Anno accademico 1989-1990

Dottorato di ricerca in informatica

Università degli Studi di Milano e Università di Torino

(III ciclo)

Sulla dimostrazione di teoremi per completamento

Tesi della Dott.ssa **Maria Paola Bonacina**

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Department of Computer Science
State University of New York at Stony Brook

Riassunto

Questa tesi è uno studio delle procedure di completamento di tipo Knuth-Bendix per dimostrazione di teoremi, generazione di procedure di decisione, dimostrazione di teoremi induttivi e programmazione logica. La tesi offre una nuova presentazione delle procedure di completamento, dove il concetto di base è la *riduzione di prove*. Il nostro approccio estende quello di [8, 11], che era stato concepito in origine per le procedure di completamento intese come generatori di procedure di decisione.

L'interpretazione del completamento come procedura di semidecisione di teoremi è stata considerata fin dalla sua introduzione in [52], come un effetto collaterale dell'interpretazione come generatore di sistemi di riscrittura confluenti. Una procedura che garantisce che il limite della derivazione è un sistema confluente non può essere efficiente come dimostratore di teoremi, perchè la confluenza del limite richiede che tutte le coppie critiche siano prima o poi considerate durante la derivazione.

Il primo passo per affrontare questo problema è rendere l'interpretazione del completamento come procedura di semidecisione indipendente dall'interpretazione come generatore di sistemi confluenti. Per ottenere ciò, definiamo un nuovo concetto di *fairness*, che non richiede che tutte le coppie critiche siano prima o poi considerate e quindi permette piani di ricerca efficienti per la dimostrazione di teoremi. Dimostriamo che se le regole di inferenza sono refutazionalmente complete e il piano di ricerca é fair secondo la nostra definizione, una procedura di completamento è una procedura di semidecisione. Questo risultato rappresenta una nuova prova del teorema sul completamento come semidecisione in [52], ottenuta a partire da ipotesi più deboli e strettamente pertinenti alla concezione del completamento come dimostrazione automatica di teoremi.

Se il piano di ricerca è *uniformemente fair*, cioè tale che tutte le coppie critiche vengono prima o poi considerate, il limite di una derivazione è un sistema confluente, o, più in generale, una presentazione *saturata* della teoria data. Sotto condizioni aggiuntive che dipendono dalla logica, un insieme saturato è una *procedura di decisione* e la procedura di completamento è un generatore di procedure di decisione. Il processo di semidecisione e il processo di saturazione coesistono, come venne provato in [52] per il caso equazionale. Otteniamo una nuova, generale versione di questo teorema, che chiamiamo *il teorema di Knuth-Bendix-Huet generalizzato.*

Diamo inoltre una nuova presentazione della procedura *Unfailing Knuth-Bendix* [48, 14], della procedura *AC-UKB* [70, 55, 9, 1] con *Leggi di cancellazione* [49], della *S-strategy* [48] e specialmente della *Inequality Ordered Saturation strategy* [3], che non era stata mai descritta come un insieme di regole di inferenza prima d'ora.

Per l'applicazione del completamento alla confutazione di congetture induttive, mostriamo che il cosiddetto metodo di *inductionless induction* è un processo di semidecisione. Per la programmazione logica, presentiamo una semantica operazionale e denotazionale di *programmi di regole di riscrittura.* La semantica operazionale è data da una procedura di completamento chiamata *Linear Completion.*

Concludiamo indicando alcune direzioni per future ricerche, che includono un'idea di *completamento modulo un algoritmo di decisione.*

Indice

Capitolo 1

Introduzione

La procedura di completamento di Knuth-Bendix [62] computa un sistema di riscrittura, non necessariamente finito, equivalente a un dato insieme di equazioni [51]. Dati un insieme di equazioni E e un'equazione $s \simeq t$, la procedura semidecide il problema dell'appartenenza di $s \simeq t$ all'insieme dei teoremi di E, come venne osservato per la prima volta in [65, 52]. Questi risultati valgono se la procedura non fallisce per via di una equazione non orientabile. Equazioni non orientabili sono ammesse se si adotta il metodo Unfailing Knuth-Bendix [48, 14], che genera un insieme di equazioni confluente su termini ground.

A partire dal modello Knuth-Bendix, si sono progettate molte procedure di completamento, più o meno vicine alla procedura originaria. Si hanno procedure di completamento per teorie equazionali con assiomi speciali, quali associatività e commutatività, [70, 55, 9], la logica di Horn con uguaglianza [63, 36], la logica del prim'ordine [44, 45, 59, 10], la logica del prim'ordine con uguaglianza [46, 47, 49, 50, 74, 78, 15, 16], la dimostrazione di teoremi induttivi in teorie equazionali e di Horn [53, 38, 63] e la programmazione logica [29, 30, 31, 20]. Rassegne di queste procedure sono state date in [33, 34].

In questa tesi presentiamo un nuovo approccio allo studio delle procedure di completamento di tipo Knuth-Bendix e lo applichiamo alla dimostrazione di teoremi, alla generazione di procedure di decisione, alla dimostrazione di teoremi induttivi e alla programmazione logica.

Una *procedura di completamento* si compone di *regole di inferenza* e di un *piano di ricerca*. Le regole di inferenza determinano cosa può essere derivato dai dati disponibili. Il piano di ricerca sceglie ad ogni passo della derivazione quale regola di inferenza applicare a quali dati e quindi determina l'unica derivazione che la procedura computa dato un certo input.

Un processo di completamento può essere considerato a un livello molto generale come *trasformazione di problemi*. Sebbene il problema $E \models \forall \bar{x} s \simeq t$ sia semidecidibile in generale, è decidibile via riduzione se E è ground confluente. In conseguenza, la trasformazione di un insieme di equazioni E in un insieme E' ground confluente, che presenta la stessa teoria, può essere considerato come trasformazione di un problema semidecidibile in un problema decidibile. Il problema semidecidibile è dimostrare la validità di un teorema usando E, mentre il problema decidibile è dimostrare la validità di un teorema usando E'. In dimostrazione di teoremi, uno specifico problema semidecidibile $E \models \forall \bar{x} s \simeq t$, scritto $(E; \hat{s} \simeq \hat{t})$, viene trasformato finché ha la

forma $(E_k; \hat{s}_k \simeq \hat{s}_k)$, dove $E_k \models \forall \bar{x} s_k \simeq s_k$ è banalmente decidibile. Nel caso di teoremi induttivi, il problema semidecidibile è che $\forall \bar{x} s \simeq t$ *non* è un teorema induttivo di E: $E \cup \{s \simeq t\}$ viene trasformato finché il problema può essere deciso da un dato oracolo.

Studiamo queste trasformazioni al livello delle *prove*. Fin da [8, 11], il completamento di un insieme di equazioni in un sistema confluente è stato considerato come riduzione delle prove equazionali $s \leftrightarrow_E^* t$ nella presentazione originale a prove di riscrittura $s \rightarrow_{E_\infty}^* \circ \leftarrow_{E_\infty}^* t$ nel limite E_∞ della derivazione. Continuiamo ed estendiamo questo approccio alle altre applicazioni del completamento, quali la dimostrazione di teoremi e di teoremi induttivi. Questa estensione richiede nozioni di *riduzione di prove* e di *ordinamento su prove* più generali di quelle in [8, 11], dal momento che le nozioni originali di [8, 11] non tengono in considerazione il *backward reasoning*, che è necessario in dimostrazione di teoremi. Ogni passo di inferenza riduce delle prove o riduce la presentazione stessa eliminando dati *ridondanti*. Gli ordinamenti su prove vengono usati per definire tutti i concetti fondamentali della tesi.

Consideriamo innanzitutto la dimostrazione di teoremi. L'interpretazione del completamento come procedura di semidecisione per la dimostrazione di teoremi è la più importante per noi. Comparve per la prima volta nel fondamentale lavoro dove Huet provò la correttezza della procedura di Kunth-Bendix [52]. Huet dimostrò che se il piano di ricerca è fair e la derivazione non fallisce, il limite di una derivazione mediante Knuth-Bendix è un sistema di riscrittura confluente, e, in conseguenza, se un teorema $s \simeq t$ viene dato alla procedura, essa semidecide la validità di $\forall \bar{x} s \simeq t$. Chiamiamo questo teorema il *teorema di Knuth-Bendix-Huet*.

Nonostante questo risultato sia stato conosciuto per anni, l'interpretazione del completamento come procedura di semidecisione non ha ricevuto molta attenzione, essendo stata oscurata dalla interpretazione come generatore di sistemi confluenti. Quest'ultima è di gran lunga la più nota, mentre la dimostrazione di teoremi è sostanzialmente considerata un effetto collaterale della generazione di sistemi confluenti. Una tale prospettiva non è accettabile dal punto di vista della dimostrazione di teoremi, perchè una procedura che prima o poi ottiene un sistema confluente non può essere efficiente come dimostratore. Il nostro approccio al completamento è l'opposto di quello tradizionale: presentiamo le procedure di completamento innanzitutto come procedure di semidecisione, con la generazione di sistemi confluenti come uno speciale effetto collaterale.

L'interpretazione del completamento come semidecisione è resa indipendente da quella come generazione di sistemi confluenti. Dimostriamo che se le regole di inferenza sono *refutazionalmente complete* e il piano di ricerca è *fair*, una procedura di completamento è una *procedura di semidecisione*. *Completezza refutazionale* significa che per tutti gli inputs insoddisfacibili, esistono derivazioni, mediante le regole di inferenza della strategia, che hanno successo, cioè terminano segnalando l'insoddisfaciblità. *Fairness* significa che se esistono derivazioni che hanno successo, il piano di ricerca garantisce che la derivazione computata dalla procedura ha successo, ovvero tutte le inferenze necessarie a provare il goal sono prima o poi eseguite. In particolare, tutte le coppie critiche che sono necessarie a provare il goal sono prima o poi considerate. Diamo una nuova definizione di fairness per esprimere questo concetto.

Questa idea di fairness è la differenza chiave tra il completamento per la dimostrazione di teoremi e il completamento per la generazione di sistemi confluenti. Nel lavoro di Huet [52] e in tutti i seguenti [8, 11, 74, 16], la fairness di una derivazione consiste nel considerare prima o poi

tutte le coppie critiche. Battezziamo questa proprietà *fairness uniforme*, allo scopo di distinguerla dalla fairness per dimostrazione di teoremi. La fairness uniforme è necessaria affinché il limite della derivazione sia confluente, ma non è necessaria per dimostrare teoremi, perché non tutte le coppie critiche sono necessarie a dimostrare un dato teorema. Considerare tutte le coppie critiche è una fonte di inefficienza non necessaria in una derivazione per dimostrazione di teoremi. Tutte le definizioni di fairness per procedure di completamento apparse finora nella letteratura [52, 11, 74, 16] richiedono la fairness uniforme, perché non separano la dimostrazione di teoremi dalla generazione di sistemi confluenti.

In seguito, consideriamo la più classica delle interpretazioni del completamento, ovvero la generazione di sistemi confluenti, o, più in generale, di procedure di decisione. Se il piano di ricerca è uniformemente fair, il limite di una derivazione è una presentazione *saturata*. Il concetto di insieme saturato è una generalizzazione di quello di sistema confluente: una presentazione è saturata se nessuna conseguenza non banale può esserle aggiunta [63, 16]. Se una presentazione è saturata, le derivazioni da tale presentazione sono derivazioni *lineari input* [24]. Se si può garantire la terminazione delle derivazioni lineari input da un insieme saturato, un insieme saturato è una *procedura di decisione* e la procedura di completamento è un *generatore di procedure di decisione*. Condizioni per la terminazione di derivazioni lineari input sono note in logica equazionale e logica di Horn con uguaglianza [63, 16]. Le nostre definizioni di fairness uniforme, insieme saturato, derivazione lineare input ben fondata, procedura di decisione e i teoremi relativi danno una presentazione della applicazione del completamento alla generazione di procedure di decisione, che unifica tutti i principali risultati relativi a questa interpretazione del completamento in logica equazionale e di Horn con uguaglianza.

Tuttavia, lo studio separato del completamento come procedura di semidecisione e come generatore di procedure non coglie il significato complessivo del teorema di Knuth-Bendix-Huet da cui siamo partiti. In quel teorema le due interpretazioni del completamento sono connesse: se il limite di una derivazione è saturato, qualsiasi teorema può essere provato durante la derivazione, indipendentemente dal fatto che il limite sia finito o no. Perciò diamo anche un nuovo teorema di Knuth-Bendix-Huet generalizzato per mettere in relazione il processo di semidecisione e quello di saturazione.

Il nostro approccio basato sulla riduzione di prove ci ha permesso di ottenere due risultati principali. Innanzitutto abbiamo dimostrato il classico risultato di Huet sul completamento come semidecisione a partire da ipotesi più deboli e orientate esclusivamente alla dimostrazione di teoremi, ipotesi che non implicano alcuna proprietà di confluenza del limite della derivazione. Questo prova che l'interpretazione del completamento come semidecisione è indipendente dalla interpretazione come generazione di insiemi saturati e quindi non è soltanto un effetto collaterale di quest'ultima. In secondo luogo, abbiamo dato la prima generalizzazione del teorema di Knuth-Bendix-Huet oltre la logica equazionale. Il nostro teorema di Knuth-Bendix-Huet generalizzato non dipende da una logica in particolare. Il solo elemento che dipende dalla logica sono le condizioni affinché un insieme saturato sia una procedura di decisione.

La tesi è organizzata come segue. Il secondo capitolo contiene gli elementi di base del nostro approccio: *ordinamenti su prove*, *riduzione di prove*, *ridondanza* e *procedure di completamento*. Il terzo capitolo è dedicato alla dimostrazione di teoremi. Diamo la nostra definizione di *fairness* e

proviamo che una procedura di completamento, le cui regole di inferenza sono refutazionalmente complete e il cui piano di ricerca è fair, è una procedura di semidecisione. Nel quarto capitolo, l'attenzione si sposta su *fairness uniforme*, presentazioni *saturate* e la *generazione di procedure di decisione*. Il quinto capitolo contiene il teorema di Knuth-Bendix-Huet generalizzato. Nel sesto capitolo presentiamo alcune procedure di completamento per la logica equazionale: mostriamo che la procedura *Unfailing Knuth-Bendix* [48, 14] e alcune sue estensioni, quali la procedura *AC-UKB* [70, 55, 9, 1] con *Leggi di cancellazione* [49], la *S-strategy* [48] e la *Inequality Ordered Saturation strategy* [3] sono descritte elegantemente dal nostro approccio. In base alle nostre conoscenze, questa è la prima presentazione di queste estensioni della procedura UKB come insiemi di regole di inferenza. Il settimo capitolo è dedicato alla applicazione del completamento alla confutazione di congetture induttive, il cosiddetto metodo di *inductionless induction* [53]. Mostriamo che inductionless induction è una *procedura di semidecisione per confutare congetture induttive*. Nell'ottavo capitolo, consideriamo l'applicazione del completamento alla programmazione logica, dando una semantica operazionale e denotazionale di programmi a riscrittura interpretati da *Linear Completion*. Nell'ultimo capitolo diamo un sommario dei nostri risultati ed indichiamo alcune possibili direzioni per future ricerche. Queste includono l'estensione della nostra idea del completamento come trasformazione di problemi, al *completamento modulo un algoritmo di decisione*. Se un algoritmo di decisione è disponibile per una certa teoria, non è necessario spingere il completamento di un problema di dimostrazione $(S;\varphi)$ fino a uno stadio $(S_k;\varphi_k)$ dove $S_k \models \varphi_k$ è banale, come nel caso in cui φ_k è un'equazione $s \simeq s$. Il completamento termina a uno stadio k se $S_k \models \varphi_k$ è decidibile dall'algoritmo dato. Questo approccio apre la via allo studio del completamento in connessione con procedure di decisione per teorie speciali.

Capitolo 2

Procedure di completamento

Il nostro studio delle procedure di completamento è interamente basato su una nozione di *riduzione di prove*. In dimostrazione di teoremi, si vuole ridurre una singola prova, quella del teorema obbiettivo della dimostrazione: la derivazione termina con successo se l'obbiettivo è stato ridotto a un teorema banalmente vero, come $s \simeq s$, la cui prova è vuota. Nel completamento tradizionale, si vogliono ridurre tutte le prove: per esempio, nel completamento di Knuth-Bendix tutte le prove vengono ridotte a prove di riscrittura.

Allo scopo di formalizzare tutti i concetti in termini di riduzione di prove, occorre una nozione di *ordinamento su prove* ben fondato. Il nostro punto di partenza sono gli ordinamenti su prove presentati originariamente in [8, 11]. Tuttavia, gli ordinamenti su prove in [8] non si applicano a una derivazione per dimostrazione di teoremi, perché permettono di confrontare solo due prove dello stesso teorema. In dimostrazione di teoremi, l'obbiettivo viene modificato dai passi di inferenza applicati all'obbiettivo stesso. Quindi, diamo una nuova definizione di ordinamento su prove, dove prove di diversi teoremi possono essere confrontate.

