{s}, \tilde{t}$ sono in forma normale. La Regola di Sovrapposizione Destra produce coppie critiche estese: sia

$$\frac{\Rightarrow l = r \qquad \Rightarrow l' = r'}{(l'\mu)[r\mu]_p = r'\mu \ \ \| \ \ VD}$$

un esempio (π) di tale Regola applicata alle nostre clausole; ricordiamo che VD è il vincolo

$$l'_{|p} \overset{?}{=} l \quad \& \quad l' \overset{?}{>} r' \quad \& \quad l \overset{?}{>} r \quad \& \quad l' = r' \overset{?}{>} l = r.$$

Sia ora

$$\frac{\Rightarrow l\sigma = r\sigma \qquad \Rightarrow l'\sigma = r'\sigma}{(l'\sigma)[r\sigma]_p = r'\sigma}$$

un'istanza ground $(\pi\sigma)$ di (π); abbiamo $l'\sigma > (l'\sigma)[r\sigma]_p$ e $l'\sigma > r'\sigma$, per cui la clausola ground $\Rightarrow l'\sigma = r'\sigma$ è maggiore di ogni clausola ground usata per normalizzare ad una clausola del tipo $\Rightarrow u = u$ (mediante la ground congiungibilità delle coppie critiche estese) la clausola $(l'\sigma)[r\sigma]_p = r'\sigma$. Ciò prova la ridondanza dell'inferenza. $\dashv$

Se un sistema di riscrittura ordinato $(E, >)$ soddisfa la condizione di ground congiungibilità delle sue coppie critiche estese, il problema della parola per la teoria equazionale E si può di nuovo decidere mediante il confronto delle forme normali (basta, per la $(*)$, sostituire nell'equazione da testare le variabili con nuove costanti di $\mathcal{L}_C$). Tuttavia il problema di stabilire la ground confluenza delle coppie critiche estese può non essere banale: certamente la confluenza *tout court*[12] costituisce un test sufficiente in tal senso, ma tale test può risultare troppo debole (ritorneremo più oltre sull'argomento).

5.3 Il completamento come saturazione

Se un sistema di riscrittura ordinato non soddisfa la condizione del Teorema 5.2.6 (o anche se semplicemente non si sa o non si vuole testare tale condizione), resta comunque la risorsa di utilizzare come nuovi assiomi equazionali le coppie critiche estese che non confluiscono. Questo processo, che deve prevedere per motivi di efficienza anche dei passi intrecciati di semplificazione, è detto processo di *completamento* ordinato. Il completamento ordinato può concludersi in un numero finito di passi, dando come corollario una procedura di decisione per il pro-

[12] Si ha la confluenza *tout court* della coppia critica estesa (s, t) qualora esista un termine w di $\mathcal{L}$, tale che

$$s \overset{>}{\longrightarrow}^{*} w \overset{*}{\underset{<}{\longleftarrow}} t$$

(qui $\overset{>}{\longrightarrow}^{*}$ è la chiusura riflessiva e transitiva della relazione $\overset{>}{\rightarrow}$ della Definizione 5.2.2). Per le proprietà di stabilità degli ordini di riduzione, la confluenza *tout court* di s e t implica ovviamente $s \downarrow_g t$.

blema della parola della teoria in ingresso, oppure può divergere dando comunque un metodo di semidecisione per tale problema.[13]

Le varie forme note di completamento differiscono fra loro per dettagli, concernenti principalmente i passi di semplificazione. Il processo di saturazione (in presenza di regole di riduzione, quali la demodulazione) descritto nel Teorema 4.6.1 *descrive nei fatti una procedura di completamento.* Negli esempi che seguono, scriviamo direttamente nella forma $l \to r$ le equazioni $l = r$ che siano già orientate (ossia per le quali già valga che $l > r$).

ESEMPIO 4. Costruiamo un TRS convergente per la teoria dei gruppi. Partiamo da un insieme minimale di assiomi, ossia consideriamo la teoria G data da

$$G \;=\; \{(x * y) * z = x * (y * z),\ e * x = x,\ i(x) * x = e\}.$$

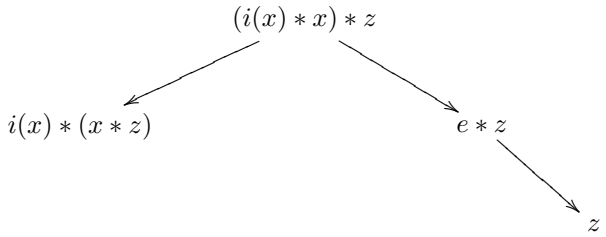
Usando un LPO indotto dalla precedenza $i > * > e$, le equazioni di G sono tutte orientabili come segue:

$$e * x \;\to\; x \tag{5.1}$$

$$i(x) * x \to e \tag{5.2}$$

$$(x * y) * z \to x * (y * z) \tag{5.3}$$

Sovrapponendo la regola (5.2) sulla regola (5.3) in posizione 1, si ottiene la coppia critica

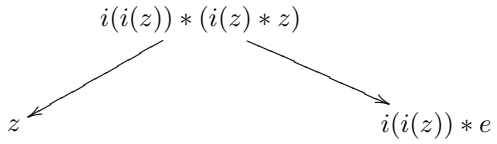


(che abbiamo già semplificato riscrivendo $e * z$ in z mediante la regola (5.1)). Tale coppia critica (normalizzata) si orienta nella nuova regola

$$i(x) * (x * z) \to z \tag{5.4}$$

[13] La procedura di semidecisione (giustificata dal Teorema 4.6.1) consiste nello svolgere in parallelo il completamento ordinato e la riscrittura dell'equazione da testare (resa ground mediante l'introduzione di nuove costanti di $\mathcal{L}_C$ al posto delle variabili).

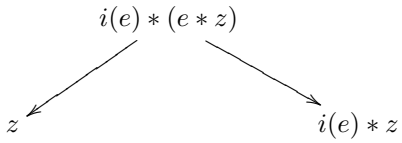
Sovrapponendo la regola (5.2) sulla regola (5.4) in posizione 2, otteniamo la coppia critica



Orientando, otteniamo la nuova regola

$$i(i(z)) * e \to z \tag{5.5}$$

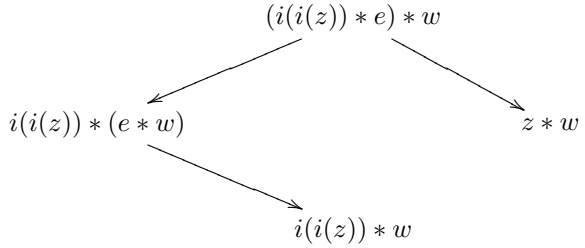
(questa regola non sarà persistente, come vedremo). Se ora sovrapponiamo la (5.1) sulla (5.4) in posizione 2, otteniamo



e quindi la nuova regola (anch'essa non sarà persistente)

$$i(e) * z \to z \tag{5.6}$$

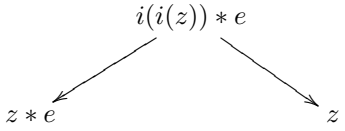
Se sovrapponiamo la (5.5) sulla (5.3) in posizione 1, otteniamo la coppia critica



e quindi la nuova regola (non persistente)

$$i(i(z)) * w \to z * w \tag{5.7}$$

Sovrapponendo in radice la (5.5) e la (5.7), si ottengono la coppia critica



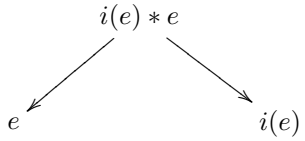
e la regola

$$z * e \to z \tag{5.8}$$

Questa regola semplifica il membro sinistro della regola (5.5) a $i(i(z))$, per cui la (5.5) viene cancellata, reintrodotta come equazione e infine reorientata con

$$i(i(z)) \to z \tag{5.9}$$

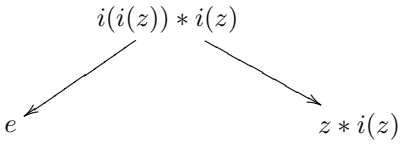
Ma allora, grazie a questa nuova regola, la (5.7) viene semplificata e infine cancellata (perchè entrambi i suoi membri si riducono a $z * w$). Sovrapponendo in radice le (5.6) e (5.8), si ottengono la coppia critica



e la regola

$$i(e) \to e \tag{5.10}$$

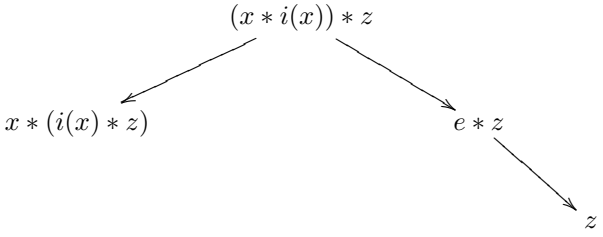
La (5.6) viene cancellata. Sovrapponendo la (5.9) sulla (5.2) in posizione 1, si ottengono la coppia critica



e la regola

$$z * i(z) \to e \tag{5.11}$$

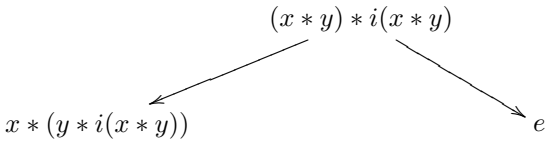
Sovrapponendo la (5.11) sulla (5.3) in posizione 1, si ottengono la coppia critica