In seguito caratterizziamo *regole di inferenza* e *piano di ricerca* di una *procedura di completamento*. Ad ogni stadio di una derivazione l'insieme di dati corrente è la porzione dello spazio di ricerca che è stata attualmente generata. La dimensione dello spazio di ricerca cresce esponenzialmente con la dimensione dell'input. Una strategia di dimostrazione ideale è quella che trova una prova dell'obbiettivo consumando la quantità di spazio e tempo più piccola possibile, ovvero generando la porzione dello spazio di ricerca più piccola possibile. Una strategia è tanto più efficiente quanto più si avvicina a questo ideale, su una classe ragionevolmente larga di problemi. Le regole di inferenza e il piano di ricerca contribuiscono entrambe a determinare l'efficienza della strategia.

Un piano di ricerca consiste in uno o più criteri per decidere quali dati e quale regola di inferenza selezionare per il prossimo passo. Questi criteri sono di natura euristica, in quanto in generale è impossibile determinare a priori se un dato passo ci porterà più vicino alla prova. Vedremo nel seguito che alcuni semplici criteri possono essere descritti usando ordinamenti.

Le regole di inferenza sono o regole di *espansione* o regole di *contrazione*. Le regole di espansione generano nuove formule. Una strategia di dimostrazione che includa soltanto regole di espansione è molto inefficiente, perché a ogni passo il numero delle clausole generate cresce. La

memoria disponibile viene rapidamente esaurita senza raggiungere una prova dell'obbiettivo anche nel caso di problemi relativamente semplici. Una regola di contrazione elimina una formula[1] o la sostituisce con altre più piccole: cancellando una formula fa sì che la procedura non generi tutte le formule che potrebbero essere generate da quella. Le regole di contrazione riducono la crescita dello spazio di ricerca. Il rapporto tra espansione e contrazione influisce quindi significativamente sull'efficienza del metodo.

Le regole di inferenza di una procedura di completamento hanno la proprietà di ridurre delle prove o eliminare formule *ridondanti*. Sia la riduzione di prove sia la ridondanza vengono definite in termini di ordinamenti di prove.

2.1 Definizioni preliminari

In questa sezione ricordiamo alcuni concetti e notazioni di base per dimostrazione di teoremi. La nostra presentazione è consistente con quella in [34, 35].

Dati un insieme finito F di simboli di costante e di funzione con le loro arità e un insieme numerabile X di simboli di variabile, $T(F, X)$ è l'insieme dei *termini* su F e X. Un termine è *ground* se non contiene variabili. L'insieme dei termini ground è denotato da $T(F)$.

Un termine s è un *sottotermine* di un termine t se s appare in t. Si scrive t come $c[s]$ per indicare che s è un sottotermine di t nel *contesto* c.

Assumiamo l'usuale rappresentazione dei termini come alberi finiti ordinati: una variabile x o una costante c è rappresentata da un albero con un singolo nodo etichettato da x o c rispettivamente. Un termine $f(t_1 \dots t_n)$ è rappresentato da un albero la cui radice ha etichetta f ed ha n archi uscenti ordinati, che puntano alle radici degli alberi per i termini $t_1 \dots t_n$.

Si scrive $s = t|u$ per specificare che s è un sottotermine di t alla *posizione* u, dove u è la stringa di numeri naturali che etichettano gli archi nel cammino dalla radice dell'albero di t alla radice del sottoalbero di s. Più precisamente, l'insieme $\mathcal{O}(t)$ delle posizioni in un termine $t = f(t_1 \dots t_n)$ è l'insieme delle stringhe di numeri naturali tale che $\lambda \in \mathcal{O}(t)$, dove λ è la stringa vuota, e se $u \in \mathcal{O}(t_i)$ per qualche i, $1 \leq i \leq n$, allora $i \cdot u \in \mathcal{O}(t)$. La stringa vuota λ denota la posizione della radice, cioè $t|\lambda = t$, e la stringa $i \cdot u$ denota la posizione u nell'i-esimo sottotermine di t, cioè $f(t_1 \dots t_n)|i \cdot u = t_i|u$. Due distinte posizioni u e v in $\mathcal{O}(t)$ sono *disgiunte* se nè u è un prefisso di v nè v è un prefisso di u. La notazione $s[t]_u$ rappresenta il termine ottenuto rimpiazzando $s|u$ con t.

Una *sostituzione* σ è un insieme $\{x_1 \mapsto s_1 \dots x_n \mapsto s_n\}$ tale che

- $\forall i, j$, $i \neq j$ implica $x_i \neq x_j$,
- $\forall i, j$, $x_i \notin V(s_j)$, dove $V(s_j)$ è l'insieme delle variabili nel termine s_j.

L'insieme $\{x_1 \dots x_n\}$ è detto il *dominio* della sostituzione σ: $Dom(\sigma) = \{x_1 \dots x_n\}$. L'insieme di tutte le variabili che compaiono nei termini $s_1 \dots s_n$ è detto il *rango* di σ: $Ran(\sigma) = \bigcup_{j=1}^{n} V(s_j)$.

[1] Qui e nel seguito *formula* sta per *formula chiusa*, ovvero formula dove tutte le variabili sono quantificate.

Una sostituzione σ è *ground* se $Ran(\sigma) = \emptyset$. Una sostituzione σ si applica a un termine come segue:

- $x\sigma = s$ se $x \mapsto s \in \sigma$,
- $x\sigma = x$ se $x \notin Dom(\sigma)$,
- $c\sigma = c$ se c è una costante,
- $f(t_1 \dots t_n)\sigma = f(t_1\sigma \dots t_n\sigma)$, altrimenti.

Date due sostituzioni $\sigma = \{x_1 \mapsto s_1 \dots x_n \mapsto s_n\}$ e $\rho = \{y_1 \mapsto r_1 \dots y_m \mapsto r_m\}$ tali che $Dom(\sigma) \cap Ran(\rho) = \emptyset$, la loro *composizione* è la sostituzione $\sigma\rho = \{x_1 \mapsto s_1\rho \dots x_n \mapsto s_n\rho\} \cup \{y_j \mapsto r_j | y_j \mapsto r_j \in \rho, y_j \notin Dom(\sigma)\}$. La composizione di sostituzioni è associativa, cioè $(\sigma\rho)\theta = \sigma(\rho\theta)$. La seconda condizione nella nostra definizione di sostituzione implica che $Dom(\sigma) \cap Ran(\sigma) = \emptyset$. Questa proprietà è equivalente a $\sigma\sigma = \sigma$, ovvero all'*idempotenza* della composizione.

Un *ordinamento* $\succ$ è una relazione binaria, transitiva ed irriflessiva. La transitività e l'irriflessività implicano l'asimmetria. Un ordinamento è *parziale* in generale. E' *totale* se per ogni due distinti elementi s e t nell'insieme ordinato, è $s \succ t$ o $t \succ s$. Un ordinamento è *ben fondato* se non esistono sequenze infinite $s_1 \succ s_2 \succ \dots s_n \succ \dots$.

La relazione di sottotermine definisce un ordinamento, detto l'*ordinamento per sottotermine*: $t \trianglerighteq s$ se $t = c[s]$ e $t \triangleright s$ se c è non vuoto. Un termine t è un'*istanza* di un termine s se c'è una sostituzione σ tale che $t = s\sigma$: si scrive $t \trianglerighteq s$ e la relazione $\trianglerighteq$ è detta *ordinamento per sussunzione*. La sostituzione σ è detta *matching*. Se t è un'istanza di s ed s è un'istanza di t, i due termini sono uguali a meno di una permutazione di variabili. Una sostituzione che è soltanto una permutazione di variabili è detta *ridenominazione* e i due termini s e t si dicono *varianti* l'uno dell'altro, scritto $s \stackrel{\bullet}{=} t$. Altrimenti, t è una *istanza propria* di s: si scrive $t \triangleright s$ e la relazione $\triangleright$ si chiama *ordinamento per sussunzione propria*.

L'*ordinamento per contenimento* $\trianglerighteq$ è la composizione dell'ordinamento per sottotermine e dell'ordinamento per sussunzione: $t \trianglerighteq s$ se $t|u = s\sigma$ per una posizione u e sostituzione σ; $t \triangleright s$ se $t \trianglerighteq s$ e $s \stackrel{\bullet}{\neq} t$.

Similmente, una sostituzione σ è un'istanza di una sostituzione θ se c'è una terza sostituzione ρ tale che $\forall x \in Dom(\sigma), x\sigma = x\theta\rho$. La sostituzione σ è una istanza propria di θ se ρ non è una ridenominazione. Come per i termini, si scrive $\sigma \trianglerighteq \theta$ e $\sigma \triangleright \theta$.

Dati due termini s e t, una sostituzione σ è un *unificatore* di s e t se $s\sigma = t\sigma$. Una sostituzione σ è un *unificatore più generale* (most general unifier: mgu) di s e t se σ è un unificatore di s e t e per tutti gli unificatori ρ di s e t, $\rho \trianglerighteq \sigma$.

Si definiscono le seguenti proprietà per un ordinamento $\succ$ su termini:

- *monotonicità:* $s \succ t$ implica $c[s] \succ c[t]$ per tutti i contesti c,
- *stabilità*: $s \succ t$ implica $s\sigma \succ t\sigma$ per tutte le sostituzioni σ e
- *proprietà sottotermine*: $c[s] \succ s$ per tutti i termini s e contesti c.

La prima proprietà dice che se $s \succ t$, allora ogni termine $c[s]$ ottenuto inserendo s in un contesto c è maggiore del termine $c[t]$. La seconda proprietà stabilisce che le sostituzioni preservano la re-

lazione d'ordine sui termini. La terza dice che un termine è maggiore di ogni suo sottotermine, cioè un ordinamento con questa proprietà include l'ordinamento per sottotermine. Un ordinamento monotono, stabile e ben fondato è un *ordinamento di riduzione*. Un ordinamento monotono, stabile e con la proprietà sottotermine è un *ordinamento di semplificazione*. Monotonicità, stabilità e proprietà sottotermine implicano che l'ordinamento è ben fondato [27]. Quindi un ordinamento di semplificazione è anche un ordinamento di riduzione.

Un'*equazione* è una coppia di termini non ordinata $l \simeq r$. Una *regola di riscrittura* è una coppia ordinata di termini $l \to r$. Un insieme di regole di riscrittura è detto *sistema di riscrittura di termini* o semplicemente *sistema di riscrittura*. Un'equazione $l \simeq r$ è orientata in una regola di riscrittura $l \to r$, se $l \succ r$ secondo un ordinamento di riduzione $\succ$.

Un sistema di riscrittura R definsce una relazione $\to_R$ su termini come segue: $s \to_R t$ se ci sono una regola di riscrittura $l \to r \in R$, una sostituzione σ e una posizione u tali che $s|u = l\sigma$ e t è $s[r\sigma]_u$. La relazione $\leftrightarrow_R$ è definita come l'unione $\to_R \cup \leftarrow_R$. Per un insieme di equazioni E, $s \leftrightarrow_E t$ se ci sono un'equazione $l \simeq r \in E$, una sostituzione σ e una posizione u tali che $s|u = l\sigma$ e t è $s[r\sigma]_u$; $s \to_E t$ se $s \leftrightarrow_E t$ e $s \succ t$ secondo un ordinamento di riduzione $\succ$. Indichiamo con $\leftrightarrow_E^*$ la chiusura transitiva e riflessiva di $\leftrightarrow_E$. La relazione $\leftrightarrow_E^*$ è una congruenza, la congruenza definita da E sull'insieme dei termini. L'uguaglianza $\leftrightarrow_E = \to_E \cup \leftarrow_E$ non vale in generale: vale se e solo se l'ordinamento $\succ$ è totale in ogni classe di congruenza definita da E.

Le definizioni seguenti si applicano sia a un sistema di riscrittura R che a un insieme di equazioni E. La sola differenza è nel modo in cui le relazioni $\to_R$ e $\to_E$ sono definite, come mostrato sopra. Si denotano con $\leftrightarrow_E^*$, $\to_E^*$ e $\leftarrow_E^*$ la chiusura transitiva e riflessiva di $\leftrightarrow_E$, $\to_E$ e $\leftarrow_E$ rispettivamente. Un termine s è in *forma normale* rispetto a E, o equivalentemente è E-*irriducibile*, se non esiste un termine t tale che $s \to_E t$. Un insieme di equazioni E è *Church-Rosser* se per tutti i termini s e t, $s \leftrightarrow_E^* t$ implica $s \to_E^* \circ \leftarrow_E^* t$. E' *confluente* se per tutti i termini s e t, $s \leftarrow_E^* \circ \to_E^* t$ implica $s \to_E^* \circ \leftarrow_E^* t$. La proprietà di Church-Rosser e la proprietà di confluenza sono equivalenti se $\leftrightarrow_E = \to_E \cup \leftarrow_E$. Un insieme di equazioni E è *localmente confluente* se per tutti i termini s e t, $s \leftarrow_E \circ \to_E t$ implica $s \to_E^* \circ \leftarrow_E^* t$. E' *canonico* se è sia confluente che *ridotto*, cioè per ogni $l \simeq r \in E$, l e r sono in forma normale rispetto a $E - \{l \simeq r\}$.

Sia P un insieme finito di simboli di predicato con le loro arità. Si denotano con $A(P, F, X)$ e $A(P, F)$ gli insiemi degli *atomi* e degli *atomi ground* su $< P, F, X >$.

Se P include il predicato di uguaglianza $\simeq$, un'equazione è un atomo in $A(P, F, X)$. Un *letterale* è un atomo o un atomo negato. Una *clausola* è una disgiunzione di letterali. Una *clausola unitaria* è una clausola contenente un singolo letterale. Una *clausola di Horn* è una clausola contenente al più un letterale positivo, detto la *testa*, mentre i letterali negativi formano il *corpo* della clausola. Tutte le variabili che compaiono in una clausola sono implicitamente quantificate universalmente. Denotiamo con $\hat{s} \simeq \hat{t}$ un'equazione che contiene soltanto variabili quantificate universalmente e quindi può essere considerata come un'equazione ground.

Le definizioni date per termini e sostituzioni si estendono ad atomi, letterali e clausole. In particolare, l'ordinamento (proprio) per sussunzione si estende naturalmente ad atomi e clausole e l'ordinamento per contenimento si estende alle equazioni nel modo seguente: $(p \simeq q) \trianglerighteq (l \simeq r)$ significa che $p = c[l\sigma]$ e $q = c[r\sigma]$ per un contesto c e sostituzione σ; $(p \simeq q) \triangleright (l \simeq r)$ vale se c è

non vuoto o σ non è una ridenominazione.

I due ordinamenti di semplificazione più noti sono il *recursive path ordering* [27] e il *lexicographic path ordering* [58].

Dati n insiemi parzialmente ordinati $(A_1, \succ_1) \dots (A_n, \succ_n)$ l'*estensione lessicografica* $\succ_{lex}$ degli ordinamenti $\succ_1 \dots \succ_n$ è l'ordinamento su $A_1 \times \dots \times A_n$ così definito: $(a_1 \dots a_n) \succ_{lex} (b_1 \dots b_n)$ se e solo se c'è un i, $1 \leq i \leq n$, tale che $a_j = b_j$, $\forall j < i$ e $a_i \succ_i b_i$. Se gli ordinamenti $\succ_1 \dots \succ_n$ sono ben fondati, la loro estensione lessicografica è anch'essa ben fondata.

Dato un insieme parzialmente ordinato $(A, \succ)$ la *estensione ai multiinsiemi* $\succ_{mul}$ dell'ordinamento $\succ$ è l'ordinamento sull'insieme dei multiinsiemi su A, $M(A)$, definito nel modo seguente:

- $\{a\} \cup M \succ_{mul} \emptyset$, dove $\emptyset$ è il multiinsieme vuoto.
- $\{a\} \cup M \succ_{mul} \{a\} \cup N$ se $M \succ_{mul} N$.
- $\{a\} \cup M \succ_{mul} \{b\} \cup N$ se $a \succ b$ e $\{a\} \cup M \succ_{mul} N$.

La estensione ai multiinsiemi di un ordinamento ben fondato è ben fondata [26].

Assumiamo che $>$ sia un ordinamento parziale, detto *precedenza*, su F. Un termine $s = f(s_1 \dots s_n)$ è maggiore di un termine $t = g(t_1 \dots t_m)$ nel *recursive path ordering*, cioè $s = f(s_1 \dots s_n) \succ^{rpo} g(t_1 \dots t_m) = t$, se e solo se una delle condizioni seguenti è soddisfatta:

- $s_i \succeq^{rpo} t$ per un i, $1 \leq i \leq n$.
- $f > g$ e $s \succ^{rpo} t_j$, $\forall j$, $1 \leq j \leq m$.
- $f = g$ e $\{s_1 \dots s_n\} \succ^{rpo}{}_{mul} \{t_1 \dots t_m\}$ dove $\succ^{rpo}{}_{mul}$ è l'estensione ai multiinsiemi di $\succ^{rpo}$.

Un termine $s = f(s_1 \dots s_n)$ è maggiore di un termine $t = g(t_1 \dots t_m)$ nel *lexicographic path ordering*, cioè $s = f(s_1 \dots s_n) \succ^{lpo} g(t_1 \dots t_m) = t$, se e solo se una delle condizioni seguenti è soddisfatta:

- $s_i \succeq^{lpo} t$ per un i, $1 \leq i \leq n$.
- $f > g$ e $s \succ^{lpo} t_j$, $\forall j$, $1 \leq j \leq m$.
- $f = g$, $(s_1 \dots s_n) \succ^{lpo}{}_{lex} (t_1 \dots t_m)$ dove $\succ^{lpo}{}_{lex}$ è l'estensione lessicografica di $\succ^{lpo}$ e $\forall j \geq 2$, $s \succ^{lpo} t_j$.