e la regola

$$x * (i(x) * z) \to z \tag{5.12}$$

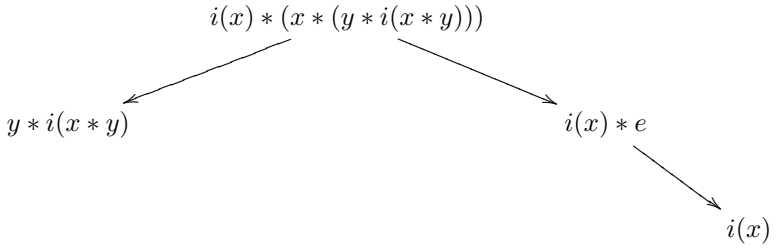
Sovrapponendo la (5.3) e la (5.11) in radice, si ottengono la coppia critica



e la regola (che non sarà persistente)

$$x * (y * i(x * y)) \to e \tag{5.13}$$

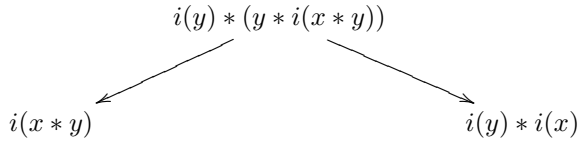
Sovrapponendo la (5.13) sulla (5.4) in posizione 2, si ottengono la coppia critica



e la regola (che non sarà persistente)

$$y * i(x * y) \to i(x) \tag{5.14}$$

La (5.13) viene cancellata. Infine sovrapponendo la (5.14) sulla (5.4) in posizione 2, si ottengono la coppia critica



e la regola

$$i(x * y) \to i(y) * i(x) \tag{5.15}$$

per effetto della quale la (5.14) viene cancellata. Qui la procedura si arresta: si può verificare che non ci sono ulteriori coppie critiche non confluenti. Riportiamo le *dieci* regole sopravvissute:

$$\begin{array}{ll} e * x \ \rightarrow \ x & i(x) * x \rightarrow e \\ (x * y) * z \rightarrow x * (y * z) & i(x) * (x * z) \rightarrow z \\ z * e \rightarrow z & i(i(z)) \rightarrow z \\ i(e) \rightarrow e & z * i(z) \rightarrow e \\ x * (i(x) * z) \rightarrow z & i(x * y) \rightarrow i(y) * i(x) \end{array}$$

Queste regole danno il famoso sistema di riscrittura convergente per la teoria dei gruppi.

Si osservi che se fossimo partiti, invece che dalla teoria G, dalla teoria (solo lievemente diversa)

$$G' \ = \ \{(x * y) * z = x * (y * z),\ e * x = x,\ x * i(x) = e\}.$$

non avremmo ottenuto il completamento di cui sopra, ma uno differente. In effetti, G' *non è* equivalente alla teoria dei gruppi, perchè non ha come conseguenza logica nè $i(x) * x = e$ nè $x * 1 = e$, come si evince facilmente dal completamento di G'. Il completamento di G' si può ottenere immettendo le equazioni di G' in un dimostratore automatico (la procedura di completamento/saturazione infatti si arresta anche in questo caso).

ESEMPIO 5. I sistemi di riscrittura sono un formalismo completo per la computabilità (esattamente come le macchine di Turing). Presentiamo qui, a titolo di esempio, un sistema di riscrittura che realizza l'algoritmo di ordinamento per inserzione. Dobbiamo ordinare in modo non crescente una lista di numeri naturali. Rappresentiamo il numero naturale n con $s^n(0)$ dove s è il simbolo unario di successore e 0 è una costante che rappresenta il numero zero. Le liste sono costruite applicando il costruttore testa/coda (che corrisponderà ad un simbolo di funzione binaria, $x : y$, scritto per comodità in notazione infissa) alla lista vuota ϵ. Così, ad esempio, la lista $[2, 1, 2]$ è rappresentata dal termine $s(s(0)) : (s(0) : (s(s(0)) : \epsilon))$. Avremo due simboli binari $max(x, y)$ e $min(x, y)$ per il calcolo del massimo e del minimo fra due numeri; in aggiunta, ci saranno un simbolo unario $sort(x)$ (che, applicato ad una lista x, denota il suo ordinamento) e un simbolo binario $insert(x, y)$ che denota il risultato dell'inserimento nel posto appropriato dell'elemento y nella lista (che si suppone già ordinata) x. Le regole

di riscrittura sono le seguenti:

$$\begin{gathered}
max(0,x)\to x\\
max(x,0)\to x\\
max(s(x),s(y))\to s(max(x,y))\\
min(0,x)\to 0\\
min(x,0)\to 0\\
min(s(x),s(y))\to s(min(x,y))\\
sort(\epsilon)\to\epsilon\\
sort(x:y)\to insert(sort(y),x)\\
insert(\epsilon,x)\to x:\epsilon\\
insert(y:z,x)\to max(x,y):insert(z,min(x,y)).
\end{gathered}$$

Questo TRS è convergente; per la terminazione si può usare un ordinamento LPO indotto dalla precedenza

$$sort > insert > : > max > min > s > \epsilon > 0.$$

Normalizzando un termine che rappresenta una lista si ottiene il termine che rappresenta l'ordinamento della lista stessa.

ESEMPIO 6. (Camaleonti). Vogliamo risolvere il seguente problema:

Nell'isola dei camaleonti sono presenti tre tipi di camaleonti: rossi, gialli e verdi. Quando due camaleonti di colore diverso si incontrano, cambiano entrambi colore e assumono il terzo colore neutro. Quando due camaleonti dello stesso colore si incontrano, cambiano entrambi colore ed assumono i due colori rimanenti (nel senso che uno dei due sceglie uno dei colori rimanenti e l'altro prende il terzo colore). Ogni camaleonte può incontrarsi con ciascuno degli altri un numero illimitato di volte. Al momento attuale sono presenti sull'isola 3 camaleonti rossi, 2 gialli e 1 verde. Si chiede se sarà possibile, in un qualche futuro prossimo o lontano, trovare sull'isola 6 camaleonti tutti rossi.

Consideriamo la seguente teoria equazionale: abbiamo un simbolo di funzione binario a e tre costanti r, g, v. Il simbolo a è soggetto alla legge associativa

$$a(a(x,y),z) \;=\; a(x,a(y,z)).$$

Le tre costanti sono soggette alle equazioni seguenti:

$$a(r,g) = a(v,v) \quad a(r,v) = a(g,g) \quad a(g,v) = a(r,r)$$
$$a(r,g) = a(g,r) \quad a(r,v) = a(v,r) \quad a(g,v) = a(v,g).$$

Il problema ha risposta positiva qualora sia possibile riscrivere il termine

$$t_1 \equiv a(r, a(r, a(r, a(g, a(g, v)))))$$

nel termine

$$t_2 \equiv a(r, a(r, a(r, a(r, a(r, r)))))$$

usando le equazioni di cui sopra come regole di riscrittura *nei due sensi* (infatti le equazioni dell'ultima riga e l'associatività di a servono a permutare l'ordine di presentazione dei camaleonti consentendo tutti gli incontri possibili e le equazioni della prima riga corrispondono ai cambiamenti di colore previsti dai vari incontri). Siccome però le equazioni di cui sopra possono essere orientate in un TRS convergente (in più modi, a seconda dell'ordinamento che si sceglie) possiamo assumere che la eventuale catena di riscritture sia in realtà un processo di normalizzazione dei due termini coinvolti. Quindi basta inserire le equazioni di cui sopra in un dimostratore automatico e aggiungere la congettura $t_1 = t_2$. Il messaggio 'proof found' corrisponde alla risposta affermativa al problema e il messaggio 'saturation found' alla risposta negativa.

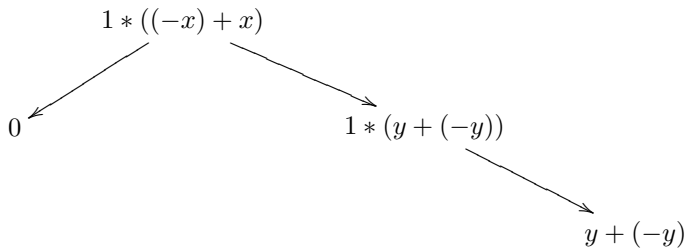
ESEMPIO 7. Questo esempio prova che per ottenere un TRS convergente nel senso classico (non ordinato), talvolta è necessario passare attraverso la riscrittura ordinata, perchè si possono incontrare equazioni non orientabili che vengono poi eliminate in passi successivi. Consideriamo la teoria

$$\begin{aligned} 1 * ((-x) + x) &= 0 \\ 1 * (x + (-x)) &= x + (-x) \\ (-x) + x &= y + (-y) \end{aligned}$$

e vediamo come procede il completamento ordinato. Usando un opportuno LPO si ottiene:

$$1 * ((-x) + x) \rightarrow 0 \tag{5.1}$$
$$1 * (x + (-x)) \rightarrow x + (-x) \tag{5.2}$$
$$(-x) + x = y + (-y) \tag{5.3}$$

(si noti che la (5.3) non è orientabile e che non ci sono coppie critiche fra le (5.1),(5.2)). Sovrapponendo la (5.3) alla (5.1) si ha la coppia critica estesa



che dà la nuova regola

$$y + (-y) \to 0. \tag{5.4}$$

Quest'ultima può essere usata per semplificare la (5.2) e ottenere

$$1 * 0 \to 0. \tag{5.5}$$

Allo stesso modo, anche la (5.3) può essere sostituita dalla seguente

$$(-x) + x \to 0. \tag{5.6}$$

A questo punto, la (5.6) e la (5.5) eliminano pure la (5.1). La procedura si arresta con le sole regole (5.4), (5.5) e (5.6), che formano un TRS convergente ed equivalente alle equazioni iniziali date.