Questi due ordinamenti sono facili da implementare, perché sono definiti in modo puramente *sintattico*. Un altro ordinamento ben noto è il *Knuth-Bendix ordering* [62]. Una diversa famiglia di ordinamenti è quella degli *ordinamenti semantici*, che richiedono un'interpretazione della segnatura dei termini su qualche dominio parzialmente ordinato. Una rassegna sugli ordinamenti e le loro proprietà si può trovare in [32].

Un ordinamento di semplificazione su termini si estende naturalmente a letterali e clausole. Per esempio, data una precedenza $>$ sull'insieme $F \cup P$ dei simboli di funzione e di predicato, un ordinamento di semplificazione $\succ$ su termini e letterali può essere definito come segue: $L = A(s_1 \dots s_n) \succ B(t_1 \dots t_m) = M$, se e solo se una delle condizioni seguenti è soddisfatta:

- $s_i \succeq M$ per un i, $1 \leq i \leq n$.

- $A > B$ e $L \succ t_j$, $\forall j$, $1 \leq j \leq m$.
- $A = B = \simeq$ e $\{s_1, s_2\} \succ_{mul} \{t_1, t_2\}$ dove $\succ_{mul}$ è l'estensione ai multiinsiemi di $\succ$.
- $A = B \neq \simeq$ e $(s_1 \dots s_n) \succ_{lex} (t_1 \dots t_m)$ dove $\succ_{lex}$ è l'estensione lessicografica di $\succ$.

Un *ordinamento di semplificazione completo* è un ordinamento di semplificazione totale sull'insieme $T(F) \cup A(P, F)$ dei termini e letterali ground. Una volta che si ha un ordinamento di semplificazione (completo) su letterali, lo si può estendere a un ordinamento di semplificazione (completo) su clausole applicando la estensione ai multiinsiemi all'ordinamento sui letterali.

2.2 Ordinamenti su prove per dimostrazione di teoremi

Un insieme finito di formule S è una *presentazione* della *teoria* $Th(S) = \{\varphi | S \models \varphi\}$. Un *problema di dimostrazione di teoremi* consiste nel decidere se $\varphi \in Th(S)$, dati una presentazione S di una teoria e un *obbiettivo* φ. Una *derivazione in dimostrazione di teoremi* è una sequenza di deduzioni

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \dots \vdash (S_i; \varphi_i) \vdash \dots,$$

dove ad ogni passo il problema di decidere $\varphi_i \in Th(S_i)$ viene ridotto al problema di decidere $\varphi_{i+1} \in Th(S_{i+1})$. Un passo $(S_i; \varphi_i) \vdash (S_{i+1}; \varphi_i)$, che modifica la presentazione, è un passo di *forward reasoning*. Un passo $(S_i; \varphi_i) \vdash (S_i; \varphi_{i+1})$, che modifica l'obbiettivo derivando un nuovo goal da quello dato, è un passo di *backward reasoning*. Informalmente, diciamo che la derivazione termina con successo allo stadio k se $\varphi_k \in Th(S_k)$ è banalmente vero e quindi si può asserire che $\varphi_0 \in Th(S_0)$.

In questa sezione introduciamo una nozione di *ordinamento su prove*, che ci permette di descrivere una derivazione in dimostrazione di teoremi come un processo di *riduzione di prove*. Denotiamo le prove con lettere greche maiuscole: $\Upsilon(S, \varphi)$ denota una prova di φ dagli assiomi in S. Le prove sono comunemente rappresentate come alberi i cui nodi sono etichettati da formule: l'albero associato a $\Upsilon(S, \varphi)$ ha φ come etichetta della radice, elementi di S come etichette delle foglie e un nodo ψ ha figli $\psi_1 \dots \psi_n$ se ψ è derivato da $\psi_1 \dots \psi_n$ in un passo in $\Upsilon(S, \varphi)$. In logica equazionale, una prova può anche essere rappresentata come una catena [8]

$$s_1 \leftrightarrow_{l_1 \simeq r_1} s_2 \leftrightarrow_{l_2 \simeq r_2} \dots \leftrightarrow_{l_{n-1} \simeq r_{n-1}} s_n,$$

dove $s_1 \leftrightarrow_{l_1 \simeq r_1} s_2$ significa che l'uguaglianza di s_1 ed s_2 è stabilita dall'equazione $l_1 \simeq r_1$, in quanto s_1 ed s_2 sono $c[l_1\sigma]$ e $c[r_1\sigma]$ per un contesto c e una sostituzione σ. Si scrive $s \rightarrow_{l \simeq r} t$ se $s \succ t$ vale nel dato ordinamento $\succ$ sui termini. Una prova di forma $s \leftarrow \circ \rightarrow t$ è detta un *picco*. Una prova di forma $s \rightarrow^* \circ \leftarrow^* t$ è detta una *prova di riscrittura*.

Un ordinamento su prove si definisce in generale a partire da un ordinamento sui dati della prova. Assumiamo pertanto un ordinamento di semplificazione completo $\succ$. Preferiamo un ordinamento di semplificazione, anche se un ordinamento di riduzione che sia totale su dati ground è sufficiente. Un *ordinamento su prove* è un ordinamento ben fondato, monotono e stabile su prove [8]. Per esempio, il seguente ordinamento definito in [36] è un ordinamento su prove ground equazionali:

Esempio 2.2.1 *Si associa a ogni passo $s \leftrightarrow_{l \simeq r} t$ di una prova ground equazionale la tripla (s, l, t),*

se $s \succ t$. Si confrontano queste triple con l'estensione lessicografica $>^e$ dell'ordinamento di semplificazione completo $\succ$, dell'ordinamento per contenimento stretto $\rhd$ e ancora dell'ordinamento $\succ$. Infine si confrontano due prove $\Upsilon(E, s \simeq t)$ ed $\Upsilon'(E', s \simeq t)$ con l'estensione ai multiinsiemi, $>^e{}_{mul}$, di $>^e$.

Gli ordinamenti su prove così come definiti in [8] ci permettono di confrontare solo due prove $\Upsilon(S, \varphi)$ ed $\Upsilon'(S', \varphi)$ dello stesso teorema φ in diverse presentazioni S ed S' della teoria. Questa nozione di ordinamento su prove non è adatta alla dimostrazione di teoremi, perché in una derivazione in dimostrazione di teoremi

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots$$

sia la presentazione che l'obbiettivo vengono trasformati. Allo scopo di confrontare la prova di φ_i in S_i e la prova di φ_{i+1} in S_{i+1}, ci occorre un ordinamento su prove tale che due prove $\Upsilon(S, \varphi)$ ed $\Upsilon'(S', \varphi')$ di teoremi diversi possono essere paragonate. Ordinamenti su prove con questa proprietà esistono e si possono ottenere abbastanza facilmente. L'ordinamento su prove dell'esempio precedente può essere trasformato in un ordinamento per prove di teoremi diversi nel modo seguente:

Esempio 2.2.2 *Possiamo confrontare due prove equazionali ground $\Upsilon(E, s \simeq t) = s \leftrightarrow^*_E t$ e $\Upsilon'(E', s' \simeq t') = s' \leftrightarrow^*_{E'} t'$ confrontando le coppie $(\{s, t\}, s \leftrightarrow^*_E t)$ e $(\{s', t'\}, s' \leftrightarrow^*_{E'} t')$ con l'estensione lessicografica $>_u$ dell'estensione ai multiinsiemi $\succ_{mul}$ dell'ordinamento $\succ$ sui termini e l'estensione ai multiinsiemi $>^e{}_{mul}$ di $>^e$.*

La prova minima è la *prova vuota*. Denotiamo con *true* il teorema la cui prova è vuota e assumiamo che *true* sia l'elemento minimo nell'ordinamento su termini e letterali. Data una coppia $(S; \varphi)$, si può selezionare una prova minimale tra tutte le prove di φ in S:

Definizione 2.2.1 *Dato un ordinamento su prove $>_p$, denotiamo con $\Pi(S, \varphi)$ una* prova minimale *di φ in S rispetto a $>_p$, cioè una prova tale che per tutte le prove $\Upsilon(S, \varphi)$ di φ in S, $\Upsilon(S, \varphi) \not<_p \Pi(S, \varphi)$.*

Grazie all'assunzione di un ordinamento su prove $>_p$, possiamo considerare una derivazione in dimostrazione di teoremi

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots,$$

come un processo di riduzione della prova $\Pi(S_0, \varphi_0)$ alla prova vuota e di φ_0 a *true*. A ogni passo $\Pi(S_i, \varphi_i)$ è rimpiazzato da $\Pi(S_{i+1}, \varphi_{i+1})$ e la derivazione termina con successo allo stadio k se $\Pi(S_k, \varphi_k)$ è vuota e φ_k è *true*.

La nostra generalizzazione della nozione classica di ordinamento su prove è più significativa di quanto possa sembrare a prima vista. Gli ordinamenti su prove vennero introdotti in [8] per dimostrare la correttezza della procedura di completamento di Knuth-Bendix intesa come procedura che genera un sistema di riscrittura confluente, non necessariamente finito. Una derivazione mediante la procedura di Knuth-Bendix in quel contesto è un processo

$$S_0 \vdash S_1 \vdash \ldots \vdash S_i \vdash \ldots,$$

che trasforma soltanto una presentazione, senza un teorema obbiettivo. In altri termini, è una derivazione esclusivamente forward. Il fine di tale derivazione è trasformare una data presentazione equazionale finchè è confluente. Quindi è sufficiente essere in grado di paragonare $\Pi(S_i, \varphi)$ e $\Pi(S_{i+1}, \varphi)$ per ogni teorema φ nella teoria.

La situazione è diversa in dimostrazione di teoremi, dal momento che il fine di una derivazione è provare uno specifico teorema obbiettivo. La dimostrazione di teoremi richiede qualche forma di *backward reasoning*, in quanto un problema di dimostrazione di teoremi include un obbiettivo. Inoltre, una componente di backward reasoning è necessaria per ottenere una procedura *orientata all'obbiettivo* e dunque presumibilmente efficiente. L'approccio classico agli ordinamenti su prove non è adatto alla dimostrazione di teoremi perché non permette di descrivere il backward reasoning. Al contrario, il nostro approccio agli ordinamenti su prove descrive sia la dimostrazione di teoremi sia il completamento inteso in modo tradizionale.

2.3 Regole di inferenza e piani di ricerca

Poiché le procedure di completamento sono strategie di dimostrazione di teoremi con proprietà speciali, incominciamo con l'introdurre alcuni concetti di base sulle strategie di dimostrazione di teoremi.

Una *strategia di dimostrazione di teoremi* è data da una coppia $\mathcal{P} =< I; \Sigma >$, dove I è un insieme di *regole di inferenza* e Σ è un *piano di ricerca*. Le regole di inferenza in I decidono quali conseguenze si possono dedurre dai dati disponibili e Σ decide quale regola di inferenza e quali dati scegliere a ogni passo. Analizziamo prima le regole di inferenza e poi il piano di ricerca.

La forma generale di una regola di inferenza f è:

$$f\colon \frac{S}{S'}$$

dove S ed S' sono insiemi di formule. La regola dice che dato S, si può inferire l'insieme S'. Distinguiamo tra regole di *espansione* e regole di *contrazione*, come sono chiamate in [36]. Una regola di espansione espande l'insieme dato S in un nuovo insieme S' derivando nuove formule da formule in S:

$$f\colon \frac{S}{S'} \text{ dove } S \subset S'.$$

Una regola di contrazione contrae l'insieme dato S in un nuovo insieme S' eliminando alcune formule in S o rimpiazzandole con altre:

$$f\colon \frac{S}{S'} \text{ dove } S \not\subseteq S'.$$

Distinguiamo ulteriormente tra regole di inferenza che trasformano la presentazione e regole di inferenza che trasformano l'obbiettivo. Assumiamo che l'obbiettivo sia in forma clausale e quindi possa essere considerato come un insieme di letterali:

- *Regole di inferenza sulla presentazione*:

  - *Regole di espansione*: f: $\dfrac{(S;\varphi)}{(S';\varphi)}$ dove $S \subset S'$.
  - *Regole di contrazione*: f: $\dfrac{(S;\varphi)}{(S';\varphi)}$ dove $S \not\subseteq S'$.
- *Regole di inferenza sull'obbiettivo*:
  - *Regole di espansione*: f: $\dfrac{(S;\varphi)}{(S;\varphi')}$ dove $\varphi \subset \varphi'$.
  - *Regole di contrazione*: f: $\dfrac{(S;\varphi)}{(S;\varphi')}$ dove $\varphi \not\subseteq \varphi'$.

Esempio 2.3.1 *Per il metodo Unfailing Knuth-Bendix la presentazione S è un insieme di equazioni e l'obbiettivo φ è un'equazione ground $\hat{s} \simeq \hat{t}$. La* Deduzione *di una coppia critica è una regola di espansione sulla presentazione, poiché aggiunge all'insieme dato una nuova equazione:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma = l\sigma$$

dove X è l'insieme delle variabili della segnatura, σ è l'unificatore più generale di $p|u$ ed l e $\succ$ è il dato ordinamento di semplificazione completo su termini. La regola Deduzione *esegue una* sovrapposizione *tra due equazioni $p \simeq q$ ed $l \simeq r$. Un'equazione $l \simeq r$ si sovrappone su un'equazione $p \simeq q$ se esiste un sottotermine non variabile $p|u$, che unifica con mgu σ con l. Questo significa che il termine $p\sigma$ è uguale sia a $q\sigma$ che a $p[r]_u\sigma$. La nuova equazione $p[r]_u\sigma \simeq q\sigma$ è detta una* coppia critica. *Si genera una coppia critica solo se $p\sigma \not\preceq q\sigma, p[r]_u\sigma$, ovvero le due equazioni sono applicate in accordo con l'ordinamento di semplificazione. La* Semplificazione *è una regola di contrazione che si applica sia alla presentazione:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u = l\sigma \\ p \trianglerighteq l \vee q \succ p[r\sigma]_u \end{array} \qquad p \succ p[r\sigma]_u$$

che all'obbiettivo:

$$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u = l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$

Le condizioni $p \succ p[r\sigma]_u$ ed $\hat{s} \succ \hat{s}[r\sigma]_u$ assicurano che la semplificazione è ben fondata e $p \trianglerighteq l \vee q \succ p[r\sigma]_u$ si spiega come segue. Poiché $p|u = l\sigma$, è $p \trianglerighteq l$. Se $p \triangleright l$, o p è un'istanza propria di l o un sottotermine proprio di p è un'istanza di l. Il lato l dell'equazione applicata è strettamente minore del termine semplificato nell'ordinamento per contenimento e questa è una condizione sufficiente, con $p \succ p[r\sigma]_u$, affinché il passo di semplificazione riduca le prove equazionali dove $p \simeq q$ è applicata. Se $p \stackrel{\bullet}{=} l$, p ed l sono varianti, e quindi incommensurabili nell'ordinamento per contenimento. In questo caso, si richiede che la condizione $q \succ p[r\sigma]_u$ sia soddisfatta, cioè che il nuovo termine $p[r\sigma]_u$ sia minore di entrambi i lati dell'equazione semplificata $p \simeq q$. Questo requisito rappresenta una restrizione solo se $q \not\succ p$, cioè la semplificazione è eseguita sul lato maggiore di $p \simeq q$ o $p \simeq q$ non è ordinata. Se $q \succ p$, la semplificazione si applica al lato minore di $p \simeq q$ e la condizione $q \succ p[r\sigma]_u$ è banalmente soddisfatta. Vedremo nel seguito che queste condizioni garantiscono che un passo di semplificazione riduca le prove equazionali dove $p \simeq q$ è applicata.

Un requisito fondamentale per le regole di inferenza è quello di *soundness*. Una regola di inferenza sulla presentazione è *sound* se $Th(S') \subseteq Th(S)$, poiché un'applicazione di una regola sulla pre-

sentazione coinvolge la presentazione solamente. Una regola di inferenza sull'obbiettivo è *sound* se $Th(S \cup \{\varphi'\}) \subseteq Th(S \cup \{\varphi\})$.

La deduzione mediante regole sulla presentazione è deduzione di conseguenze dagli assiomi o *forward reasoning*. Una regola di inferenza sull'obbiettivo coinvolge sia la presentazione sia l'obbiettivo per derivare un nuovo obbiettivo: deduzione mediante regole sull'obbiettivo è *backward reasoning*. La procedura di Knuth-Bendix che computa coppie critiche e semplifica regole di riscrittura per ottenere un sistema canonico esegue esclusivamente forward reasoning. Se un teorema da provare viene dato come obbiettivo, la procedura esegue anche del backward reasoning, quando semplifica l'obbiettivo.