ESEMPIO 8. La teoria dei gruppoidi entropici è data dagli assiomi

$$\begin{aligned} (x \cdot y) \cdot (z \cdot w) &= (x \cdot z) \cdot (y \cdot w) \\ (x \cdot y) \cdot x &= x \end{aligned}$$

La prima equazione non è orientabile, perché la sostituzione

$$\begin{cases} y \longmapsto z \\ z \longmapsto y \end{cases}$$

manda il primo membro nel secondo e viceversa. La seconda equazione si orienta con

$$(x \cdot y) \cdot x \longrightarrow x.$$

Si può verificare attraverso un dimostratore automatico che il completamento ordinato si arresta e dà il seguente sistema ground convergente:

$$\begin{aligned}(x \cdot y) \cdot z &= (x \cdot w) \cdot z \\ (x \cdot y) \cdot x &\longrightarrow x \\ x \cdot (y \cdot z) &\longrightarrow x \cdot z \\ ((x \cdot y) \cdot z) \cdot w &\longrightarrow x \cdot w.\end{aligned}$$

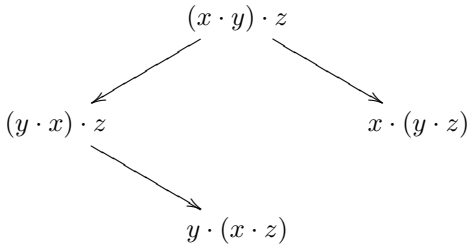
ESEMPIO 9. (Semigruppi commutativi). Sia data la seguente teoria:

$$\begin{aligned}(x \cdot y) \cdot z &= x \cdot (y \cdot z) \\ x \cdot y &= y \cdot x.\end{aligned}$$

Orientando dove possibile, abbiamo:

$$\begin{aligned}(x \cdot y) \cdot z &\longrightarrow x \cdot (y \cdot z) && (5.7)\\ x \cdot y &= y \cdot x. && (5.8)\end{aligned}$$

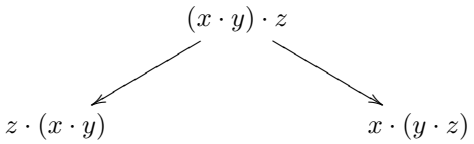
La coppia critica estesa (ottenuta sovrapponendo la (5.7) e la (5.8))



dà una nuova equazione non orientabile, che noi aggiungiamo a quelle date, ottenendo

$$\begin{aligned}(x \cdot y) \cdot z &\longrightarrow x \cdot (y \cdot z) && (5.9)\\ x \cdot y &= y \cdot x && (5.10)\\ y \cdot (x \cdot z) &= x \cdot (y \cdot z)\,. && (5.11)\end{aligned}$$

Questo sistema è ground Church-Rosser, ma la procedura non si arresta e genera la coppia critica estesa



che non va aggiunta al sistema, perché è ground convergente. Bisogna infatti provare che vale

$$u \cdot (s \cdot t) \downarrow s \cdot (t \cdot u)\,,$$

per ogni s, t, u termini ground. Si procede per casi. Ne mostriamo uno: siano s, t, u tali che $s > u > t$. Allora

$$\begin{array}{lll} u \cdot (s \cdot t) \longrightarrow & u \cdot (t \cdot s) \longrightarrow & t \cdot (u \cdot s) \\ s \cdot (t \cdot u) \longrightarrow & t \cdot (s \cdot u) \longrightarrow & t \cdot (u \cdot s)\,. \end{array}$$

Vanno testati tutti i casi restanti. Siccome tutti i test sono positivi, è dimostrata la ground confluenza. Più in generale, si verifica che il sistema di riscrittura ordinato

$$\begin{array}{rcl} (x \cdot y) \cdot z & \longrightarrow & x \cdot (y \cdot z) \\ x \cdot y & = & y \cdot x \\ y \cdot (x \cdot z) & = & x \cdot (y \cdot z)\,. \end{array}$$

che abbiamo trovato è ground Church-Rosser, mediante la ground congiungibilità di tutte le sue coppie critiche estese.[14] Naturalmente, solo i dimostratori automatici che implementano opportuni (costosi) test di confluenza ground saranno in grado di rilevarlo; gli altri, invece, proseguiranno con il completamento ordinato, divergendo (ma con buone possibibilità di farcela comunque, qualora il completamento ordinato sia usato come procedura di semidecisione rispetto al problema di verificare che un'equazione *specifica* è conseguenza logica della teoria che stiamo analizzando).

Il metodo di 'analisi per casi' dell'esempio precedente, una volta formalizzato per bene, dà un test di ground-congiungibilità che è utile

[14] Tale sistema di riscrittura ordinato simula, nel processo di normalizzazione di termini ground, l'algoritmo di 'bubble sort'.

in molti casi;[15] tuttavia il successo di tale test è una condizione sufficiente, ma non necessaria per la ground congiungibilità delle coppie critiche estese di un sistema di riscrittura ordinato. Nel prossimo paragrafo vedremo un test più potente, in grado di fornire una condizione necessaria e sufficiente.

Concludiamo con un esempio che prova che la scelta di $\mathcal{L}_C$ e della relativa estensione dell'ordine di riduzione impiegato può essere determinante per la ground convergenza.

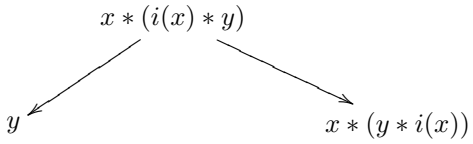
ESEMPIO 10. (Gruppi Abeliani). Si consideri il seguente sistema di riscrittura ordinato per i gruppi abeliani:

$$\begin{array}{ll} e * x \ \rightarrow \ x & x * i(x) \rightarrow e \\ (x * y) * z \rightarrow x * (y * z) & x * (i(x) * z) \rightarrow z \\ z * e \rightarrow z & i(i(z)) \rightarrow z \\ i(e) \rightarrow e & i(x * y) \rightarrow i(x) * i(y) \\ x * y = y * x & x * (y * z) = y * (x * z). \end{array}$$

Abbiamo $\mathcal{L} = \{e, *, i\}$ e supponiamo di usare per $\mathcal{L}$ un LPO basato sulla precedenza $e <_p * <_p i$. Sia $\mathcal{L}_C$ dato da $\{e, *, i, c_0, c_1, \dots\}$ e si estenda la precedenza con $e <_p * <_p i <_p c_0 <_p c_1 <_p \cdots$. Siccome risulta che

$$c_0 <_{lpo} i(c_0) <_{lpo} c_1 <_{lpo} i(c_1) <_{lpo} \cdots$$

non è difficile vedere (ragionanando sulle forme normali dei termini ground) che il sistema ha la proprietà di Church-Rosser ground. Tuttavia, se scegliamo una diversa estensione della precedenza come $e <_p * <_p c_0 <_p c_1 <_p \cdots <_p i$, ricaviamo ad esempio $c_0 <_{lpo} c_1 <_{lpo} i(c_0)$. Ma allora la coppia critica estesa



non è più ground confluente (ad esempio, i termini c_1 e $c_0 * (c_1 * i(c_0))$ sono entrambi in forma normale).

[15] Ad esempio, mediante tale test si può costruire un sistema di riscrittura ordinato ground convergente per la teoria degli Anelli Booleani.

5.4 Alberi di confluenza

Concludiamo il libro con un risultato avanzato e piuttosto sorprendente, che dimostra (fatte alcune piccole precisazioni) *la decidibilità della proprietà di Church-Rosser ground per sistemi di riscrittura ordinati finiti che utilizzino ordinamenti* LPO. Tale decidibilità si ottiene mediante un test completo (detto degli *alberi di confluenza*) di ground congiungibilità per le coppie critiche estese.

Per tutto il presente paragrafo fissiamo:

(a) un linguaggio finito ed una sua estensione anch'essa finita $\mathcal{L}_C$ che comprenda un nuovo simbolo di funzione unario *succ* e una nuova costante 0;

(b) una relazione di precedenza totale $>_p$ sui simboli di $\mathcal{L}_C$ per la quale si abbia $f >_p succ >_p 0$ (per ogni f diverso da $succ, 0$);

(c) un insieme finito E di equazioni, che non sia banale (ossia che non contenga nessuna equazione del tipo $x = r$ con $x \notin Var(r)$).[16]

Denotiamo con $(E, >)$ il relativo sistema di riscrittura ordinato,[17] dove $>$ è l'ordine di riduzione del tipo LPO indotto da $>_p$.