Infine, un piano di ricerca Σ decide quale regola di inferenza applicare a quali dati ad ogni dato passo durante una derivazione. Un piano molto primitivo consiste semplicemente nell'ordinare le formule nell'insieme dato in base a un criterio first-in first-out: le formule che sono state generate per prime sono selezionate per prime. Un simile piano è molto inefficiente perché non è basato su alcuna conoscenza dello stato della prova. Piani di ricerca più sofisticati fanno riferimento a qualche misura della distanza tra lo stato corrente della prova e la prova finale. Dal momento che la prova finale non è data, questa distanza non può essere determinata durante la derivazione e tale misura è un'euristica. Una volta che la misura è stata definita, la scelta migliore è quella che minimizza tale distanza. Un esempio di siffatta misura è la complessità delle clausole secondo l'ordinamento di semplificazione: dal momento che il teorema banale *true* è il minimo, allo stadio i si selezionano le clausole più piccole in S_i. Questo esempio mostra che una misura euristica della distanza tra lo stato corrente della computazione e uno stato finale può essere descritta mediante un *ordinamento ben fondato* sui dati. In aggiunta, il piano di ricerca può fissare una *precedenza* tra le regole di inferenza e procedere di conseguenza:

Esempio 2.3.2 *Un piano di ricerca* Simplification-first *[48] per la procedura Unfailing Knuth-Bendix è un piano di ricerca dove la Semplificazione ha priorità più alta della Deduzione. Quindi la Deduzione viene considerata solo se nessun passo di Semplificazione può essere eseguito. Le equazioni possono essere ordinate con l'estensione ai multiinsiemi $\succ_{mul}$ dell'ordinamento sui termini, o in base alla lunghezza, o in base alla durata in vita nell'insieme dei dati come in un piano first-in first-out.*

In [16] sono stati presentati diversi schemi per regole di inferenza, chiamati *deduzione* ed *eliminazione*. Lo schema di deduzione in [16] è uguale al nostro schema di espansione. Lo schema di eliminazione invece è diverso dal nostro schema di contrazione, perché permette di derivare S' da S solamente eliminando una formula in S. Se si adottano questi schemi di deduzione/eliminazione, una regola di inferenza che sostituisce una formula con altre formule deve essere schematizzata come la composizione di una regola di deduzione e di una regola di eliminazione. Per esempio, la Semplificazione di $p \simeq q$ in $p[r\sigma]_u \simeq q$ è descritta come la generazione dell'equazione $p[r\sigma]_u \simeq q$ seguita dall'eliminazione di $p \simeq q$.

Questo approccio ha il sostanziale svantaggio di richiedere che siano prese in considerazione regole di inferenza più generali di quelle effettivamente presenti nell'insieme delle regole di inferenza della strategia in esame. Nel caso della Semplificazione di $p \simeq q$ in $p[r\sigma]_u \simeq q$, non è detto che l'equazione $p[r\sigma]_u \simeq q$ sia una coppia critica derivabile mediante un passo della regola Deduzione.

Questo problema si verifica per esempio se $q \succ p$, cioè il lato destro di una regola di riscrittura viene semplificato. E' necessario allora assumere una regola di paramodulazione più generale della Deduzione, allo scopo di simulare la Semplificazione, ma tale regola di paramodulazione non appartiene alle regole di inferenza del metodo Unfailing Knuth-Bendix. Preferiamo i nostri schemi di espansione/contrazione, perché ci permettono di classificare direttamente ogni regola di inferenza della strategia in esame come una regola di espansione o una regola di contrazione.

2.4 Procedure di completamento

Una procedura di completamento $\mathcal{C}$ ha dunque tre componenti, $\mathcal{C} =< I_p, I_t; \Sigma >$, dove I_p è l'insieme delle regole di inferenza sulla presentazione, I_t è l'insieme delle regole di inferenza sull'obbiettivo e Σ è il piano di ricerca.

Una derivazione prodotta da una procedura di completamento è un processo di *riduzione di prove*. Un passo di inferenza sull'obbiettivo modifica l'obbiettivo a quindi la prova dell'obbiettivo. Richiediamo che la prova dell'obbiettivo venga ridotta:

Definizione 2.4.1 *Un passo di inferenza sull'obbiettivo* $(S;\varphi) \vdash (S;\varphi')$ *è* prova-riducente *se* $\Pi(S,\varphi) \geq_p \Pi(S,\varphi')$. *E'* strettamente prova-riducente *se* $\Pi(S,\varphi) >_p \Pi(S,\varphi')$.

Esempio 2.4.1 *La Semplificazione dell'obbiettivo definita nell'Esempio 2.3.1 è strettamente prova-riducente. Se assumiamo l'ordinamento su prove* $>_u$ *introdotto nell'Esempio 2.2.2, abbiamo* $\{\hat{s},\hat{t}\} \succ_{mul} \{\hat{s}',\hat{t}'\}$, *poiché* $\hat{s} \succ \hat{s}'$ *e* $\hat{t} = \hat{t}'$, *assumendo che* $\hat{s}$ *sia semplificato in* $\hat{s}'$. *Perciò* $\Pi(E, \hat{s} \simeq \hat{t}) >_u \Pi(E, \hat{s}' \simeq \hat{t}')$.

Per una passo di inferenza sulla presentazione si consente più flessibilità:

Definizione 2.4.2 *Date due coppie* $(S;\varphi)$ *e* $(S';\varphi')$, *la relazione* $(S;\varphi) \rhd_{p,\mathcal{T}} (S';\varphi')$ *vale se una delle due condizioni seguenti è soddisfatta:*

1. $\Pi(S,\varphi) >_p \Pi(S',\varphi')$ *oppure*
2. *(a)* $\Pi(S,\varphi) = \Pi(S',\varphi')$,
   (b) $\forall \psi \in \mathcal{T}$, $\Pi(S,\psi) \geq_p \Pi(S',\psi)$ *e*
   (c) $\exists \psi \in \mathcal{T}$ *tale che* $\Pi(S,\psi) >_p \Pi(S',\psi)$.

Definizione 2.4.3 *Un passo di inferenza sulla presentazione* $(S;\varphi) \vdash (S';\varphi)$ *è* prova-riducente *su* $\mathcal{T}$ *se* $(S;\varphi) \rhd_{p,\mathcal{T}} (S';\varphi)$. *E'* strettamente prova-riducente *se* $\Pi(S,\varphi) >_p \Pi(S',\varphi)$.

La condizione $(S_i;\varphi_i) \rhd_{p,\mathcal{T}} (S_{i+1};\varphi_{i+1})$ dice che il passo $(S_i;\varphi_i) \vdash (S_{i+1};\varphi_{i+1})$ riduce la prova dell'obbiettivo o riduce la prova di almeno un teorema in $\mathcal{T}$, mentre non rende più complessa la prova di alcun teorema in $\mathcal{T}$. Un passo che riduce la prova dell'obbiettivo è prova-riducente, indipendentemente dal suo effetto sulle prove di altri teoremi. D'altra parte, un passo di inferenza sulla presentazione può non ridurre immediatamente la prova dell'obbiettivo e tuttavia essere necessario per ridurla in seguito. Un simile passo è anch'esso prova-riducente, purché non aumenti la complessitá di nessuna prova e ne riduca strettamente almeno una.

Esempio 2.4.2 *La Deduzione di una coppia critica definita nell'Esempio 2.3.1 è prova-riducente. Si assuma l'ordinamento su prove $>_u$ introdotto nell'Esempio 2.2.2. Date due equazioni $l \simeq r$ e $p \simeq q$, una* sovrapposizione critica *di $l \simeq r$ e $p \simeq q$ è una prova $s \leftarrow_{l\simeq r} v \rightarrow_{p\simeq q} t$, dove v è $c[p\tau]$, c è un contesto, τ è una sostituzione, $p|u$ è un sottotermine non variabile di p e $(p|u)\tau = l\tau$ così che s è $c[p[r]_u\tau]$ e t è $c[q\tau]$. Un'applicazione della regola Deduzione a $l \simeq r$ e $p \simeq q$ genera la coppia critica $p[r]_u\sigma \simeq q\sigma$, dove σ è l'mgu di $p|u$ ed l, così che $\tau = \sigma\rho$ per qualche sostituzione ρ. Tale passo di Deduzione modifica una prova minimale sostituendo ogni occorrenza della sovrapposizione critica $s \leftarrow_{l\simeq r} v \rightarrow_{p\simeq q} t$ con il passo equazionale $s \leftrightarrow_{p[r]_u\sigma\simeq q\sigma} t$, giustificato dalla coppia critica. Si ha $\{(v,l,s),(v,p,t)\} >^e_{mul} \{(s,p[r]_u\sigma,t)\}$ se $s \succ t$, o $\{(v,l,s),(v,p,t)\} >^e_{mul} \{(t,q\sigma,s)\}$ se $t \succ s$, in quanto $v \succ s$ e $v \succ t$. Quindi $\Pi(E,\psi) >_u \Pi(E \cup \{p[r]_u\sigma \simeq q\sigma\},\psi)$ se una prova minimale di ψ in E contiene una sovrapposizione critica di $l \simeq r$ su $p \simeq q$, $\Pi(E,\psi) = \Pi(E \cup \{p[r]_u\sigma \simeq q\sigma\},\psi)$ altrimenti. Questo significa che la Deduzione è prova-riducente. Se una prova minimale dell'obbiettivo contiene una sovrapposizione critica di $l \simeq r$ su $p \simeq q$, il passo di Deduzione è strettamente prova-riducente.*

Esempio 2.4.3 *La Semplificazione di una equazione nella presentazione, data nell'Esempio 2.3.1, è prova-riducente. Si assuma l'ordinamento su prove $>_u$ introdotto nell'Esempio 2.2.2. Un passo di Semplificazione dove un'equazione $p \simeq q$ viene semplificata in $p[r\sigma]_u \simeq q$ da un'equazione $l \simeq r$, modifica una prova minimale sostituendo ogni passo $s \leftrightarrow_{p\simeq q} t$ con due passi $s \rightarrow_{l\simeq r} v \leftrightarrow_{p[r\sigma]_u\simeq q} t$.*

- *Se $t \succ s$, si ha $\{(t,q,s)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ poiché $t \succ s$ ed $s \succ v$.*
- *Se $s \succ t$,*
  - *se $p \triangleright l$, abbiamo*
    - *se $t \succ v$, $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ in quanto $p \triangleright l$ ed $s \succ t$,*
    - *se $v \succ t$, $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(v,q,t)\}$ in quanto $p \triangleright l$ ed $s \succ v$;*
  - *se $p \stackrel{\bullet}{=} l$ e $q \succ p[r\sigma]_u$, $t \succ v$ segue da $q \succ p[r\sigma]_u$ per stabilità e monotonicità di $\succ$ e abbiamo $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ perché $t \succ v$ e $s \succ t$.*

Quindi $\Pi(E \cup \{p \simeq q\},\psi) >_u \Pi(E \cup \{p[r\sigma]_u \simeq q\},\psi)$ se una prova minimale di ψ in E contiene un passo $s \leftrightarrow_{p\simeq q} t$, $\Pi(E \cup \{p \simeq q\},\psi) = \Pi(E \cup \{p[r\sigma]_u \simeq q\},\psi)$ altrimenti. Come per Deduzione, se un passo $s \leftrightarrow_{p\simeq q} t$ occorre in una prova minimale dell'obbiettivo, il passo di Semplificazione è strettamente prova-riducente.

Questa nozione di riduzione di prova si applica a passi che espandono la presentazione ed a passi che contraggono la presentazione sostituendo delle formule con altre. Un passo di contrazione che elimina una formula φ da un insieme S senza aggiungerne non può ridurre nessuna prova minimale: è impossibile che sia $\Pi(S,\psi) <_p \Pi(S \cup \{\varphi\},\psi)$ per qualche ψ, dal momento che $\Pi(S,\psi)$ è una prova anche in $S \cup \{\varphi\}$. Allo scopo di caratterizzare questi passi, introduciamo una nozione di *ridondanza*:

Definizione 2.4.4 *Una formula φ è* ridondante *in S sul dominio $\mathcal{T}$ se $\forall\psi \in \mathcal{T}$, $\Pi(S,\psi) = \Pi(S \cup \{\varphi\},\psi)$.*

Una formula è ridondante in una presentazione relativamente a un certo dominio se la sua presenza lascia inalterate tutte le prove minimali in S dei teoremi in $\mathcal{T}$.

Esempio 2.4.4 *Una regola di inferenza della procedura Unfailing Knuth-Bendix che elimina un'equazione senza aggiungerne è la* Sussunzione funzionale*:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright (l \simeq r)$$

La Sussunzione funzionale *elimina un'equazione $p \simeq q$ perché è sussunta da un'altra equazione $l \simeq r$, cioè $p = c[l\sigma]$ e $q = c[r\sigma]$, dove c è un contesto e σ è una sostituzione tali che o c è non vuoto o σ non è una ridenominazione di variabili. Un'equazione $p \simeq q$ sussunta da $l \simeq r$ è ridondante secondo l'ordinamento su prove ${>^e}_{mul}$ e quindi secondo l'ordinamento su prove $>_u$ definito nell'Esempio 2.2.2. Nessuna prova minimale contiene un passo $s \leftrightarrow_{p \simeq q} t$ poiché il passo $s \leftrightarrow_{l \simeq r} t$ è più piccolo: se $s \succ t$, è $\{(s, p, t)\} {>^e}_{mul} \{(s, l, t)\}$ e se $t \succ s$, è $\{(t, q, s)\} {>^e}_{mul} \{(t, r, s)\}$, in quanto $p \triangleright l$ e $q \triangleright r$.*

Una nozione di clausole ridondanti venne introdotta in [74] e in [16], dove il termine "ridondante" fu usato per la prima volta. Dedicheremo una sezione del prossimo capitolo a una discussione della definizione di ridondanza e vedremo che la nostra è più generale di quelle in [74] e [16]. La seguente definizione di *riduzione* copre sia la riduzione di prove sia l'eliminazione di elementi ridondanti:

Definizione 2.4.5 *Un passo di inferenza $(S; \varphi) \vdash (S'; \varphi')$ è* riducente *su $\mathcal{T}$ se è prova-riducente su $\mathcal{T}$ o elimina formule ridondanti in S sul dominio $\mathcal{T}$.*

Definizione 2.4.6 *Una regola di inferenza f è* riducente *se tutti i passi $(S; \varphi) \vdash_f (S'; \varphi')$ dove si applica f sono riducenti.*

Abbiamo finalmente tutti gli elementi per definire una procedura di completamento:

Definizione 2.4.7 *Una strategia di dimostrazione di teoremi $\mathcal{C} =< I_p, I_t; \Sigma >$ è una* procedura di completamento *su dominio $\mathcal{T}$ se per tutte le coppie $(S_0; \varphi_0)$, dove S_0 è una presentazione di una teoria e $\varphi_0 \in \mathcal{T}$, la derivazione*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

ha le proprietà seguenti:

- *monotonicità: $\forall i \geq 0$, $Th(S_{i+1}) \subseteq Th(S_i)$,*
- *rilevanza: $\forall i \geq 0$, $\varphi_i \in \mathcal{T}$ e $\varphi_{i+1} \in Th(S_{i+1})$ se e solo se $\varphi_i \in Th(S_i)$ e*
- *riduzione: $\forall i \geq 0$, il passo $(S_i; \varphi_i) \vdash_{\mathcal{C}} (S_{i+1}; \varphi_{i+1})$ è riducente su $\mathcal{T}$.*

Il *dominio* $\mathcal{T}$ è l'insieme delle formule su cui le regole di inferenza della procedura di completamento sono riducenti. Per esempio, per la procedura di completamento di Knuth-Bendix, $\mathcal{T}$ è l'insieme di tutte le equazioni. Per Unfailing Knuth-Bendix, $\mathcal{T}$ è l'insieme di tutte le equazioni ground.

Le proprietà di *monotonicità* e di *rilevanza* stabiliscono la soundness delle regole di inferenza sulla presentazione e sull'obbiettivo rispettivamente. La monotonicità assicura che un passo di inferenza sulla presentazione non crei nuovi elementi che non appartengono alla teoria, mentre la rilevanza assicura che un passo sull'obbiettivo lo sostituisca con un nuovo obbiettivo in modo

tale che dimostrare quest'ultimo equivalga a dimostrare il primo. Per esempio, un passo di semplificazione che riduce un obbiettivo φ a φ' soddisfa il requisito di rilevanza, poiché φ' è vero se e solo se φ è vero. Un'interessante regola di espansione sull'obbiettivo, detta *Saturazione ordinata*, verrà descritta in dettaglio nella Sezione 6.3.

La proprietà di *riduzione* è quella che caratterizza le procedure di completamento. E' chiaro che se tutte le regole di inferenza di una procedura sono riducenti, la procedura ha la proprietà di riduzione. Abbiamo visto negli esempi precedenti e vedremo inoltre nel Capitolo 6 che le regole di inferenza delle procedure di completamento note in logica equazionale sono riducenti. Molte regole di inferenza sono riducenti perché la loro applicazione è ristretta in modo appropriato da un ordinamento di semplificazione completo sui termini. Un ordinamento di semplificazione completo sui dati risulta essere un elemento chiave nel caratterizzare una strategia di dimostrazione di teoremi come procedura di completamento.

Capitolo 3

Dimostrazione di teoremi

La completezza di una procedura di completamento per dimostrazione di teoremi dipende sia dalle regole di inferenza che dal piano di ricerca. Innanzitutto, richiede che ogni volta che $\varphi_0 \in Th(S_0)$ per il dato input $(S_0; \varphi_0)$, ci siano derivazioni di successo prodotte dalle regole di inferenza della procedura. In secondo luogo, richiede che se esistono derivazioni di successo, il piano di ricerca garantisca che la derivazione computata avrà successo. Chiamiamo queste proprietà *completezza refutazionale* delle regole di inferenza e *fairness* del piano di ricerca, rispettivamente. Usiamo completezza refutazionale per le regole di inferenza onde riservare *completezza* per la strategia complessiva data dalle regole di inferenza e dal piano di ricerca.

Si conoscono molti insiemi di regole di inferenza refutazionalmente completi, ma relativamente scarsa attenzione è stata data al problema della fairness. Un piano di ricerca che esaurisce l'intero spazio di ricerca è ovviamente fair, ma inefficiente in modo grossolano. Il problema è riconciliare fairness ed efficienza. Questo problema diventa ancora più intricato in presenza di regole di contrazione.

La fairness di metodi basati sul completamento è stata studiata in [11, 74, 16]. Una derivazione prodotta da una procedura di completamento è fair secondo [11] se tutte le coppie critiche da equazioni *persistenti* vengono prima o poi generate e tutte le equazioni generate vengono prima o poi ridotte il più possibile. Questa definizione di fairness coglie esattamente i requisiti che una derivazione deve soddisfare per generare un sistema confluente, ma non è intesa per una derivazione per provare uno specifico teorema. Infatti, la maggioranza delle derivazioni per dimostrare teoremi non soddisfa questa definizione di fairness, dal momento che provare un teorema specifico non richiede in generale di generare tutte le coppie critiche.

Quindi, un piano di ricerca che sia fair secondo queste definizioni può portare il dimostratore ad eseguire deduzioni che sono completamente irrilevanti a provare il teorema desiderato. In questo capitolo definiamo la completezza refutazionale delle regole di inferenza e la fairness del piano di ricerca in termini di riduzione di prove, in accordo con l'approccio sviluppato nel capitolo precedente. Dimostriamo che se le regole di inferenza sono refutazionalmente complete e e il piano di ricerca è fair secondo le nostre definizioni, la procedura di completamento è completa e quindi è una *procedura di semidecisione* per dimostrazione di teoremi. Secondo il nostro punto di vista, questa è l'interpretazione più importante del completamento.

Il nostro approccio al problema della fairness è nuovo in tre aspetti, tutti rilevanti per la dimostrazione di teoremi. Innanzitutto, la nostra definizione è *orientata all'obbiettivo*, cioè tiene in considerazione il teorema da provare. Una definizione di fairness orientata all'obbiettivo è importante affinché un piano di ricerca possa essere sia fair che efficiente. Per esempio, si consideri un problema $(E; \hat{s} \simeq \hat{t})$, dove l'obbiettivo $s \simeq t$ è un'equazione su una segnatura F_1 e la presentazione in input E è l'unione di un insieme E_1 di equazioni sulla segnatura F_1 e di un insieme E_2 di equazioni su un'altra segnatura F_2, disgiunta da F_1. Una simile situazione può verificarsi in definizioni di tipi di dati astratti, se la segnatura F_1 contiene i costruttori e parte dei simboli definiti, mentre la segnatura F_2 è un altro insieme di simboli definiti. Secondo la nostra definizione, una derivazione da $(E; \hat{s} \simeq \hat{t})$ dove non si esegue nessuna inferenza da E_2 è fair. Al contrario, la definizione di fairness nei precedenti lavori sul completamento [11, 74, 16] richiederebbe di computare anche le coppie critiche tra le equazioni in E_2.

In secondo luogo, mettiamo in evidenza la distinzione tra fairness come proprietà del piano di ricerca e completezza come proprietà delle regole di inferenza. La maggior parte delle strategie di dimostrazione di teoremi sono definite dando semplicemente un insieme di regole di inferenza e il compito di progettare un opportuno piano di ricerca viene lasciato alla fase di implementazione. Questa scelta non è soddisfacente, dal momento che le prestazioni effettive del dimostratore dipendono fortemente dal piano di ricerca. Quindi, riteniamo che sia importante studiare sistematicamente i piani di ricerca e definire formalmente i loro requisiti. Il nostro approccio al problema della fairness è un primo passo in questa direzione. Infine, è generalmente riconosciuto che l'uso di regole di contrazione è necessario per l'efficienza della strategia. Tuttavia, le note definizioni di fairness pongono l'accento sull'applicazione delle regole di espansione, mentre la nostra tratta uniformemente sia le regole di espansione che quelle di contrazione.

Nel capitolo precedente abbiamo introdotto una nozione di *ridondanza*. L'interesse per la ridondanza di dati in derivazioni per dimostrazione di teoremi risiede nell'importanza delle regole di contrazione. Le regole di contrazione sono necessarie per rendere attuabile la dimostrazione di teoremi. Ciò nonostante, si conoscono ben poche regole di contrazione. Il fine dello studio della ridondanza è di aiutare la ricerca di nuove e potenti regole di contrazione. Pertanto concludiamo questo capitolo sulla dimostrazione di teoremi con una discussione sulla ridondanza.

3.1 Procedure di completamento come procedure di semidecisione

Data una coppia $(S_0; \varphi_0)$ in input, una procedura di completamento lavora riducendo la prova $\Pi(S_0, \varphi_0)$. Se la prova dell'obbiettivo è mimimale, il processo termina. Poichè la prova vuota è minore di ogni altra prova, la computazione si ferma allo stadio k se $\Pi(S_k, \varphi_k)$ è vuota e φ_k è $true$.

Una procedura è *completa* se, ogni volta che φ_0 è un teorema di S_0, la derivazione da $(S_0; \varphi_0)$ riduce φ_0 a $true$ e termina. Questo richiede la *completezza refutazionale* delle regole di inferenza e la *fairness* del piano di ricerca.

Per descrivere queste due proprietà, introduciamo una struttura detta *I-albero*. Dato un

problema di dimostrazione di teoremi $(S_0; \varphi_0)$ e un insieme di regole di inferenza I, l'applicazione di I a $(S_0; \varphi_0)$ definisce un albero, l'*I-albero con radice in* $(S_0; \varphi_0)$. I nodi dell'albero sono etichettati da coppie $(S; \varphi)$. La radice è etichettata dalla coppia di input $(S_0; \varphi_0)$. Un nodo $(S; \varphi)$ ha un figlio $(S'; \varphi')$ se $(S'; \varphi')$ può essere derivato da $(S; \varphi)$ in un passo da una regola di inferenza di I. L'I-albero con radice in $(S_0; \varphi_0)$ rappresenta tutte le derivazioni possibili mediante le regole di inferenza in I a partire da $(S_0; \varphi_0)$.

Intuitivamente, un insieme I di regole di inferenza è *refutazionalmente completo* se tutte le volte che $\varphi_0 \in Th(S_0)$, l'I-albero con radice in $(S_0; \varphi_0)$ contiene nodi di successo, nodi di tipo $(S; true)$. Usiamo "completezza refutazionale" per un insieme di regole di inferenza e "completezza" per l'intera strategia. La definizione seguente è una caratterizzazione equivalente di questo concetto in termini di riduzione di prove:

Definizione 3.1.1 *Un insieme $I = I_p \cup I_t$ di regole di inferenza è* refutazionalmente completo *se ogni volta che $\varphi \in Th(S)$ e $\Pi(S, \varphi)$ è non minimale, esistono derivazioni*

$$(S; \varphi) \vdash_I (S_1; \varphi_1) \vdash_I \ldots \vdash_I (S'; \varphi')$$

tali che $\Pi(S, \varphi) >_p \Pi(S', \varphi')$.

Questa definizione dice che un insieme I di regole di inferenza è refutazionalmente completo se può ridurre la prova dell'obbiettivo qualora non sia minimale. Dal momento che un ordinamento su prove è ben fondato, segue che se $\varphi \in Th(S)$, l'I-albero con radice in $(S; \varphi)$ contiene nodi di successo. Definire la completezza in termini di riduzione di prove è vantaggioso perchè consente di ridurre il problema di provare la completezza di I al problema di dare un opportuno ordinamento su prove.

Data una procedura di completamento $\mathcal{C} =< I_p, I_t; \Sigma >$, $I = I_p \cup I_t$, l'I-albero con radice in $(S_0; \varphi_0)$ rappresenta l'intero spazio di ricerca che la procedura può derivare dall'input $(S_0; \varphi_0)$. Il piano di ricerca Σ seleziona un cammino nell'I-albero: la derivazione dall'input $(S_0; \varphi_0)$ controllata da Σ è il cammino selezionato da Σ nell'I-albero con radice in $(S_0; \varphi_0)$. Una volta che un insieme di regole di inferenza e un piano di ricerca sono entrambi dati, la derivazione da $(S_0; \varphi_0)$ è unica. Una coppia $(S_i; \varphi_i)$ generata allo stadio i della derivazione è un *nodo visitato* nell'I-albero. Ogni nodo visitato $(S_i; \varphi_i)$ ha generalmente molti figli, ma il piano di ricerca ne seleziona soltanto uno per essere $(S_{i+1}; \varphi_{i+1})$. Un piano di ricerca Σ è *fair* se ogni volta che l'I-albero con radice in $(S_0; \varphi_0)$ contiene nodi di successo, è garantito che la derivazione controllata da Σ a partire da $(S_0; \varphi_0)$ raggiungerà un nodo di successo. Come per la completezza, esprimiamo questo concetto in termini di riduzione di prove:

Definizione 3.1.2 *Una derivazione*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

controllata da un piano di ricerca Σ è fair *se e solo se per tutti gli $i \geq 0$, se esiste nell'I-albero con radice in $(S_0; \varphi_0)$ un cammino*

$$(S_i; \varphi_i) \vdash_I \ldots \vdash_I (S'; \varphi')$$

tale che $\Pi(S_i, \varphi_i) >_p \Pi(S', \varphi')$, allora esiste un $(S_j; \varphi_j)$ per un $j > i$, tale che $\Pi(S', \varphi') \geq_p \Pi(S_j, \varphi_j)$. Un piano di ricerca Σ è fair *se tutte le derivazioni controllate da Σ sono fair.*

In altre parole, se le regole di inferenza permettono di ridurre la prova dell'obbiettivo in $(S_i; \varphi_i)$, un piano di ricerca fair garantisce che la prova dell'obbiettivo sarà stata ridotta a un qualche stadio successivo $(S_j; \varphi_j)$.

Se le regole di inferenza sono complete e il piano di ricerca è fair, la procedura di completamento è *completa*:

Teorema 3.1.1 *Sia $\mathcal{C} =< I_p, I_t; \Sigma >$ una procedura di completamento. Se l'insieme $I = I_p \cup I_t$ delle regole di inferenza è refutazionalmente completo e il piano di ricerca Σ è fair, allora per tutte le derivazioni*

$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots,$

dove $\varphi_0 \in Th(S_0)$, $\forall i \geq 0$, se $\Pi(S_i, \varphi_i)$ è non minimale, allora esiste un (S_j, φ_j), per un $j > i$, tale che $\Pi(S_i, \varphi_i) >_p \Pi(S_j, \varphi_j)$.

Dimostrazione: se $\Pi(S_i, \varphi_i)$ è non minimale, allora, per completezza delle regole di inferenza, esiste un cammino $(S_i; \varphi_i) \vdash_I \ldots \vdash_I (S'; \varphi')$ tale che $\Pi(S_i, \varphi_i) >_p \Pi(S', \varphi')$. Per la fairness del piano di ricerca, esiste un $(S_j; \varphi_j)$, per qualche $j > i$, tale che $\Pi(S_i; \varphi_i) >_p \Pi(S', \varphi') \geq_p \Pi(S_j, \varphi_j)$. □

Completezza significa che la procedura di completamento è una *procedura di semidecisione* per $Th(S) \cap \mathcal{T}$, dove S è una qualsiasi presentazione e $\mathcal{T}$ è il dominio della procedura di completamento:

Corollario 3.1.1 *Se una procedura di completamento $\mathcal{C}$ su dominio $\mathcal{T}$ ha regole di inferenza refutazionalmente complete e piano di ricerca fair, allora per tutti gli inputs $(S_0; \varphi_0)$ dove $\varphi_0 \in Th(S_0)$, la derivazione*

$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$

raggiunge uno stadio k, $k \geq 0$, tale che φ_k è la clausola true.

Dimostrazione: se $\varphi_0 \in Th(S_0)$, allora per il Teorema 3.1.1 e la ben fondatezza di $>_p$, la derivazione raggiunge uno stadio k tale che la prova $\Pi(S_k, \varphi_k)$ è minimale. Poichè la prova minimale è la prova vuota, φ_k è la clausola *true*. □

3.2 Ridondanza

Una nozione di clausole ridondanti apparve in [74] e in [16], dove il termine "ridondante" venne usato per la prima volta. Clausole ridondanti secondo [74] e [16] sono ridondanti secondo la nostra definizione. D'altra parte, ci sono clausole che sono intuitivamente ridondanti e ridondanti per noi, ma non per [74] e [16].

Definizione 3.2.1 (Rusinowitch 1987) [74] *Una clausola φ è* R-ridondante *in un insieme S se esiste una clausola $\psi \in S$ tale che ψ sussume propriamente φ, cioè $\varphi \succ\!\!\!\cdot\, \psi$, dove $\succ\!\!\!\cdot$ è l'ordinamento per sussunzione propria tra clausole.*

La definizione di ridondanza in [16] assume un ordinamento $>^d$ ben fondato e totale su clausole ground [1]:

Definizione 3.2.2 (Bachmair and Ganzinger 1990) [16] *Una clausola φ è* B-ridondante *in un insieme S se per tutte le istanze ground $\varphi\sigma$ di φ, ci sono istanze ground $\psi_1 \dots \psi_n$ di clausole in S tali che $\{\psi_1 \dots \psi_n\} \models \varphi\sigma$ e $\forall j, 1 \leq j \leq n, \varphi\sigma >^d \psi_j$.*

E' immediato vedere che se l'ordinamento $>^d$ è l'ordinamento per sussunzione propria, la Definizione 3.2.2 si riduce alla Definizione 3.2.1 e quindi una clausola che può essere sussunta è un esempio di clausola B-ridondante:

Lemma 3.2.1 (Bachmair and Ganzinger 1990) [16] *Se una clausola è R-ridondante, allora è B-ridondante.*

A nostro avviso, il significato intuitivo della nozione di ridondanza è che una clausola φ è ridondante in S se aggiungere φ ad S non diminuisce nessuna prova minimale in S (Definizione 2.4.4). La nostra definizione coglie infatti il contenuto della Definizione 3.2.2:

Teorema 3.2.1 *Se una clausola φ è B-ridondante in S, allora è ridondante sul dominio di tutte le clausole ground.*

Dimostrazione: assumiamo un ordinamento su prove ground $>_p$ tale che la prova minimale in S di una clausola ground φ, $\Pi(S, \varphi)$, è il minimo insieme $\{\psi_1 \dots \psi_n\}$, secondo l'estensione ai multiinsiemi $>^d{}_{mul}$ dell'ordinamento $>^d$, di istanze ground di clausole di S tale che $\{\psi_1 \dots \psi_n\} \models \varphi$. Poiché $>^d$ è ben fondato e totale su clausole ground, $>_p$ è ben definito. Sia φ una clausola B-ridondante in S. Mostriamo che $\Pi(S \cup \{\varphi\}, \psi) = \Pi(S, \psi)$ per tutti i teoremi ground ψ. Dal momento che é $S \subset S \cup \{\varphi\}$, $\Pi(S \cup \{\varphi\}, \psi) \leq_p \Pi(S, \psi)$ vale banalmente e quindi dobbiamo semplicemente mostrare che $\Pi(S \cup \{\varphi\}, \psi) \not<_p \Pi(S, \psi)$. La prova è per assurdo: se $\Pi(S \cup \{\varphi\}, \psi) <_p \Pi(S, \psi)$, allora il minimo insieme di istanze ground di clausole in $S \cup \{\varphi\}$ da cui segue logicamente ψ ha la forma $S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ per qualche insieme S' di istanze ground di clausole in S e sostituzioni ground $\sigma_1 \dots \sigma_k$. Siccome φ è B-ridondante in S, per tutte le $\varphi\sigma_i$, $1 \leq i \leq k$, ci sono istanze ground $\{\psi^i_1 \dots \psi^i_n\}$ di clausole in S tali che $\{\psi^i_1 \dots \psi^i_n\} \models \varphi\sigma_i$ e $\varphi\sigma_i >^d \psi^i_j$, $\forall j, 1 \leq j \leq n$. Dunque, $S' \cup \{\psi^i_1 \dots \psi^i_n\}^k_{i=1} <^d{}_{mul} S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ e $S' \cup \{\psi^i_1 \dots \psi^i_n\}^k_{i=1} \models \psi$, cioè $S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ non può essere l'insieme minimo da cui segue logicamente ψ. Segue che $\Pi(S \cup \{\varphi\}, \psi) = \Pi(S, \psi)$. □

D'altra parte, ci sono casi in cui clausole banalmente ridondanti non sono tali per la Definizione 3.2.2, mentre sono ridondanti secondo la nostra definizione:

Esempio 3.2.1 *Se $S = \{P, \neg R, R\}$, dove P ed R sono atomi ground, P è intuitivamente ridondante ed è ridondante secondo la Definizione 2.4.4: la prova minimale di ogni teorema ground è data da $\{\neg R, R\}$, in quanto $\{\neg R, R\}$ produce la clausola vuota e quindi ogni clausola. Tuttavia, se $R \succ P$ e quindi $R >^d P$, P non è B-ridondante.*

[1] In [16] una nozione di *ordinamento di eliminazione* viene definita a questo scopo. La ben fondatezza e la totalità su clausole ground sono le sole proprietà di un ordinamento di eliminazione che siano rilevanti per la nostra discussione.