Si noti che, data la struttura di $\mathcal{L}_C$, l'algoritmo della Tabella 3.1 è in grado di testare in modo completo la soddisfacibilità in $\mathcal{L}_C$ di vincoli su termini del linguaggio $\mathcal{L}$.[18] Nel seguito però dovremo trattare vincoli su termini del linguaggio $\mathcal{L}_0 := \mathcal{L} + \{0\}$. Per ottenere un algoritmo che testa la soddisfacibilità in $\mathcal{L}_C$ di tali vincoli, è sufficiente aggiungere alla Tabella 3.1 la seguente istruzione:

$$\text{(x)} \quad \frac{V}{\text{FAIL}} \quad \text{se } 0 \stackrel{?}{>} t \in V$$

(ovviamente, alla definizione di vincolo completamente analizzato V andrà aggiunta la condizione $0 \stackrel{?}{>} t \notin V$). Con questa lieve modifica, si riescono ancora a dimostrare (senza variazioni di rilievo) la Proposizione 3.5.1 e i Lemmi 3.5.2 e 3.5.3 (che ora utilizzeremo, insieme alla relativa

[16] Se E contiene un'equazione del tipo $x = r$ con $x \notin Var(r)$, E ha come conseguenza logica ogni equazione e la si può completare (ottenendo subito la proprietà di Church-Rosser ground) aggiungendo semplicemente l'equazione $x = y$.

[17] Come si è osservato a suo tempo, i termini $succ^n(0)$ codificheranno in $\mathcal{L}_C$ infinite nuove costanti.

[18] Si veda, in particolare, la dimostrazione della Proposizione 3.5.4.

definizione di analisi completa di un vincolo soddisfacibile V). Anche la Proposizione 3.5.4[19] e il Teorema 3.5.5 non offrono difficoltà alcuna.

Un'*equazione soggetta a vincoli* è una coppia data da un'equazione fra termini di $\mathcal{L}_0$ e da un vincolo; la scriveremo nella forma

$$s = t \mid V$$

o anche nella forma $e \mid V$, qualora non ci interessi specificare i due termini che sono membri di e.

Le equazioni soggette a vincoli emergono dalla definizione di coppia critica estesa: infatti, la coppia critica estesa ottenuta sovrapponendo le equazioni $s = t$ e $l = r$ di E in posizione p (dove $s_{|p}$ non è una variabile) origina l'equazione soggetta a vincoli

$$(s\mu)[r\mu]_p = t\mu \mid (s\mu \stackrel{?}{>} t\mu \ \&\ l\mu \stackrel{?}{>} r\mu),$$

dove μ è upg di $s_{|p} \stackrel{?}{=} l$.

Definizione 5.4.1 *Un complesso di riduzione per l'equazione soggetta a vincoli $e \mid V$ è una terna*

$$(p, l = r, \theta)$$

tale che:

(i) $p \in Pos(e)$ *e* $e_{|p}$ *non è una variabile;*

(ii) $l = r \in E$*;*

(iii) θ *è una sostituzione tale che* $dom(\theta) \subseteq Var(l) \cup Var(r)$ *e tale che* $x\theta \equiv 0$ *per ogni* $x \in Var(r) \setminus Var(l)$*;*

(iv) $e_{|p} \equiv l\theta$*;*

(v) $l\theta \stackrel{?}{=} r\theta \notin V$ *e* $r\theta \stackrel{?}{>} l\theta \notin V$.

[19] L'unica piccola modifica da inserire nella dimostrazione della la Proposizione 3.5.4 è la seguente. Quando si analizza la soddisfacibilità da parte della soluzione canonica del vincolo atomico stretto $s \stackrel{?}{>} x \in U$, dove s è un termine di $\mathcal{L}_0$ che non è una variabile, occorre osservare anche che $s \not\equiv 0$ perchè $0 \stackrel{?}{>} x$ non può appartenere a U, in quanto U è completamente analizzato, secondo la nuova definizione.

Un *albero di confluenza* è un albero T i cui nodi sono etichettati con equazioni soggette a vincoli o con $\top$ e che è costruito secondo le istruzioni della Tabella 5.1. Detto in modo più esplicito: se un nodo di T è etichettato con un'equazione vincolata $e \mid V$, allora i sucessori di tale nodo devono essere etichettati come previsto da un'istruzione della Tabella 5.1 avente $e \mid V$ come premessa. Stipuliamo anche però che alle regole (i)-(ii) della Tabella 5.1 debba essere data *priorità assoluta* nella costruzione di un albero di confluenza (ossia, se una di tali regole si applica, essa deve essere applicata a preferenza delle rimanenti).[20]

Un albero di confluenza è *completo*, qualora a nessuna delle sue foglie sia applicabile un'istruzione della Tabella 5.1. Un albero di confluenza è *inizializzato all'equazione vincolata* $e \mid V$, qualora la sua radice sia etichettata proprio con $e \mid V$.

Proposizione 5.4.2 *Ogni albero di confluenza T è finito.*

Dim. T è a diramazione finita, perciò per il Lemma di König, è sufficiente verificare che T non ha rami infiniti. Sia per assurdo

$$e_1 \mid V_1, \quad e_2 \mid V_2, \quad \ldots$$

un ramo infinito di T. Per la condizione (iii) della Definizione 5.4.1 e per le regole delle Tabelle 5.1, 3.1, 2.1, le $e_i \mid V_i$ non possono contenere variabili che non siano già presenti in $e_1 \mid V_1$; siccome l'applicazione di un upg che non sia l'identità riduce sempre il numero delle variabili presenti nell'espressione a cui si applica, è chiaro che dovrà esistere un indice k tale che da k in poi solo upg identici intervengono nel ramo. Questo significa anche (per come sono fatte le istruzioni della Tabella 3.1) che si avrà $V_k \subseteq V_i \subseteq V_j$ per ogni $j \geq i \geq k$.

In ogni applicazione dell'istruzione (iv) della Tabella 5.1, chiamiamo principale il primo nodo del conseguente di tale istruzione e complementari gli altri due. Siccome E è finito, esistono solo finiti complessi di riduzione di una data equazione vincolata, per cui nella successione infinita di nodi

$$e_k \mid V_k, \quad e_{k+1} \mid V_{k+1}, \quad \ldots$$

dovranno per forza esserci infiniti nodi principali (si veda la condizione (v) della Definizione 5.4.1), siano essi

$$e_{i_1} \mid V_{i_1}, \quad e_{i_2} \mid V_{i_2}, \quad \ldots$$

[20] Ciò non è solo dovuto a ragioni di efficienza, ma è in qualche modo necessario per ottenere la proprietà di finitezza della Proposizione 5.4.2.

Tabella 5.1: Costruzione di un Albero di Confluenza

(i) $$\frac{s = s \mid V}{\top}$$

(ii) $$\frac{e \mid V}{\top}$$

se V è insoddisfacibile in $\mathcal{L}_C$;

(iii) $$\frac{e \mid V}{e\mu_1 \mid U_1 \quad \| \quad \cdots \quad \| \quad e\mu_k \mid U_k}$$

se V è soddisfacibile in $\mathcal{L}_C$, ma non è completamente analizzato e $(U_1, \mu_1), \ldots, (U_k, \mu_k)$ è un'analisi completa di V;

(iv) $$\frac{e \parallel V}{e[r\theta]_p \mid V \;\&\; l\theta \stackrel{?}{>} r\theta \quad \| \quad e \mid V \;\&\; l\theta \stackrel{?}{=} r\theta \quad \| \quad e \mid V \;\&\; r\theta \stackrel{?}{>} l\theta}$$

se $(p, l = r, \theta)$ è un complesso di riduzione di V.

Per il Teorema di Kruskal, avremo $e_{i_j} \leq e_{i_l}$ per certi $k \leq i_j < i_l$.[21] Tuttavia, il vincolo $V_{i_l} \supseteq V_{i_j}$ deve essere soddisfacibile perchè il ramo continui all'infinito (si ricordi che l'istruzione (ii) della Tabella 5.1 è prioritaria); per la σ che risolve tale vincolo, si deve avere che $e_{i_j}\sigma > e_{i_l}\sigma$: infatti ricordiamo che $i_j < i_l$, che il nodo i_j è principale e che tra i_j e i_l non sono previste applicazioni della (iii) con upg diversi dall'identità, quindi si deve avere $e_{i_j}\sigma > e_{i_j+1}\sigma \geq \cdots \geq e_{i_l}\sigma$. Tuttavia le relazioni $e_{i_j} \leq e_{i_l}$ e $e_{i_j}\sigma > e_{i_l}\sigma$ sono incompatibili. $\dashv$

Per ogni coppia critica estesa

$$C \;=\; \langle (s\mu)[r\mu]_p, t\mu \rangle$$

di E (ottenuta sovrapponendo le equazioni $s = t$ e $l = r$ in posizione p), scegliamo un albero di confluenza completo T_C inizializzato all'equazione soggetta a vincoli

$$(s\mu)[r\mu]_p = t\mu \mid (s\mu \stackrel{?}{>} t\mu \;\&\; l\mu \stackrel{?}{>} r\mu),$$

dove μ è upg di $s_{|p} \stackrel{?}{=} l$.

Proposizione 5.4.3 *Sia C una coppia critica estesa di $(E, >)$; se tutte le foglie di T_C sono marcate $\top$, allora C è ground confluente.*

Dim. Data la coppia critica estesa $C = \langle (s\mu)[r\mu]_p, t\mu \rangle$, si osservi che $((s\mu)[r\mu]_p)\sigma \downarrow t\mu\sigma$ vale per ogni σ ground che non soddisfi il vincolo $s\mu \stackrel{?}{>} t\mu \;\&\; l\mu \stackrel{?}{>} r\mu$.[22] Quindi ci basta provare più in generale che, se un albero di confluenza T ha tutte le foglie marcate con $\top$ ed è inizializzato all'equazione soggetta a vincoli $s = t \mid V$, allora *per ogni sostituzione σ (a valori in $gr(\mathcal{L}_C)$) che soddisfa V, si ha che $s\sigma \downarrow t\sigma$.* La dimostrazione è per induzione sul numero N dei nodi di T non etichettati con $\top$.

Se $N = 1$, allora a $s = t \mid V$ si applica una delle istruzioni (i)-(ii) della Tabella 5.1 e l'asserto è banale.

[21] Per ricondursi alla lettera alla formulazione che abbiamo dato del Teorema di Kruskal nel paragrafo 3.3, si può ad esempio inserire l'uguaglianza fra i simboli soggetti a precedenza, onde ordinare le equazioni come formule atomiche.

[22] Si ricordi che la σ è ground, quindi se σ non soddisfa il vincolo $s\mu \stackrel{?}{>} t\mu \;\&\; l\mu \stackrel{?}{>} r\mu$, si deve avere che $s\mu\sigma \leq t\mu\sigma$ o che $l\mu\sigma \leq r\mu\sigma$. Nel primo caso, ad esempio, si ha che $t\mu\sigma \underset{E}{\overset{>}{\longrightarrow}}{}^{*} s\mu\sigma$ e quindi $(s\mu[r\mu]_p)\sigma \underset{E}{\overset{>}{\longrightarrow}}{}^{*} s\mu\sigma$ oppure $s\mu\sigma \underset{E}{\overset{>}{\longrightarrow}}{}^{*} (s\mu[r\mu]_p)\sigma$, ecc.

Sia $N > 1$ e supponiamo che alla radice di T sia applicata l'istruzione (iii) della Tabella 5.1. Se σ soddisfa il vincolo V, allora per il Lemma 3.5.3, esistono σ' ed $i = 1, \ldots, k$ tali che $\sigma = \mu_i\sigma'$ e tali che che σ' soddisfa U_i. Per ipotesi induttiva, i termini $s\mu_i\sigma' \equiv s\sigma$ e $t\mu_i\sigma' \equiv t\sigma$ confluiscono.

Infine, supponiamo che alla radice sia applicata l'istruzione (iv) della Tabella 5.1. In tal caso la σ soddisfa il vincolo del nodo principale o di uno dei due nodi complementari. Nel secondo caso, si applica subito l'ipotesi induttiva, mentre nel primo caso (supponendo per esempio che $p \in Pos(s)$, perchè il caso $p \in Pos(t)$ è analogo), si applica ancora l'ipotesi induttiva osservando che $s\sigma \xrightarrow[E]{>} (s[r\theta]_p)\sigma \downarrow t\sigma$. $\dashv$

Prima di dimostrare il risultato principale di decidibilità della proprietà di Church-Rosser ground per $(E, >)$, ci servono alcuni Lemmi tecnici.

Se X è un insieme finito di variabili e σ è una sostituzione (a valori nei termini di $\mathcal{L}_C$), diciamo che σ è $\mathcal{L}_0(X)$-*iniettiva* sse per ogni coppia di termini s, t di $\mathcal{L}_0$ tali che $Var(s) \cup Var(t) \subseteq X$, si ha che

$$s\sigma \equiv t\sigma \quad \Rightarrow \quad s \equiv t.$$

È facile verificare (per induzione su $|s| + |t|$) che una sostituzione σ è certamente $\mathcal{L}_0(X)$-iniettiva, qualora soddisfi i seguenti due requisiti: a) per ogni $x, y \in X$, $x\sigma \equiv y\sigma \Rightarrow x \equiv y$; b) per ogni $x \in X$, $top(x\sigma) \equiv succ$.

Lemma 5.4.4 *Dati un vincolo completamente analizzato V e un insieme finito di variabili X, è possibile trovare una soluzione di V che è $\mathcal{L}_0(X)$-iniettiva.*

Dim. Consideriamo la relazione R_V che vale fra due variabili x e y di $X \cup Var(V)$ sse

$$\exists t \ (x \overset{?}{>} t \in V \ \& \ y \in Var(t));$$

la sua chiusura transitiva R_V^+ è terminante, a causa dell'assenza di cicli in V. Quindi è possibile estendere R_V^+ ad una relazione di ordine totale stretto $\succ_V$ fra le variabili di $X \cup Var(V)$. Definiamo la soluzione $\mathcal{L}_0(X)$-iniettiva cercata per induzione su $\succ_V$. Data una variabile x in tale insieme, si consideri il termine $u \equiv max_>\{t\sigma \,|\, x \overset{?}{>} t \in V\}$

(poniamo $u \equiv 0$, se in V non compaiono vincoli del tipo $x \stackrel{?}{>} t$). Sia ora $k \geq 1$ grande abbastanza per cui $succ^k(u)$ sia diverso dai termini $y\sigma$ (per $x \succ_V y$) già definiti e poniamo

$$(*) \qquad x\sigma :\equiv succ^k(u).$$

Chiaramente la σ è $\mathcal{L}_0(X)$-iniettiva per quanto osservato sopra. Per verificare che è anche soluzione di V basta ripetere il ragionamento della dimostrazione della Proposizione 3.5.4. Vediamo ad esempio il caso di vincoli del tipo $s \stackrel{?}{>} x$ dove s non è una variabile e x non occorre in s. Allora $s \not\equiv 0$ perchè i vincoli completamente analizzati non contengono disequazioni del tipo $0 \stackrel{?}{>} x$ e inoltre per ogni t tale che $x \stackrel{?}{>} t$ occorre in V, il vincolo $s \stackrel{?}{>} t$ appartiene a V ed è soddisfatto da σ per ipotesi induttiva.[23] Siccome $s \not\equiv 0$ non è una variabile, vale $top(s\sigma) >_p succ$. Per (LPO2), da questo segue che $s\sigma >_{lpo} succ^k(t\sigma)$ per ogni k e per ogni t tale che $x \stackrel{?}{>} t \in U$, quindi abbiamo anche che $s\sigma > x\sigma$ per la definizione di σ (si noti che se non c'è nessun $x \stackrel{?}{>} t \in V$, abbiamo comunque $s\sigma > x\sigma \equiv succ^k(0)$). ⊣

Usiamo la notazione $\sigma \equiv \sigma' \ [X]$ (dove σ, σ' sono sostituzioni e X è un insieme di variabili), per dire che vale $x\sigma \equiv x\sigma'$ per ogni $x \in X$.

Lemma 5.4.5 *Siano v, l degli $\mathcal{L}_0$-termini e siano σ, θ delle sostituzioni tali che $v\sigma \equiv l\theta$. Si supponga anche che, per un certo $X \supseteq Var(v)$, la σ sia $\mathcal{L}_0(X)$-iniettiva e che valga $top(x\sigma) \equiv succ$ per ogni $x \in X$. Allora esiste una sostituzione θ' tale che*

$$v \equiv l\theta' \quad e \quad \theta \equiv \theta'\sigma \ [Var(l)].$$

Dim. Per induzione su $N = |l|$. Sia $N = 1$; allora l è una costante o una variabile. Se l è una costante c, deve essere anche $v \equiv c$ per le ipotesi su σ, quindi si può prendere la sostituzione identica come θ'. Se $l \equiv x$ è una variabile, basta definire θ' mediante $\theta' : x \mapsto v$.

[23] Ricordiamo, dalla dimostrazione della Proposizione 3.5.4, che l'induzione viene fatta sui vincoli atomici stretti $s_1 \stackrel{?}{>} s_2 \in V$ rispetto alla misura di complessità data dalle coppie $\langle Var(s_1) \cup Var(s_2), |s_1| + |s_2| \rangle$ (tali coppie sono ordinate lessicograficamente e sulle prime componenti si utilizza l'estensione ai multiinsiemi della relazione R_V^+).

Sia $N > 1$; allora $l \equiv f(l_1, \ldots, l_n)$. Siccome $top(x\sigma) \equiv succ$ per ogni $x \in Var(v)$, abbiamo $v \equiv f(v_1, \ldots, v_n)$ e $v_i\sigma \equiv l_i\theta$ per ogni $i = 1, \ldots, n$. Esistono dunque delle sostituzioni θ'_i tali che $v_i \equiv l_i\theta'_i$ e $\theta \equiv \theta'_i\sigma\ [Var(l_i)]$. Basta dunque verificare che, se $x \in Var(l_i) \cap Var(l_j)$ (per $i \neq j$), abbiamo comunque $x\theta'_i \equiv x\theta'_j$. Ma se $x \in Var(l_i) \cap Var(l_j)$, risulta $x\theta'_i\sigma \equiv x\theta \equiv x\theta'_j\sigma$, quindi $x\theta'_i \equiv x\theta'_j$ per la $\mathcal{L}_0(X)$-iniettività di σ (si osservi che, siccome $x \in Var(l_i) \cap Var(l_j)$ e $v_i \equiv l_i\theta'_i$ e $v_j \equiv l_j\theta'_j$, si deve avere $Var(x\theta'_i) \cup Var(x\theta'_j) \subseteq Var(v) \subseteq X$). $\dashv$

Teorema 5.4.6 *$(E, >)$ ha la proprietà di Church-Rosser ground sse per ogni coppia critica estesa C di $(E, >)$ tutte le foglie dell'albero di confluenza T_C sono marcate con $\top$.*

Dim. Un lato è assicurato dalla Proposizione 5.4.3 e dal Teorema 5.2.6. Supponiamo quindi che esista una coppia critica C il cui albero di confluenza T_C ha una foglia marcata non con $\top$, ma con l'equazione soggetta a vincoli $s_1 = s_2 \mid V$. Siccome si ha che $E \models s_1 = s_2$ (questo fatto vale banalmente per tutte le equazioni che compaiono in T_C), basterà trovare una sostituzione σ a valori in $gr(\mathcal{L}_C)$ tale che $s_1\sigma$ e $s_2\sigma$ siano termini distinti e in forma normale (da questo risulterà che $(E, >)$ non ha la proprietà di Church-Rosser ground).