Questo esempio mostra che una nozione di ridondanza basata su un ordinamento su clausole non è ideale, perché può essere necessario stabilire diverse precedenze sui simboli di predicato per caratterizzare come ridondanti diverse clausole durante la derivazione. Una nozione di ridondanza basata su un ordinamento su prove sembra essere più soddisfacente.

Concludiamo con una discussione sulla relazione tra ridondanza e regole di contrazione. Nel capitolo precedente abbiamo usato la ridondanza per caratterizzare quelle regole di contrazione che eliminano formule. E' immediato mostrare che la ridondanza ha una parte anche nelle regole di contrazione che sostituiscono una formula con altre. Se un passo di contrazione $(S \cup \{\psi\}; \varphi) \vdash (S \cup \{\psi'\}; \varphi)$ è prova-riducente secondo la Condizione 2 in Definizione 2.4.2, la formula ψ che è stata eliminata è chiaramente ridondante in $S \cup \{\psi'\}$. Però, questo non è necessariamente vero per una regola di contrazione che è prova-riducente secondo la Condizione 1 in Definizione 2.4.2, dal momento che la Condizione 1 dice semplicemente che la prova dell'obbiettivo viene strettamente ridotta, senza implicare nulla sulle prove degli altri teoremi. Una regola di contrazione può anche eliminare formule non ridondanti, purché sia sound e riduca strettamente la prova dell'obbiettivo.

Questa è un'ulteriore differenza tra il nostro approccio e quello in [16]. In [16] la ridondanza è usata per definire lo schema stesso di eliminazione: una regola di eliminazione è una regola che elimina formule ridondanti. Preferiamo non collegare così strettamente le nozioni di contrazione e di ridondanza, perchè le nozioni di ridondanza proposte finora non sono orientate all'obbiettivo. Secondo le tre definizioni di ridondanza che abbiamo considerato, una formula è ridondante se è inutile rispetto a tutti i teoremi nel dominio. Intuitivamente, una formula ridondante è una formula che è ovvio scartare. La nostra definizione di riduzione di prove (Definizione 2.4.2) permette in linea di principio regole di contrazione molto forti che possono eliminare anche formule non ridondanti se queste non contribuiscono a provare l'obbiettivo specifico. E' necessario ulteriore lavoro per volgere uno studio astratto della contrazione e della ridondanza in regole di inferenza concrete.

Capitolo 4

Generazione di procedure di decisione

Finora abbiamo considerato una procedura di completamento come una procedura per dimostrazione di teoremi. In questo capitolo estendiamo il nostro studio per includere procedure di completamento come *generatori di procedure di decisione*. In questo contesto le derivazioni prodotte da una procedura di completamento hanno la forma

$(S_0; \emptyset) \vdash_{\mathcal{C}} (S_1; \emptyset) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \emptyset) \vdash_{\mathcal{C}} \ldots$.

Non c'è un teorema obbiettivo: il fine della derivazione è di trasformare la presentazione solamente.

Se il piano di ricerca di una procedura di completamento soddisfa una proprietà di fairness più forte, che chiamiamo *fairness uniforme*, la procedura di completamento genera una presentazione *saturata*, potenzialmente infinita. La fairness uniforme è la classica proprietà di fairness richiesta nei precedenti lavori sulle procedure di completamento [52, 11, 74, 16]. Consiste sostanzialmente nel considerare prima o poi tutti i passi di inferenza. Il concetto di insieme saturato è una generalizzazione di quello di sistema di riscrittura confluente: una presentazione è saturata se nessuna conseguenza non banale può esserle aggiunta [63, 16]. Definiamo sia la fairness uniforme che la saturazione usando la nostra nozione di ridondanza e diamo una nuova prova del teorema che stabilisce che una derivazione uniformemente fair genera una presentazione saturata, non necessariamente finita. Questo teorema copre la generazione di sistemi di riscrittura confluenti mediante il completamento di Knuth-Bendix [62, 52, 8], la generazione di insiemi di equazioni ground confluenti mediante la procedura Unfailing Knuth-Bendix [48, 14] e la generazione di insiemi saturati in [63, 16]. Il nostro teorema è più generale di quelli in [63, 16], perché la nostra nozione di ridondanza è più generale, come abbiamo discusso nella Sezione 3.2.

Se una presentazione è saturata, le derivazioni da tale presentazione sono derivazioni *lineari input* [24], cioè derivazioni dove ogni passo si applica all'obbiettivo. Se si può garantire che le derivazioni lineari input da un insieme saturato terminino, un insieme saturato è una *procedura di decisione* e la procedura di completamento è un *generatore di procedure di decisione*.

Requisiti addizionali, che dipendono dalla logica, sono necessari per implicare la terminazione delle derivazioni lineari input. In logica equazionale, una derivazione $s \to^* \circ \leftarrow^* t$, che si compone soltanto di passi di semplificazione ben fondata, è una derivazione lineare input ben fondata e una presentazione equazionale ground confluente è una procedura di decisione per teoremi di tipo $\forall \bar{x} s \simeq t$. Si conoscono anche condizioni sufficienti per la terminazione delle derivazioni lineari

input per obbiettivi ground in logica di Horn con uguaglianza [63, 16].

Le nostre nozioni di fairness uniforme, insieme saturato, derivazione lineare input ben fondata, procedura di decisione e i relativi teoremi danno una presentazione dell'applicazione del completamento alla generazione di procedure di decisione, che unifica tutti i risultati noti sulla generazione di procedure di decisione in logica equazionale e di Horn con uguaglianza.

Poche teorie hanno una presentazione saturata finita e ancor più poche soddisfano i requisiti addizionali affinché una presentazione saturata sia una procedura di decisione. Quindi, l'interpretazione del completamento come procedura di semidecisione è la più utile in pratica.

4.1 Fairness uniforme e insiemi saturati

La Definizione 3.1.2 di fairness è sufficiente per la dimostrazione di teoremi, come mostrato dal Teorema 3.1.1. La Definizione 3.1.2 applicata al completamento di Knuth-Bendix, non è sufficiente a garantire che un sistema di riscrittura confluente sarà prima o poi generato, dal momento che non garantisce che tutte le coppie critiche vengano prima o poi considerate. Questo richiede una proprietà di fairness molto più onerosa, che chiamiamo *fairness uniforme*.

La prima definizione di fairness uniforme comparve in [52], dove si richiede che il piano di ricerca ordini le regole di riscrittura nell'insieme di dati con un ordinamento ben fondato, in modo da assicurare che nessuna regola sia posposta indefinitamente. Ricordiamo qui questa prima idea di fairness, perché asserisce esplicitamente che la fairness è una proprietà del piano di ricerca. Le successive definizioni di fairness sono state date a un livello di astrazione molto più alto che può non far notare al lettore che la fairness è una proprietà del meccanismo di ricerca.

Data una derivazione prodotta da una procedura di completamento a partire da una presentazione S_0, denotiamo con $S_\infty = \bigcup_{j\geq 0}\bigcap_{i\geq j} S_i$ l'insieme, potenzialmente infinito, di tutte le formule *persistenti*, cioè quelle formule che sono generate a un certo stadio e non più eliminate nel seguito. L'insieme S_∞ è detto il *limite* della derivazione. Usare il limite S_∞ è vantaggioso perchè se la derivazione termina a un qualche stadio k, $S_\infty = S_k$. Perciò, proprietà espresse facendo riferimento al limite S_∞ valgono uniformemente sia per derivazioni finite che per derivazioni infinite. Questo concetto è stato introdotto per la prima volta in [52] e usato tra gli altri in [8, 11, 33, 16]. Inoltre, indichiamo con $I_e(S)$ l'insieme delle clausole generabili in un passo da S mediante le regole di espansione della presentazione nella procedura di completamento.

Definizione 4.1.1 (Rusinowitch 1987) [74], (Bachmair and Ganzinger 1990) [16] *Una derivazione*

$(S_0;\emptyset)\vdash_{\mathcal{C}}(S_1;\emptyset)\vdash_{\mathcal{C}}\ldots(S_i;\emptyset)\vdash_{\mathcal{C}}\ldots$

da una procedura di completamento $\mathcal{C}$ su dominio $\mathcal{T}$ è uniformemente fair *su dominio $\mathcal{T}$ se $\forall\varphi\in I_e(S_\infty)$, esiste un S_j per qualche $j\geq 0$ tale che $\varphi\in S_j$ o φ è ridondante in S_j su dominio $\mathcal{T}$.*

Questa definizione di fairness generalizza le definizioni date in precedenza in [52] and [11]. Essa dice che ogni clausola φ che può essere persistentemente generata da una regola di espansione durante il processo di completamento viene o effettivamente generata in qualche passo, o rimpiazzata da altre clausole, che danno una prova di φ più piccola e rendono φ stessa ridondante. Per

esempio, una derivazione Knuth-Bendix tale che tutte le coppie critiche da equazioni persistenti sono prima o poi generate o sussunte o ridotte a un termine comune è uniformemente fair secondo questa definizione.

La fairness uniforme è stata studiata e ridefinita a più riprese in [11, 74, 16] al fine di risolvere il problema della interazione tra regole di espansione e regole di contrazione. Il significato intuitivo della fairness uniforme è di essere fair verso le regole di inferenza, cioè applicare tutte le regole di inferenza a tutti i dati. Questo è impossibile, tuttavia, perchè le regole di inferenza includono sia regole di espansione che regole di contrazione: se una clausola φ viene eliminata da un passo di contrazione prima che una regola di espansione f le sia applicata, la derivazione non è fair verso f. Il problema è quello di definire la fairness in modo tale che l'applicazione delle regole di contrazione sia fair. Questo problema è stato risolto nella definizione di fairness uniforme stabilendo che è fair non eseguire passi di espansione le cui premesse non siano persistenti ed è fair sostituire una clausola φ con clausole che la rendono ridondante. In questo modo l'applicazione di regole di contrazione come la semplificazione e la sussunzione è fair. In pratica, una procedura uniformemente fair eseguirà in modo esaustivo tutti i passi di espansione che non siano inibiti da passi di contrazione.

Fairness e fairness uniforme differiscono in tre punti fondamentali. La fairness è *orientata all'obbiettivo*, mentre la fairness uniforme è definita per una derivazione senza obbiettivo, perché non tiene in considerazione l'obbiettivo e quindi le regole di inferenza per l'obbiettivo. Infatti la Definizione 4.1.1 non è una definizione di fairness per la dimostrazione di teoremi. In [74, 16], la Definizione 4.1.1 viene applicata alla dimostrazione di teoremi per refutazione, dove S_0 contiene la negazione dell'obbiettivo. In questo caso la sola clausola persistente è la clausola vuota $\Box$ e $I_e(\bigcup_{i\geq 0}\bigcap_{j\geq i} S_j) = \{\Box\}$. Allora la Definizione 4.1.1 dice che il limite della derivazione è la clausola vuota. Una siffatta nozione di fairness non dà nessuna informazione utile alla progettazione di piani di ricerca perché non pone nessun requisito su come un piano di ricerca debba scegliere il successore ad ogni dato stadio della derivazione.

La definizione di fairness non fa differenza tra regole di espansione e di contrazione e tra clausole persistenti e non, perchè l'interazione tra regole di espansione e di contrazione non è più un problema. Tutte le regole di inferenza sono trattate in modo uniforme considerando i loro effetti rispetto al goal di ridurre la prova dell'obbiettivo. D'altra parte, regole di espansione e di contrazione sono trattate differentemente nella definizione di fairness uniforme. La fairness uniforme mette in evidenza le regole di espansione, mentre il ruolo delle regole di contrazione è nascosto nella restrizione a clausole persistenti. La ragione di ciò è che è necessario considerare tutte le coppie critiche per ottenere un insieme confluente, ma non è necessario ridurle, benché in pratica il completamento senza semplificazione sia inefficiente senza speranza.

L'esempio seguente illustra un insieme di condizioni per una derivazione della procedura Unfailing Knuth-Bendix, che sono state provate sufficienti per la fairness uniforme in [11]. Queste condizioni rappresentano la più nota definizione di fairness (uniforme) di una procedura di completamento:

Esempio 4.1.1 *Una derivazione*

$(E_0;\emptyset)\vdash_{UKB}(E_1;\emptyset)\vdash_{UKB}\ldots\vdash_{UKB}(E_i;\emptyset)\vdash_{UKB}\ldots$

è uniformemente fair se

- *per tutte le coppie critiche* $g \simeq d \in I_e(E_\infty)$, $g \simeq d \in \bigcup_{i\geq 0} E_i$ *e*
- E_∞ *è ridotto.*

La regola Deduzione data nell'Esempio 2.3.1 è la sola regola di espansione della procedura Unfailing Knuth-Bendix. Quindi la prima condizione dice che tutte le coppie critiche derivabili per Deduzione da equazioni persistenti sono prima o poi generate. La seconda condizione dice che tutte le equazioni persistenti sono prima o poi semplificate il più possibile. Come notato sopra, l'applicazione delle regole di contrazione è permessa, ma non richiesta: la prima condizione da sola è sufficiente per la fairness uniforme. Dal momento che ad ogni stadio della computazione non si sa quali equazioni persisteranno e quali saranno semplificate, queste condizioni per la fairness uniforme prescrivono in pratica di applicare esaustivamente tutte le regole di inferenza del metodo Unfailing Knuth-Bendix finché nessuna è applicabile.

Il concetto di fairness uniforme conduce alla seguente nozione di presentazione *saturata* :

Definizione 4.1.2 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *Una presentazione* S *è* saturata *sul dominio* $\mathcal{T}$ *di una procedura di completamento se e solo se* $\forall \psi \in I_e(S)$, $\psi \in S$ *o* ψ *è ridondante in* S *su* $\mathcal{T}$.

Tutte le conseguenze che si possono aggiungere ad una presentazione saturata sono banali. Nel caso equazionale, come venne notato in [63], un insieme di equazioni è saturato se nessuna coppia critica non banale può esserne dedotta, o, equivalentemente, l'insieme è *localmente confluente* [1]. Come nella definizione di fairness uniforme, l'applicazione delle regole di contrazione è permessa ma non richiesta: regole di contrazione possono essere ancora applicabili a un insieme saturato. Una presentazione equazionale localmente confluente non è necessariamente ridotta.

Se una derivazione è uniformemente fair, S_∞ è saturato. Poiché la fairness uniforme è definita in termini di ridondanza e la nostra nozione di ridondanza è più generale di quelle in [74] e [16], diamo una nuova prova di questo risultato:

Teorema 4.1.1 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *Se una derivazione*

$$(S_0;\emptyset) \vdash_{\mathcal{C}} (S_1;\emptyset) \vdash_{\mathcal{C}} \ldots (S_i;\emptyset) \vdash_{\mathcal{C}} \ldots$$

è uniformemente fair sul dominio $\mathcal{T}$, *allora* S_∞ *è* saturato *sul dominio* $\mathcal{T}$.

Dimostrazione: mostriamo che per tutte le $\varphi \in I_e(S_\infty)$, $\varphi \in S_\infty$ o φ è ridondante in S_∞ su $\mathcal{T}$. Per la fairness uniforme della derivazione, esiste un S_j, per qualche $j \geq 0$, tale che $\varphi \in S_j$ o φ è ridondante in S_j su $\mathcal{T}$:

1. se $\varphi \in S_j$, allora o φ è persistente, cioè $\varphi \in S_\infty$, o φ viene eliminata a qualche stadio $i > j$. Si hanno allora due casi: o φ è semplicemente eliminata o viene rimpiazzata da un'altra formula φ':

[1] Una coppia critica $g \simeq d$ è banale in E se $g \rightarrow_E^* \circ \leftarrow_E^* d$. E' ben noto, grazie al lemma di Knuth-Bendix [62, 51, 8] e alla sua estensione per UKB [48, 14], che un insieme di equazioni è localmente confluente se e solo se tutte le sue coppie critiche sono banali.

(a) se $S_i = S \cup \{\varphi\}$ ed $S_{i+1} = S$, per la Definizione 2.4.7 di procedura di completamento, un tale passo elimina un elemento ridondante. Allora φ è ridondante in S_i su $\mathcal{T}$. Poiché $\forall \psi \in \mathcal{T}$, $\Pi(S_i, \psi) \geq_p \Pi(S_\infty, \psi)$, per la Definizione 2.4.7, φ è ridondante anche in S_∞ su $\mathcal{T}$.