Si noti che, siccome T_C è un albero di confluenza completo, le istruzioni (i)-(ii) della Tabella 5.1 non si applicano a $s_1 = s_2 \mid V$: questo vuol dire, in particolare, che $s_1 \not\equiv s_2$ e che il vincolo V è soddisfacibile in $\mathcal{L}_C$. Facciamo due ulteriori importanti osservazioni.

(1) V è completamente analizzato, perchè l'istruzione (iii) della Tabella 5.1 non si può applicare.

(2) Siccome non si applica nemmeno la (iv), per ogni terna $(p, l = r, \theta)$ che soddisfa le condizioni (i)-(iv) della Definizione 5.4.1, si deve avere che $l\theta \stackrel{?}{=} r\theta \in V$ o che $r\theta \stackrel{?}{>} l\theta \in V$.

Sia $V_0 \subseteq V$ l'insieme dei vincoli atomici di V che o sono equazionali o sono del tipo $s \stackrel{?}{>} t$ per un t che sia sottotermine di s_1 o s_2. È immediato verificare che V_0 è ancora completamente analizzato.

Sia $X = Var(s_1) \cup Var(s_2)$; per il Lemma 5.4.4, la sostituzione σ (definita dalla $(*)$) è una soluzione $\mathcal{L}_0(X)$-iniettiva di V_0. Quindi, siccome $s_1 \not\equiv s_2$, abbiamo che $s_1\sigma \not\equiv s_2\sigma$.

Per concludere la dimostrazione, proviamo che per ogni sottotermine v che occorre in $s_1 = s_2$, il termine $v\sigma$ è in forma normale. La verifica verrà fatta per induzione sulle coppie

$$\langle Var(v), |v| \rangle$$

ordinate lessicograficamente (per la prima componente si utilizza l'estensione ai multiinsiemi di variabili della relazione terminante $R^+_{V_0}$ utilizzata nella definizione della σ, si veda la dimostrazione del Lemma 5.4.4).

Sia $v \equiv x$ una variabile; allora, per la $(*)$, $v\sigma$ è del tipo $succ^k(t\sigma)$, dove t è 0 (che non è riducibile, essendo il termine ground minimo) oppure è un termine per cui $x \stackrel{?}{>} t$ occorre in V_0. Per la definizione di V_0, t è un termine che occorre nell'equazione $s_1 = s_2$ e che è soggetto perciò all'ipotesi induttiva (abbiamo infatti $xR_{V_0}y$ per ogni $y \in Var(t)$). Quindi $t\sigma$ è in forma normale e tale è $succ^k(t\sigma)$.[24]

Se v non è una variabile, supponiamo per assurdo che $v\sigma$ sia riducibile: l'ipotesi induttiva copre il caso in cui la riduzione avvenga in una $p \in Pos(v)$, $p \neq \epsilon$ e il caso in cui la riduzione avvenga in una p per cui esiste $q \leq p$ tale che $v_{|q} \equiv x$ sia una variabile.[25] Quindi possiamo assumere che la riduzione di $v\sigma$ avvenga in radice. Avremo, per $l = r \in E$ e per una certa θ, che $v\sigma \equiv l\theta$ e che $l\theta > r\theta$. Possiamo anche assumere che valga $x\theta \equiv 0$ per ogni $x \in Var(r) \setminus Var(l)$.[26] Per il Lemma 5.4.5, esiste una sostituzione θ' tale che $v \equiv l\theta'$ e che $\theta \equiv \theta'\sigma\ [Var(l)]$. Se modifichiamo la θ' su $Var(r) \setminus Var(l)$ in modo da avere $x\theta' \equiv 0$ per ogni $x \in Var(r) \setminus Var(l)$, abbiamo

$$v \equiv l\theta' \quad \text{e} \quad \theta \equiv \theta'\sigma\ [Var(l) \cup Var(r)].$$

A questo punto però, se p è una posizione di v come sottotermine di $s_1 = s_2$, la terna $(p, l = r, \theta')$ soddisfa le condizioni (i)-(iv) della

[24] Non è possibile ridurre in radice sottotermini che cominciano con *succ*, perchè le equazioni di E non possono essere orientate con in testa una variabile. Si noti, infatti, che se $x = r \in E$, allora x occorre in r (altrimenti E è banale, cosa che abbiamo escluso all'inizio del paragrafo) e quindi $r > x$ (non può essere $r \equiv x$, perchè l'equazione $x = x$ non ha istanze orientabili, quindi non è mai utilizzabile per nessuna riscrittura).

[25] In tal caso, infatti, $q \neq \epsilon$ (perchè v non è una variabile) e quindi x, che è minore di v nell'ordinamento su cui lavoriamo induttivamente, è tale che $x\sigma$ è in forma normale. Segue, se $qr = p$, che $(v\sigma)_{|p} \equiv (v_{|q}\sigma)_{|r} \equiv x\sigma_{|r}$ è in forma normale (abbiamo usato, al solito, il Lemma 2.3.2 e la Proposizione 2.6.1).

[26] Se questo fatto non vale, basta cambiare la θ in modo che valga e la riduzione avviene comunque, perchè $r\theta$ risulta ancora minore nell'ordinamento.

Definizione 5.4.1; quindi per la (2), abbiamo che $l\theta' \stackrel{?}{=} r\theta' \in V$ o che $r\theta' \stackrel{?}{>} l\theta' \in V$. Siccome $l\theta' \equiv v$ è un sottotermine di s_1 o di s_2, si avrà che $l\theta' \stackrel{?}{=} r\theta' \in V_0$ o che $r\theta' \stackrel{?}{>} l\theta' \in V_0$. Ma la σ è soluzione del vincolo V_0, per cui risulta $r\theta'\sigma \geq l\theta'\sigma$, cioè $r\theta \geq l\theta$, assurdo. $\dashv$

Il Teorema 5.4.6 è di notevole interesse teorico e può essere utilizzato per provare in modo completamente automatico la proprietà di Church-Rosser ground per sistemi di riscrittura ordinata quali quello dell'Esempio 10 del paragrafo 5.3. Si noti che, tuttavia, il costo computazionale del test di confluenza ground suggerito dal Teorema 5.4.6 è elevato.

5.5 Nota bibliografica

Le tecniche e i risultati principali sui sistemi di riscrittura (lemma delle Coppie Critiche, procedura di Completamento, ecc.) sono dovuti a Knuth-Bendix [54]; la teoria è stata poi sistematizzata da Huet [48], [49], circa dieci anni più tardi. Tuttavia il metodo (detto metodo dei 'proof orders') oggi più utilizzato per dimostrare la completezza delle procedure di completamento[27] è ancora successivo: esso fu introdotto per la prima volta in [9] e ampiamente utilizzato in [8].

Il superamento del problema delle identità non orientabili (cui la teoria della riscrittura classica non può dar risposta) è avvenuto in due direzioni: da un lato, mediante la teoria della E-riscrittura [76], [50], [8] e dall'altro mediante la teoria della riscrittura ordinata [10], [46], [8]. All'interno di quest'ultimo approccio, il problema dei test di confluenza ground (per gli ordinamenti LPO) è stato affrontato con il metodo (incompleto) dell'analisi per casi sulle variabili in [61] e risolto poi con il metodo degli alberi di confluenza in [25].[28]

Molti dei classici problemi della parola che emergono nell'algebra computazionale (cfr. ad esempio [58]) sono in realtà dei problemi della parola *condizionali*, cioè si occupano di decidere la validità di implicazioni di equazioni. Il caso in cui anche la teoria in ingresso non sia equazionale, ma sia assiomatizzata tramite implicazioni fra equazioni è coperto dalla teoria della riscrittura condizionale, per cui si vedano ad esempio [52], [19], [32], [39].

In generale, sulla teoria dei sistemi di riscrittura esistono buoni manuali (ad esempio, [4], [8]) e lavori di rassegna che offrono una visione complessiva dello stato dell'arte, come [33], [30], [78].[29]

Diamo infine un cenno ai problemi della parola *combinati*: se il problema della parola per due teorie equazionali E_1, E_2 è decidibile,

[27] Le procedure di completamento sviluppate all'interno della teoria dei sistemi di riscrittura sono leggermente diverse da quella da noi illustrata nel paragrafo 5.3, che è basata sulla teoria delle ridondanze per il calcolo $\mathcal{S}$. La principale differenza sta nel fatto che tali procedure consentono di usare una regola $l \to r$ per demodulare la testa di un'altra regola $s \to t$, qualora si abbia $s \sqsupset l$ (la relazione $u_1 \sqsupset u_2$ vale fra due termini u_1, u_2 qualora u_2 abbia un matcher ad un sottotermine di u_1, ma non viceversa).

[28] Nell'esporre tale metodo nel paragrafo 5.4, abbiamo notevolmente rielaborato la presentazione di [25], facendo uso anche dei risultati di [67], pubblicati in tempi successivi.

[29] Da questi testi abbiamo tratto alcuni degli esempi che abbiamo presentato, in particolare: da [8] sono tratti gli Esempi 7, 8, da [33] gli Esempi 5, 6 (quest'ultimo con rimaneggiamenti) e da [4] l'Esempio 2, nonchè l'Esempio 11 del Capitolo 4.

cosa si può dire del problema della parola per la teoria unione $E_1 \cup E_2$? Si noti che, in generale, non è detto che se il problema della parola è decidibile per una teoria, la decisione si possa comunque ottenere tramite tecniche di riscrittura o che sia conveniente ottenerla così.[30] Per questo, nei problemi della parola combinati, si assume che le procedure di decisione per le teorie in ingresso siano 'scatole nere' inanalizzate. Nel caso di linguaggi disgiunti, la decidibilità si trasferisce da E_1 ed E_2 a $E_1 \cup E_2$ [77], ma nel caso generale possono affiorare problemi di indecidibilità. Per recenti risultati che danno condizioni sufficienti per la decidibilità del problema della parola in $E_1 \cup E_2$, si vedano [6], [37], [3].

[30] Tra l'altro, l'unione di sistemi di riscrittura operanti su segnature disgiunte mantiene la proprietà di confluenza [86], [53], ma non quelle della terminazione o della convergenza [4].

Bibliografia

[1] W. Ackermann, *Solvable Cases of the Decision Problem*, North-Holland, (1954).

[2] A. Armando , S. Ranise, M. Rusinowitch, *Uniform Derivation of Superposition Based Decision Procedures*, in L. Fribourg (c/di) "Proceedings of the Annual Conference on Computer Science Logic" (CSL01), Paris, France, pp. 513-527, (2001).

[3] F. Baader, S. Ghilardi, C. Tinelli, *A New Combination Procedure for the Word Problem that Generalizes Fusion Decidability Results in Modal Logics*, in "Proceedings of the Second Joint International Conference on Automated Reasoning (IJCAR 04)", Springer Lecture Notes in Artificial Intelligence, in corso di stampa. (2004).

[4] F. Baader, T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, (1998).

[5] F. Baader, W. Snyder, *Unification Theory*, A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 445-533, (2001).

[6] F. Baader, C. Tinelli, *Deciding the Word Problem in the Union of Equational Theories*, Information and Computation, 178, 2, pp.346-390 (2002).

[7] M. Baaz, U. Egly, A. Leitsch, *Normal Form Transformations*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 273-333, (2001).

[8] L. Bachmair, *Canonical Equational Proofs*, Birkhäuser, (1991).

[9] L. Bachmair, N. Dershowitz, J. Hsiang, *Orderings for Equational Proofs*, in "Proceeedings of the 1st IEEE Symposium on Logic in Computer Science", IEEE Computer Society Press, pp. 346-357, (1986).

[10] L. Bachmair, N. Dershowitz, D. A. Plaisted, *Completion without Failure*, in H. Aït-Kaci e M. Nivat (c/di) "Resolution of Equations in Algebraic Structures", vol. 2 (Rewriting Techniques), Academic Press, pp. 1-30, (1989).

[11] L. Bachmair, H. Ganzinger, *On Restrictions of Ordered Paramodulation with Simplification*, in M. Stickel (c/di) "International Conference on Automated Deduction (CADE10)", Lecture Notes in Computer Science, vol. 449, pp. 427-441, Springer-Verlag, (1990).

[12] L. Bachmair, H. Ganzinger, *Rewrite-based equational theorem proving with selection and simplification*, Journal of Logic and Computation, pp. 217-247, (1994).

[13] L. Bachmair L., H. Ganzinger, *Equational Reasoning in Saturation-Based Theorem Proving*, in Bibel L., Schmitt P.H. (c/di) "Automated Deduction - A Basis for Applications", vol. I, pp. 353-397, Kluwer (1998).

[14] L. Bachmair L., H. Ganzinger, *Strict Basic Superposition*, in C. Kirchner, H. Kirchner (c/di) "International Conference on Automated Deduction (CADE15)", Lecture Notes in Computer Science, vol. 1421, pp. 175-190, Springer-Verlag, (1998).

[15] L. Bachmair L., H. Ganzinger, *Ordered Chaining for First-Order Theories of Transitive Relations*, Journal of the ACM, 48, pp. 1007-1049, (1998).

[16] L. Bachmair, H. Ganzinger, *Resolution Theorem Proving*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 19-99, (2001).

[17] L. Bachmair, H. Ganzinger, C. Lynch, W. Snyder, *Basic Paramodulation and Superposition*, in "International Conference on Automated Deduction (CADE11)", Lecture Notes in Computer Science, vol. 607, pp. 462-476, Springer-Verlag, (1992).

[18] L. Bachmair, H. Ganzinger, C. Lynch, W. Snyder, *Basic Paramodulation*, Information and Computation, 121, pp. 172-192, (1995).

[19] J. A. Bergstra, J. W. Klop, *Conditional Rewrite Rules: Confluence and Termination*, Journal of Computer and System Sciences, 34, pp. 323-362, (1986).

[20] B. Blanchet, A. Podelski, *Verification of Cryptographic Protocols: Tagging Enforces Termination*, in A.D. Gordon (c/di) "Foundations of Software Science and Computational Structures, 6th International Conference (FOSSACS 2003)", Lecture Notes in Computer Science, vol. 2620, pp. 136-152, Springer-Verlag, (2003).

[21] D. Brand, *Proving Theorems with the Modification Method*, SIAM Journal of Computing, 4, pp. 412-430, (1975).

[22] B. Buchberger, *History and Basic Features of the Critical Pairs/Completion Procedure*, Journal of Symbolic Computation, 3, pp. 3-38, (1987).

[23] C. Chang C., R. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, (1973).

[24] H. Comon, *Solving Symbolic Ordering Constraints*, International Journal of Foundations of Computer Science, 1(4), pp. pp. 387-411, (1990).

[25] H. Comon, P. Narendran, R. Nieuwenhuis, M. Rusinowitch, *Decision Problems in Ordered Rewriting*, in "Proceeedings of the IEEE Symposium on Logic in Computer Science", IEEE Computer Society Press, pp. 276-286, (1988).

[26] H. Comon, R. Treinen, *Ordering Constraints on Trees*, in S. Tison (c/di) "Proceedings of the Colloquium on Trees in Algebra and Programming (CAAP)", Lecture Notes

in Computer Science, vol. 787, pp. 1-14, Springer-Verlag, (1994).

[27] A. Degtyarev, A. Voronkov, *Equality Reasoning in Sequent-Based Calculi*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 611-706, (2001).

[28] N. Dershowitz, *A Note on Simplification Orderings*, Information Processing Letters 9(5), pp. 212-215, (1979).

[29] N. Dershowitz, *Orderings for Term-Rewrite Systems* , Theoretical Computer Science 17, pp. 279-301, (1982).

[30] N. Dershowitz, J. P. Jouannaud, *Rewrite Systems*, in J. van Leeuwen (c/di) "Handbook of Theoretical Computer Science", vol. B (Formal Methods and Semantics), North Holland, pp.243-320, (1990).

[31] N. Dershowitz, Z. Manna, *Proving Termination with Multiset Orderings*, Communications of the ACM, 22(8), pp. 465-476 (1979).

[32] N. Dershowitz, M. Okada, G. Sivakumar, *Canonical Conditional Rewrite Systems*, in E. Lusk e R. Overbeek (c/di) "International Conference on Automated Deduction (CADE9)", Lecture Notes in Computer Science, vol. 310, pp. 538-549, Springer-Verlag, (1988).

[33] N. Dershowitz, D. A. Plaisted, *Rewriting*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 535-610, (2001).

[34] P. J. Downey, R. Sethi, R. E. Tarjan, *Variations on the Common Subexpression Problem*, Journal of the ACM, 24, pp. 758-771, (1980).

[35] C. Dwork, P. Kanellakis, J. Mitchell, *On the Sequential Nature of Unification*, Journal of Logic Programming, 1, pp. 35-50, (1984).

[36] U. Egly, T. Rath, *Practically Useful Variants of Definitional Translations to Normal Form*, Information and Computation, 162, pp.255-264, (2000).

[37] C. Fiorentini, S. Ghilardi, *Combining Word Problems through Rewriting in Categories with Products*, Theoretical Computer Science, 294, pp.103-149 (2003).

[38] M. Fitting, *First Order Logic and Automated Theorem Proving*, Springer, (1990).

[39] H. Ganzinger, *A Completion Procedure for Conditional Equations*, Journal of Symbolic Computation, 11, pp. 51-81, (1991).

[40] H. Ganzinger, T. Hillenbrand, U. Waldmann, *Superposition Modulo a Shostak Theory*, in F. Baader (c/di) "International Conference on Automated Deduction (CADE11)", Lecture Notes in Computer Science, vol. 2741, pp. 182-196, Springer-Verlag, (2003).

[41] H. Ganzinger, V. Sofronie-Stokkermans, U. Waldmann, *Modular Proof Systems for Partial Functions with Weak Equality*, in "Proceedings of the Second Joint International Conference on Automated Reasoning (IJCAR 04)", Springer Lecture Notes in Artificial Intelligence, in corso di stampa. (2004)

[42] S. Ghilardi, *Unification Through Projectivity*, Journal of Logic and Computation, 7(6), pp. 73-752, (1997).