(b) se $S_i = S \cup \{\varphi\}$ ed $S_{i+1} = S \cup \{\varphi'\}$, questo passo è prova-riducente, cioè $\forall \psi \in \mathcal{T}$, $\Pi(S_i, \psi) \geq_p \Pi(S_{i+1}, \psi)$ ed $\exists \psi \in \mathcal{T}$ tale che $\Pi(S_i, \psi) >_p \Pi(S_{i+1}, \psi)$. Se φ stessa rappresenta una prova minimale di φ in S_i, allora esiste una prova minimale $\Pi(S_{i+1}, \varphi)$ in S_{i+1} tale che $\varphi \geq_p \Pi(S_{i+1}, \varphi)$. Dal momento che $\varphi \notin S_{i+1}$, $\Pi(S_{i+1}, \varphi)$ non è φ stessa e quindi $\varphi >_p \Pi(S_{i+1}, \varphi) \geq_p \Pi(S_\infty, \varphi)$. Se φ non rappresenta una prova minimale di φ in S_i, allora esiste una prova minimale $\Pi(S_i, \varphi)$ di φ in S_i tale che $\Pi(S_i, \varphi) <_p \varphi$ e quindi $\varphi >_p \Pi(S_i, \varphi) \geq_p \Pi(S_{i+1}, \varphi) \geq_p \Pi(S_\infty, \varphi)$. In entrambi i casi c'è una prova minimale $\Pi(S_\infty, \varphi)$ di φ in S_∞ tale che $\varphi >_p \Pi(S_\infty, \varphi)$ e per monotonicità e stabilità di $>_p$, $P[\varphi\sigma] >_p P[\Pi(S_\infty, \varphi)\sigma]$ per tutti i contesti di prova P e le sostituzioni σ. In altre parole, φ non è coinvolto in nessuna prova minimale in S_∞ di un teorema di $\mathcal{T}$, perché ogni occorrenza di φ in una prova può essere rimpiazzata da una prova $\Pi(S_\infty, \varphi)$ più piccola di φ stessa. Segue che φ è ridondante in S_∞ su $\mathcal{T}$.

2. Se φ è ridondante in S_j su $\mathcal{T}$, poiché $\forall \psi \in \mathcal{T}$, $\Pi(S_j, \psi) \geq_p \Pi(S_\infty, \psi)$ per la Definizione 2.4.7, φ è ridondante in S_∞ su $\mathcal{T}$. □

Questo teorema generalizza due classici risultati:

Teorema 4.1.2 (Knuth and Bendix 1970) [62], (Huet 1981) [52], (Bachmair, Dershowitz and Hsiang 1986) [8] *Se una derivazione*

$$(E_0; \emptyset) \vdash_{KB} (E_1; \emptyset) \vdash_{KB} \ldots (E_i; \emptyset) \vdash_{KB} \ldots$$

prodotta dalla procedura di completamento di Knuth-Bendix non fallisce ed è uniformemente fair *sul dominio $\mathcal{T}$ di tutte le equazioni, allora E_∞ è un sistema di riscrittura di termini* confluente.

La procedura di completamento di Knuth-Bendix fallisce se una equazione non orientata persiste. Se una derivazione del completamento di Knuth-Bendix non fallisce, tutte le equazioni persistenti sono orientate in regole di riscrittura in base a un ordinamento di riduzione e quindi E_∞ è un sistema di riscrittura con la proprietà di terminazione. Per il Teorema 4.1.1, E_∞ è saturato, cioè localmente confluente. Per il lemma di Newman [34], un sistema di riscrittura con la proprietà di terminazione è confluente se e solo se è localmente confluente. Dunque E_∞ è confluente.

Teorema 4.1.3 (Hsiang and Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989) [14] *Se una derivazione*

$$(E_0; \emptyset) \vdash_{UKB} (E_1; \emptyset) \vdash_{UKB} \ldots (E_i; \emptyset) \vdash_{UKB} \ldots$$

prodotta dalla procedura di completamento Unfailing Knuth-Bendix è uniformemente fair *sul dominio $\mathcal{T}$ di tutte le equazioni ground, allora E_∞ è un insieme di equazioni* ground confluente.

Per questo secondo risultato rammentiamo che dato un insieme di equazioni E, $s \to_E t$ se $s \leftrightarrow_E t$ e $s \succ t$ in un ordinamento di riduzione $\succ$. La procedura Unfailing Knuth-Bendix assume che $\succ$ sia

un ordinamento di semplificazione completo. Siccome un ordinamento di semplificazione completo è totale sui termini ground, $\leftrightarrow_{E_\infty} = \rightarrow_{E_\infty} \cup \leftarrow_{E_\infty}$ vale per termini ground e E_∞ è Church-Rosser sui termini ground se e solo se è ground confluente. Poché un ordinamento di semplificazione completo è ben fondato, E_∞ ha la proprietà di terminazione sui termini ground. Il dominio $\mathcal{T}$ della procedura Unfailing Knuth-Bendix è l'insieme delle equazioni ground. Per il Teorema 4.1.1, E_∞ è saturato su $\mathcal{T}$, ovvero è localmente confluente sui termini ground. Per il lemma di Newman, E_∞ è ground confluente e perciò Church-Rosser sui termini ground.

La proprietà di Church-Rosser sui termini ground è importante perché $E \models \forall \bar{x} s \simeq t$ se e solo se $\hat{s} \leftrightarrow_E^* \hat{t}$. Se E è Church-Rosser sui termini ground, $\hat{s} \leftrightarrow_E^* \hat{t}$ se e solo se $\hat{s} \rightarrow_E^* \circ \leftarrow_E^* \hat{t}$ e quindi $E \models \forall \bar{x} s \simeq t$ è decidibile via riduzione ben fondata mediante E. Questo ci introduce al tema della prossima sezione.

4.2 Procedure di decisione

Dal momento che aggiungere formule a una presentazione saturata è ridondante, è possibile provare teoremi da una presentazione saturata eseguendo solo passi di inferenza sull'obbiettivo. Una derivazione dove ad ogni passo un nuovo obbiettivo viene ottenuto da quello dato e da un assioma nella presentazione in input è detta una derivazione *lineare input* [24]. Se, inoltre, tutti i passi sono strettamente prova-riducenti, abbiamo una derivazione *lineare input riducente*:

Definizione 4.2.1 *Una derivazione*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

prodotta da una procedura di completamento $\mathcal{C}$ è

- *una derivazione* lineare input *se tutti i suoi passi sono passi di inferenza sull'obbiettivo,*
- *una derivazione* lineare input riducente *se tutti i suoi passi sono passi di inferenza sull'obbiettivo strettamente prova-riducenti.*

Poiché un ordinamento su prove è ben fondato, la terminazione di una derivazione lineare input riducente è garantita. Diciamo che una presentazione è *strettamente prova-riducente* se tutti i passi di inferenza sull'obbiettivo in quella presentazione sono strettamente prova-riducenti:

Definizione 4.2.2 *Sia $\mathcal{C}$ una procedura di completamento su dominio $\mathcal{T}$. Una presentazione S di una teoria è* strettamente prova-riducente *su $\mathcal{T}$ se $\forall \varphi \in \mathcal{T}$ ogni passo di inferenza sull'obbiettivo $(S; \varphi) \vdash_{\mathcal{C}} (S; \varphi')$ è strettamente prova-riducente.*

Per la procedura Unfailing Knuth-Bendix, la Semplificazione dell'obbiettivo è essa stessa una regola di inferenza strettamente prova-riducente, come abbiamo visto nell'Esempio 2.4.1. Quindi, ogni insieme di equazioni è banalmente una presentazione strettamente prova-riducente e ogni derivazione che si componga soltanto di passi di semplificazione è una derivazione lineare input riducente. E' così perchè la Semplificazione è ristretta da un ordinamento di semplificazione completo in modo tale che ogni sequenza di passi di semplificazione sia ben fondata.

Per il teorema seguente ricordiamo che dato un insieme di regole di inferenza I, denotiamo con I_e il sottoinsieme delle regole di espansione sulla presentazione e con I_t il sottoinsieme delle regole sull'obbiettivo:

Teorema 4.2.1 *Sia I un insieme di regole di inferenza refutazionalmente completo, tale che il sottoinsieme $I_e \cup I_t$ è anch'esso refutazionalmente completo. Se una presentazione S di una teoria è* saturata *e* strettamente prova-riducente *su $\mathcal{T}$, allora $\forall \varphi_0 \in \mathcal{T}$, esiste una derivazione* lineare input riducente

$$(S;\varphi_0)\vdash_{\mathcal{I}}(S;\varphi_1)\vdash_{\mathcal{I}}\ldots\vdash_{\mathcal{I}}(S;\varphi_i)\vdash_{\mathcal{I}}\ldots.$$

Dimostrazione: poiché $I_e \cup I_t$ è refutazionalmente completo, posiamo restringere l'attenzione alle derivazioni mediante $I_e \cup I_t$. Innanzitutto mostriamo che nessuna regola in I_e si applica ad S. Se una regola di espansione f deriva S' da S, allora, sicome f è prova-riducente, c'è una $\psi \in \mathcal{T}$ tale che $\Pi(S,\psi) >_p \Pi(S',\psi)$. Poiché S è saturata su $\mathcal{T}$, questo è impossibile. Allora, la derivazione contiene soltanto passi di inferenza sull'obbiettivo e quindi è una derivazione lineare input. Inoltre, i passi di inferenza sull'obbiettivo in S sono strettamente prova-riducenti per ipotesi e quindi la derivazione è lineare input riducente. □

Nel caso equazionale, un insieme ground confluente di equazioni è saturato e strettamente prova-riducente e perciò produce derivazioni lineari input fatte soltanto di passi di semplificazione ben fondati.

Definizione 4.2.3 *Sia $\mathcal{C}$ una procedura di completamento completa sul dominio $\mathcal{T}$. Una presentazione S di una teoria è una* procedura di decisione *per $Th(S) \cap \mathcal{T}$, se per tutte le $\varphi_0 \in \mathcal{T}$, è garantito che la derivazione*

$$(S;\varphi_0)\vdash_{\mathcal{C}}(S;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S;\varphi_i)\vdash_{\mathcal{C}}\ldots,$$

termina a uno stadio k, per un $k > 0$, tale che $\varphi_k = true$ se $\varphi_0 \in Th(S)$ e $\varphi_k \neq true$ se $\varphi_0 \notin Th(S)$.

Consideriamo la presentazione S come un algoritmo, che, se interpretato dalla procedura $\mathcal{C}$, decide la validità delle formule in $\mathcal{T}$ nella teoria di S. Non si esegue nessuna inferenza su S stesso e l'esecuzione termina a uno stadio k se nessuna regola di inferenza sull'obbiettivo si applica a φ_k.

Lemma 4.2.1 *Sia $\mathcal{C}$ una procedura di completamento su dominio $\mathcal{T}$ con un insieme I di regole di inferenza refutazionalmente completo e un piano di ricerca fair Σ. Se una presentazione S di una teoria è tale che per tutte le $\varphi_0 \in \mathcal{T}$, esiste una derivazione* lineare input riducente

$$(S;\varphi_0)\vdash_{\mathcal{I}}(S;\varphi_1)\vdash_{\mathcal{I}}\ldots\vdash_{\mathcal{I}}(S;\varphi_i)\vdash_{\mathcal{I}}\ldots,$$

la presentazione S è una procedura di decisione *per $Th(S) \cap \mathcal{T}$.*

Dimostrazione: l'esistenza di derivazioni lineari input riducenti da $(S;\varphi_0)$ garantisce che l'I-albero con radice in $(S;\varphi_0)$ contiene cammini finiti, dal momento che la terminazione di una derivazione lineare input riducente è assicurata dalla ben fondatezza di $>_p$. Allora, la fairness del piano di ricerca Σ garantisce che per tutte le $\varphi_0 \in \mathcal{T}$, la derivazione computata da $(S;\varphi_0)$ termina a uno

stadio k. O φ_k è *true* o φ_k non è *true*. Per la completezza di I, φ_k è *true* se e solo se $\varphi_0 \in Th(S)$. Quindi, S è una procedura di decisione per $Th(S) \cap \mathcal{T}$. □

Corollario 4.2.1 *Una presentazione S di una teoria che è* saturata *e* strettamente prova-riducente *su $\mathcal{T}$ è una procedura di decisione per $\mathcal{T} \cap Th(S)$.*

Dimostrazione: segue dal Teorema 4.2.1 e dal Lemma 4.2.1. □

Possiamo finalmente caratterizzare una procedura di completamento come un *generatore di procedure di decisione*:

Definizione 4.2.4 *Una procedura di completamento $\mathcal{C}$ ha la proprietà di* monotonicità forte *se per tutti i passi $(S;\varphi) \vdash_{\mathcal{C}} (S';\varphi')$, è $Th(S') = Th(S)$.*

Corollario 4.2.2 *Sia $\mathcal{C}$ una procedura di completamento su dominio $\mathcal{T}$ tale che*

- *l'insieme I delle regole di inferenza è refutazionalmente completo,*
- *il piano di ricerca Σ è uniformemente fair e*
- *la procedura ha la proprietà di monotonicità forte.*

Per tutte le presentazioni S_0, se il limite S_∞ della derivazione

$(S_0;\emptyset) \vdash_{\mathcal{C}} (S_1;\emptyset) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i;\emptyset) \vdash_{\mathcal{C}} \ldots$

è strettamente prova-riducente su $\mathcal{T}$, allora S_∞ è una procedura di decisione per $\mathcal{T} \cap Th(S_0)$.

Dimostrazione: per il Teorema 4.1.1, S_∞ è saturata su $\mathcal{T}$. Poiché è strettamente prova-riducente su $\mathcal{T}$, S_∞ è una procedura di decisione per $\mathcal{T} \cap Th(S_\infty)$ per il Corollario 4.2.1. Per la proprietà di monotonicità forte, $Th(S_\infty) = Th(S_0)$ e perciò S_∞ è una procedura di decisione per $\mathcal{T} \cap Th(S_0)$. □

In logica equazionale, un insieme di equazioni ground confluente E è una *procedura di decisione* per la validità dei teoremi di tipo $\forall \bar{x} s \simeq t$.

Condizioni sufficienti affinché una presentazione in logica di Horn con uguaglianza sia strettamente prova-riducente sono state date in [63] e [16]:

Definizione 4.2.5 (Kounalis and Rusinowitch 1988) [63] *Una clausola di Horn $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ è* ground-preservante *se le due condizioni seguenti valgono:*

- *tutte le variabili che compaiono in un letterale negato compaiono anche in A e*
- *se A è un'equazione $s \simeq t$, o $s \simeq t$ può essere orientata in una regola di riscrittura o s e t hanno lo stesso insieme di variabili.*

Teorema 4.2.2 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *Una presentazione saturata di clausole ground-preservanti in logica di Horn con uguaglianza è una procedura di decisione per obbiettivi di tipo $B_1 \wedge \ldots \wedge B_m$, dove ogni B_i è un letterale ground positivo.*

Si assume che la presentazione sia saturata dalle regole di espansione delle procedure di completamento per la logica di Horn con uguaglianza date in [63] e [16]. Questo risultato si spiega come segue. La richiesta che le clausole siano ground-preservanti ha lo scopo di assicurare che ogni volta che si applica un passo di inferenza a una clausola nell'insieme saturato e a un obbiettivo ground, il nuovo obbiettivo così generato è a sua volta ground. Inoltre, le regole di risoluzione e di paramodulazione in [63] e [16] sono *ordinate*, cioè sono ristrette da un dato ordinamento di semplificazione completo su termini e letterali in modo tale che a ogni passo un letterale ground nell'obbiettivo sia sostituito da un insieme di letterali ground più piccoli. Segue che la risoluzione ordinata e la paramodulazione ordinata tra una clausola ground-preservante e un obbiettivo ground sono strettamente prova-riducenti. Una presentazione contenente soltanto clausole ground-preservanti è quindi strettamente prova-riducente per obbiettivi ground e se è anche saturata, è una procedura di decisione per il Teorema 4.2.1, il Lemma 4.2.1 e il Corollario 4.2.1.

La restrizione a clausole ground-preservanti è abbastanza forte. Per esempio la clausola di Horn $T(x, y) \vee \neg R(x, z) \vee \neg T(z, y)$ nella definizione della chiusura transitiva T di una relazione R non è ground-preservante, a causa della variabile z. Non è sorprendente, dal momento che poche teorie hanno una procedura di decisione.

Capitolo 5

Il teorema di Knuth-Bendix-Huet

Abbiamo visto che se il piano di ricerca è fair, una procedura di completamento è una procedura di semidecisione e se il piano di ricerca è uniformemente fair, una procedura di completamento è un generatore di procedure di decisione. In questo capitolo mostriamo la relazione tra queste due interpretazioni del completamento. In particolare, mostriamo che una procedura di completamento che è in grado di generare procedure di decisione è in sè una procedura di semidecisione. Questo teorema generalizza il risultato fondamentale in [52], dove l'interpretazione del completamento come procedura di semidecisione comparve per la prima volta.