[43] S. Ghilardi, *Quantifier Elimination and Provers' Integration*, in I. Dahn, L. Vigneron, (c/di) "Proceedings of the Workshop on First Order Theorem Proving (FTP03)", Electronic Notes in Theoretical Computer Science, 86 (1), (2003).

[44] S. Ghilardi, *Model-Theoretic Methods in Combined Constraints Satisfiability*, Journal of Automated Reasoning, in corso di stampa, (2004).

[45] J. Herbrand, *Récherches sur la Théorie de la Démonstration*, Tesi di Dottorato, Università di Parigi, (1930).

[46] J. Hsiang, M. Rusinowitch, *On Word Problems in Equational Theories*, in T. Ottmann (c/di) "International Colloquium on Automata, Languages and Programming", Lecture Notes in Computer Science, vol. 267, pp. 54-71, Springer-Verlag, (1987).

[47] J. Hsiang, M. Rusinowitch, *Proving Refutational Completeness of Theorem Proving Strategies: the Transfinite Semantic Tree Method*, Journal of the ACM, 38, pp. 559-587, (1991).

[48] G. Huet, *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*, Journal of the ACM, 27, pp. 797-821, (1980).

[49] G. Huet, *A Complete Proof of Correctness of the Knuth-Bendix Completion Procedure*, Journal of Computer and System Sciences, 23, pp. 11-21, (1981).

[50] J. P. Jouannaud, H. Kirchner, *Completion of a Set of Rules Modulo a Set of Equations*, SIAM Journal of Computing, 15, pp. 1155-1196, (1986).

[51] J. P. Jounannaud, M. Okada, *Satisfiability of Systems of Ordinal Notation with the Subterm Property is Decidable*, in J. Leach Albert, B. Monien, M. Rodriguez-Artalejo (c/di) "18th International Colloquium on Automata, Language and Programming (ICALP)", Lecture Notes in Computer Science, vol. 510, pp. 455-468, Springer-Verlag, (1991).

[52] S. Kaplan, *Conditional Rewrite Rules*, Theoretical Computer Science, 33, pp.175-193, (1974).

[53] J. W. Klop, A. Middeldorp, Y. Toyama, R. de Vrijer, *Modularity of Confluence: a Simplified Proof*, Information Processing Letters, 49, pp. 101-109, (1994).

[54] D. E. Knuth, P. B. Bendix, *Simple Word Problems in Universal Algebra*, in J. Leech (c/di) "Computational Problems in Abstract Algebra", Pergamon Press, pp. 263-297, (1970).

[55] K. Korovin, A. Voronkov, *A Decision Procedure for the Existential Theory of Term Algebras with the Knuth-Bendix*

Ordering, in "Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science", pp. 291-302, (2000).

[56] K. Korovin, A. Voronkov, *Knuth-Bendix Constraint Solving is NP-Complete*, ACM Transactions on Computational Logic, in corso di pubblicazione.

[57] J. B. Kruskal, *Well-Quasi-Ordering, the Tree Theorem, and Vazsonyi's Conjecture*, Trans. AMS, 95, pp. 210-225, (1960).

[58] P. Le Chenadec, *Canonical Forms in Finitely Presented Algebras*, Pitman, Londra, (1986).

[59] A. Leitsch, *The Resolution Calculus*, Springer-Verlag, (1997).

[60] A. Martelli, U. Montanari, *An Efficient Unification Algorithm*, ACM Trans. Programming Languages and Systems, 4(2), pp. 258-282, (1982).

[61] U. Martin, T. Nipkow, *Ordered Rewriting and Confluence*, in M. Stickel (c/di) "International Conference on Automated Deduction (CADE10)", Lecture Notes in Computer Science, vol. 449, pp. 366-380, Springer-Verlag, (1990).

[62] W. McCune, *Solution of the Robbins Problem*, Journal of Automated Reasoning, 19, pp. 263-276, (1997).

[63] A. Middeldorp, H. Zantema, *Simple Termination Revisited*, in A. Bundy (c/di) "Automated Deduction - CADE 12", Lecture Notes in Computer Science, vol. 814, pp. 451-465, Springer-Verlag, (1994).

[64] P. Narendran, M. Rusinowitch, R. Verma, *RPO Constraint Solving is in NP*, in G. Gottlob, E. Grandjean, K. Seyr (c/di) "12th International Conference of the European Association of Computer Science Logic (CSL)", Lecture Notes in Computer Science, vol. 1584, pp. 385-398, Springer-Verlag, (1998).

[65] G. Nelson, D. C. Oppen, *Fast Decision Procedures Based on Congruence Closure*, Journal of the ACM, 22, pp. 356-364, (1980).

[66] R. Nieuwhenhuis, *Simple LPO Constraint Solving Methods*, Information Processing Letters, 47, pp. 65-69, (1993).

[67] R. Nieuwhenhuis, J. M. Rivero, *Solved Forms for Path Ordering Constraints*, in P. Narendran, M. Rusinowitch (a cura di) "10th International Conference on Rewriting Techniques and Applications (RTA)", Lecture Notes in Computer Science, vol. 1631, pp. 1-15, Springer-Verlag, (1999).

[68] R. Nieuwhenhuis, A. Rubio, *Basic Superposition is Complete*, in B. Krieg-Brückner (c/di), "European Symposium on Programming", Lecture Notes in Computer Science, vol. 582, pp. 371-390, Springer-Verlag, (1992).

[69] R. Nieuwhenhuis, A. Rubio, *Theorem Proving with Ordering Constrained Clauses*, in D. Kapur (c/di) "International Conference on Automated Deduction (CADE11)", Lecture Notes in Computer Science, vol. 607, pp. 477-491, Springer-Verlag, (1992).

[70] R. Nieuwhenhuis, A. Rubio, *Theorem Proving with Ordering and Equality Constrained Clauses*, Journal of Symbolic Computation, 19(4), pp. 321-351, (1995).

[71] R. Nieuwhenhuis, A. Rubio, *Paramodulation with Built-in AC-Theories and Symbolic Constraints*, Journal of Symbolic Computation, 23, pp. 1-21, (1997).

[72] R. Nieuwhenhuis, A. Rubio, *Paramodulation Based Theorem Proving*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 371-443, (2001).

[73] A. Nonnengart, C. Weidenbach, *Computing Small Clause Normal Forms*, in A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. I, Elsevier/MIT, pp. 335-367, (2001).

[74] M. S. Paterson, M. N. Wegman, *Linear Unification*, Journal of Computer and System Sciences, 16, pp. 158-167, (1978).

[75] G. E. Peterson, *A Technique for Establishing Completeness Results in Theorem Proving with Equality*, SIAM Journal of Computing, 12, pp. 82-100, (1983).

[76] G. E. Peterson, M. E. Stickel, *Complete Sets of Reductions for Some Equational Theories*, Journal of the ACM, 28, pp. 223-264, (1981).

[77] D. Pigozzi, *The join of equational theories*, Colloq. Math., 30, pp. 15-25, (1974).

[78] D. A. Plaisted, *Equational Reasoning and Term Rewriting Systems*, in D. Gabbay, C. Hogger, J. A. Robinson, J. Sieckmannn (c/di) "Handbook of Logic in Artificial Intelligence and Logic Programming", vol I, Oxford University Press, pp. 273-364, (1993).

[79] A. Robinson, *A Machine-Oriented Logic based on the Resolution Principle*, Journal of the ACM, 12, pp. 23-41 (1965).

[80] A. Robinson, L. Wos, *Paramodulation and Theorem Proving in First-Order Theories with Equality*, Machine Intelligence, 4, pp. 135-150, (1969).

[81] J. R. Shoenfield, *Logica Matematica*, Boringhieri, (1980).

[82] J. H. Sieckmann, *Unification Theory: a Survey*, Journal of Symbolic Computation, 7, pp. 207-274, (1989).

[83] M. E. Stickel, *Automated Deduction by Theory Resolution*, Journal of Automated Reasoning, 1, pp. 333-355, (1985).

[84] J. Stuber, *Deriving Theory Superposition Calculi from Convergent Term Rewriting Systems*, in L. Bachmair (c/di) "11th International Conference on Rewriting Techniques and Applications (RTA)", Lecture Notes in Computer Science, vol. 1833, pp. 229-245, Springer-Verlag, (2000).

[85] C. Tinelli *Cooperation of Background Reasoners in Theory Reasoning by Residue Sharing*, Journal of Automated Reasoning, 30, pp.1-31, (2003).

[86] Y. Toyama, *On the Church-Rosser Property for the Direct Sum of Term Rewriting Systems*, Journal of the ACM, 34, pp. 128-143, (1987).

[87] C. Weidenbach, *Combining Superposition, Sorts and Splitting*, A. Robinson, A. Voronkov (c/di) "Handbook of Automated Reasoning", vol. II, Elsevier/MIT, pp. 1965-2013, (2001).

[88] C. Weidenbach, *Towards an Automatic Analysis of Security Protocols in First-Order Logic*, in H. Ganzinger (c/di), "International Conference on Automated Deduction (CADE16)", Lecture Notes in Arificial Intelligence, vol. 1632, pp. 378-382, Springer-Verlag, (1999).

[89] H. Zhang, D. Kapur, *First-Order Theorem Proving Using Conditional Rewrite Rules*, in E. Lusk, R. Overbeek (c/di) "International Conference on Automated Deduction (CADE9)", Lecture Notes in Computer Science, vol. 310, pp. 1-20, Springer-Verlag, (1988).