5.1 Il teorema di Knuth-Bendix-Huet generalizzato

Cominciamo generalizzando la nozione di *fallimento* di una derivazione prodotta dal completamento di Knuth-Bendix:

Definizione 5.1.1 *Una derivazione uniformemente fair*

$(S_0;\emptyset)\vdash_{\mathcal{C}}(S_1;\emptyset)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\emptyset)\vdash_{\mathcal{C}}\ldots$

prodotta da una procedura di completamento $\mathcal{C}$ su dominio $\mathcal{T}$ fallisce *se per qualche $\varphi\in\mathcal{T}\cap Th(S_\infty)$ non esiste una derivazione lineare input riducente da $(S_\infty;\varphi)$.*

Nel caso equazionale una derivazione lineare input riducente è una derivazione fatta di soli passi di semplificazione ben fondati. Equivalentemente, una derivazione lineare input riducente è una derivazione che computa una prova di riscrittura $s\rightarrow^*\circ\leftarrow^* t$.

Una derivazione Knuth-Bendix fallisce se e solo se genera un'equazione persistente che non può essere orientata dal dato ordinamento di riduzione. L'esistenza di una tale equazione $l\simeq r\in E_\infty$ è una condizione necessaria e sufficiente per il fallimento. E' sufficiente perchè $l\simeq r$ è essa stessa un teorema per cui non c'é una prova di riscrittura in E_∞: $l\leftrightarrow_{l\simeq r} r$ non è una prova di riscrittura ed l ed r sono E_∞-irriducibili, dal momento che $l\simeq r$ è persistente. E' necessaria perchè se c'è un'equazione $s\simeq t\in Th(E_\infty)$ che non ha una prova di riscrittura in E_∞, allora $\Pi(E_\infty, s\simeq t)$ contiene o un picco $\leftarrow\circ\rightarrow$ o un passo non orientato $\leftrightarrow$. Poiché la derivazione è uniformemente fair, E_∞ è localmente confluente e una prova minimale in E_∞ non contiene

picchi. Allora $\Pi(E_\infty, s \simeq t)$ contiene un passo non orientato, ovvero E_∞ contiene un'equazione non orientata.

I fallimenti sono gestiti dal piano di ricerca. Un piano di ricerca per Knuth-Bendix può decretare un fallimento non appena una equazione non orientabile viene generata oppure quando nessuna regola di inferenza è applicabile e l'insieme corrente di dati contiene equazioni non orientate.

Il metodo Unfailing Knuth-Bendix non fallisce perché assume un ordinamento totale sui termini ground e il suo dominio è ristretto alle equazioni ground. Siccome il dominio è ristretto alle equazioni ground, solo le prove equazionali ground in E_∞ sono rilevanti. Tutti i passi di prova ground sono orientabili, perché l'ordinamento è totale sui termini ground. Poiché la derivazione è uniformemente fair, E_∞ è localmente confluente sui termini ground e le prove ground minimali in E_∞ non contengono picchi, quindi per tutti i teoremi $\forall \bar{x} s \simeq t \in Th(E_\infty)$ c'è una prova di riscrittura in E_∞.

In logica di Horn con uguaglianza, sappiamo dal Teorema 4.2.2 che se S_∞ è saturata e tutte le sue clausole sono ground-preservanti, allora per tutte le $\varphi \in \mathcal{T} \cap Th(S_\infty)$ c'è una derivazione lineare input riducente da $(S_\infty; \varphi)$, dove $\mathcal{T}$ è il dominio di tutte le congiunzioni di letterali ground positivi. Per la definizione di fallimento data sopra, se una derivazione uniformemente fair fallisce, allora per qualche $\varphi \in \mathcal{T} \cap Th(S_\infty)$ non c'è una derivazione lineare input riducente da $(S_\infty; \varphi)$. Allora o S_∞ non è saturata o S_∞ contiene una clausola che non è ground-preservante. Dal momento che la derivazione è uniformemente fair, S_∞ è saturata. Segue che S_∞ contiene una clausola che non è ground-preservante. In altre parole, una violazione del requisito che le clausole siano ground-preservanti è una condizione necessaria per il fallimento. Tuttavia, non è sufficiente, come mostra l'esempio seguente:

Esempio 5.1.1 *L'insieme di input S_0 è dato dalle clausole che definiscono la chiususra transitiva T di una relazione R*

$T(x, y) \vee \neg R(x, y)$,

$T(x, y) \vee \neg R(x, z) \vee \neg T(z, y)$

e da alcune coppie in R

$R(a, b)$,

$R(b, c)$,

$R(c, d)$.

Assumiamo la precedenza $R > T > d > c > b > a$ e un ordinamento su termini e letterali come definito nella Sezione 2.1. La presentazione saturata computata via risoluzione ordinata contiene, tra le altre, tutte le clausole di input e tutti i fatti su T che possono essere dedotti dall'input:

$T(a, b)$,

$T(b, c)$,

$T(c, d)$,

$T(a, c)$,

$T(b, d)$,

$T(a, d)$.

La seconda clausola nell'input non è ground-preservante e quindi l'insieme saturato contiene una clausola che non è ground-preservante. Tuttavia, la presentazione saturata contiene tutti i fatti veri sulle relazioni R e T, così che ogni obbiettivo dato da letterali ground positivi può essere provato con una derivazione lineare input riducente che include soltanto passi di risoluzione ordinata ground tra l'obbiettivo e i fatti nell'insieme saturato.

Dal momento che la proprietà che le clausole siano ground-preservanti è una condizione sufficiente, ma non necessaria affinché una presentazione saturata di clausole di Horn dia derivazioni lineari input riducenti, segue che una violazione di tale requisito è una condizione necessaria, ma non sufficiente per il fallimento. In base alle nostre conoscenze, il problema di trovare una caratterizzazione necessaria e sufficiente per il fallimento in logica di Horn con uguaglianza è aperto.

Usiamo la nozione di fallimento nel teorema seguente:

Teorema 5.1.1 *Sia $\mathcal{C}$ una procedura di completamento su dominio $\mathcal{T}$ tale che*

- *l'insieme I delle regole di inferenza è refutazionalmente completo,*
- *il piano di ricerca Σ è fair e uniformemente fair e*
- *la procedura ha la proprietà di monotonicità forte.*

Per tutte le derivazioni

$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots,$

se $\varphi_0 \in Th(S_0)$ e la derivazione non fallisce, allora esiste un $k \geq 0$ tale che φ_k è la clausola true.

Dimostrazione: siccome Σ è uniformemente fair, S_∞ è saturata per il Teorema 4.1.1. Per la monotonicità forte, è $Th(S_\infty) = Th(S_0)$. Poiché la derivazione non fallisce, per tutte le $\varphi_0 \in \mathcal{T} \cap Th(S_0)$, esiste una derivazione lineare input riducente che termina con successo

$(S_\infty; \varphi_0) \vdash_I (S_\infty; \varphi_1) \vdash_I \ldots \vdash_I (S_\infty; true)$.

Questa derivazione è finita e pertanto coinvolge solo un sottoinsieme finito S di S_∞. Dal momento che S_∞ è l'insieme di tutte le formule persistenti generate durante la derivazione, S è un insieme finito di formule persistenti. Segue che esiste un S_j, per qualche $j \geq 0$, tale che $S \subseteq S_j$ e la derivazione lineare input riducente in S_∞ di cui sopra può essere eseguita in S_j:

$(S_j; \varphi_0) \vdash_I (S_j; \varphi_1) \vdash_I \ldots \vdash_I (S_j; true)$.

L'I-albero con radice in $(S_0; \varphi_0)$ contiene il cammino

$(S_0; \varphi_0) \vdash_I \ldots \vdash_I (S_j; \varphi_0) \vdash_I (S_j; \varphi_1) \vdash_I \ldots \vdash_I (S_j; true)$.

Più in generale, l'I-albero con radice in $(S_0; \varphi_0)$ contiene nodi di successo. Allora, per la fairness del piano di ricerca, la derivazione computata raggiunge un tale nodo. □

Il teorema non asserisce che il cammino

$(S_0; \varphi_0) \vdash_I \ldots \vdash_I (S_j; \varphi_0) \vdash_I (S_j; \varphi_1) \vdash_I \ldots \vdash_I (S_j; true)$

è quello attualmente computato dalla procedura. Questo non è vero in generale. Una ragione è che la derivazione effettivamente computata può includere passi di inferenza sull'obbiettivo che applicano elementi non persistenti della presentazione.

Fairness e fairness uniforme sono richieste entrambe: la fairness uniforme assicura che il limite sia saturato, mentre la fairness garantisce che i necessari passi di inferenza sull'obbiettivo siano prima o poi eseguiti. La fairness uniforme non implica nulla sull'applicazione delle regole di inferenza sull'obbiettivo.

Questo teorema generalizza il *teorema di Knuth-Bendix-Huet*:

Teorema 5.1.2 (Huet 1981) [52] *Per tutte le derivazioni uniformemente fair e fair*

$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{KB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{KB} \ldots \vdash_{KB} (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{KB} \ldots$

prodotte dalla procedura di completamento di Knuth-Bendix, se $\forall \bar{x} s_0 \simeq t_0 \in Th(E_0)$ *e la derivazione non fallisce, allora esiste un* $k \geq 0$ *tale che* $\hat{s}_k = \hat{t}_k$.

Nella formulazione originale di questo teorema, dalle ipotesi date segue che esiste un $k \geq 0$ tale che $\hat{s}_0 \rightarrow^*_{E_0} \circ \rightarrow^*_{E_1} \ldots \rightarrow^*_{E_k} \circ \leftarrow^*_{E_k} \ldots \leftarrow^*_{E_1} \circ \leftarrow^*_{E_0} \hat{t}_0$. Nel nostra versione, i passi di semplificazione di questa prova di riscrittura sono i passi di inferenza sull'obbiettivo eseguiti durante la derivazione. Lo stesso risultato vale per Unfailing Knuth-Bendix, senza l'assunzione che la derivazione non fallisca:

Teorema 5.1.3 (Hsiang Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989 [14] *Per tutte le derivazioni uniformemente fair e fair*

$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots \vdash_{UKB} (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$

prodotte dalla procedura di completamento di Unfailing Knuth-Bendix, se $\forall \bar{x} s_0 \simeq t_0 \in Th(E_0)$, *esiste un* $k \geq 0$ *tale che* $\hat{s}_k = \hat{t}_k$.

Il Teorema 5.1.1 può essere considerato la prima generalizzazione di questi due classici risultati. Lo studio delle presentazioni saturate per la logica di Horn con uguaglianza in [63, 16] non realizza completamente l'obbiettivo di generalizzare il teorema di Knuth-Bendix-Huet, perchè non collega il processo di saturazione a quello di semidecisione. Il concetto fondamentale nel teorema di Knuth-Bendix-Huet è che se il limite di una derivazione è saturato, ogni obbiettivo vero può essere provato durante la derivazione, indipendentemente dal fatto che il limite sia finito o no. La capacità della procedura di completamento di *semidecidere* la validità degli obbiettivi è un effetto collaterale del processo di saturazione, o, meglio ancora, il processo di saturazione stesso *è* un processo di semidecisione.

Sia [63] che [16] concentrano l'attenzione sulle proprietà di un insieme saturato finito. Il processo di saturazione è concepito non come un processo di semidecisione, ma come un processo di *compilazione* per generare un insieme saturato finito. La dimostrazione di teoremi viene quindi considerata come un processo a due fasi: prima compilare la data presentazione in una saturata e finita e poi provare i teoremi nella presentazione saturata. In conseguenza, una motivazione per lo studio di contrazione, ridondanza e restrizioni basate su ordinamenti per le regole di espansione in [63] e specialmente in [16] è la terminazione della saturazione. Il fine di questi studi è di limitare

il sistema di inferenza, pur salvandone la completezza, in modo tale che presentazioni saturate e finite possano essere ottenute per un numero relativamente alto di teorie significative.

Il nostro approccio è diverso. Preferiamo interpretare il completamento come una procedura di semidecisione e studiamo contrazione, ridondanza e restrizioni basate su ordinamenti per le regole di espansione al fine di rendere il processo di semidecisione più efficiente. Questa diversa prospettiva spiega perché abbiamo proposto una nuova definizione di fairness. Nell'approccio a due fasi la fairness uniforme è irrinunciabile, perché è necessaria per la saturazione. Invece, abbiamo provato nel Teorema 3.1.1 che una procedura di completamento è una procedura di semidecisione assumendo soltanto la fairness del piano di ricerca e la completezza refutazionale delle regole di inferenza. Queste sono ipotesi più deboli di quelle del teorema di Knuth-Bendix-Huet. Questo risultato è importante per noi perché mostra che l'interpretazione del completamento come procedura di semidecisione può essere separata dalla interpretazione come generatore di insiemi saturati e quindi non è soltanto un effetto collaterale di quest'ultima. In altre parole, non è necessario generare al limite un insieme saturato per avere una procedura di semidecisione.

Tuttavia, ci occorre anche il nostro teorema generalizzato di Knuth-Bendix-Huet per mostrare che se fairness e fairness uniforme sono assunte entrambe, semidecisione e saturazione coesistono nello stesso processo. Il Teorema 5.1.1 è generale in quanto non dipende da una logica specifica. La sola nozione che dipende dalla logica è quella di derivazione lineare input riducente e quindi quella di fallimento. Come abbiamo già discusso, caratterizzazioni soddisfacenti di questi nozioni non sono note per logiche oltre la logica equazionale. Proponiamo il nostro teorema di Knuth-Bendix-Huet generalizzato come una sorta di paradigma, che può essere specializzato per diverse logiche dando opportune caratterizzazioni delle nozioni di derivazione lineare input riducente e di fallimento.

Capitolo 6

Procedure di completamento in logica equazionale

In questo capitolo diamo una nuova presentazione, nel quadro dei concetti sviluppati finora, di alcune procedure di completamento di tipo Knuth-Bendix per la logica equazionale. Tutte le strategie che presentiamo sono estensioni o variazioni della procedura Unfailing Knuth-Bendix. Diversamente dalle procedure di Knuth-Bendix e Unfailing Knuth-Bendix, queste variazioni non sono mai state presentate prima d'ora come insiemi di regole di inferenza in modo unitario. Queste procedure sono molto più interessanti di quelle standard dal punto di vista della dimostrazione di teoremi, perchè sono molto più efficienti e infatti sono state implementate con successo [1, 2, 3, 5]. In particolare questa famiglia di procedure include la strategia *Inequality Ordered-Saturation*, una versione avanzata della procedura di Unfailing Knuth-Bendix, che ha dato ottimi risultati sperimentali [3]. Dal momento che la presentazione in [3] mette a fuoco specialmente i risultati sperimentali, la nostra è la prima descrizione da un punto di vista teorico della strategia Inequality Ordered-Saturation.

6.1 Il metodo Unfailing Knuth-Bendix

La *procedura di Unfailing Knuth-Bendix* [48, 14] è una procedura di semidecisione per teorie equazionali. Una presentazione è un insieme di equazioni E_0 e un teorema è un teorema equazionale $\forall \bar{x} s_0 \simeq t_0$. Una derivazione di UKB ha la forma

$$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$$

dove denotiamo con $\hat{s}_0 \simeq \hat{t}_0$ una equazione che contiene soltanto variabili quantificate universalmente e quindi può essere considerata come una equazione ground. Una derivazione termina allo stadio k se $\hat{s}_k$ e $\hat{t}_k$ sono identici. Assumiamo che l'ordinamento di semplificazione completo $\succ$ sia tale che $\forall s, s \succ true$. A ogni passo del processo di completamento una coppia $(E_{i+1}; \hat{s}_{i+1} \simeq \hat{t}_{i+1})$ viene derivata dalla coppia $(E_i; \hat{s}_i \simeq \hat{t}_i)$ applicando una delle regole di inferenza seguenti:

- *Regole di inferenza sulla presentazione*:

  - *Semplificazione*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u = l\sigma \\ p \triangleright l \vee q \succ p[r\sigma]_u \end{array} \qquad p \succ p[r\sigma]_u$$
  - *Deduzione*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma = l\sigma$$
  - *Eliminazione*:
    $$\frac{(E \cup \{l \simeq l\}; \hat{s} \simeq \hat{t})}{(E; \hat{s} \simeq \hat{t})}$$
  - *Sussunzione funzionale*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright (l \simeq r)$$
- *Regole di inferenza sull'obbiettivo*:
  - *Semplificazione*:
    $$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u = l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$
  - *Eliminazione*:
    $$\frac{(E; \hat{s} \simeq \hat{s}\})}{(E; true)}$$

Abbiamo già presentato queste regole di inferenza negli Esempi 2.3.1, 2.4.1, 2.4.2, 2.4.3 e 2.4.4. Ci limitiamo ad osservare che le definizioni originali di Semplificazione e Deduzione in [48] hanno condizioni leggermente differenti. Semplificazione richiede che $l\sigma \succ r\sigma$ e Deduzione richiede che $p\sigma \not\preceq q\sigma$ ed $l\sigma \not\preceq r\sigma$. Adottiamo qui le condizioni date in [36], perchè pongono requisiti più deboli sulla Semplificazione e requisiti più forti sulla Deduzione rispetto a quelle originali. Tuttavia, può essere più costoso verificare queste condizioni, poiché richiedono di applicare sia la sostituzione che la ricostruzione del termine.

La Semplificazione è la più importante tra le regole di inferenza di UKB, perché é molto efficace nel ridurre il numero e la lunghezza delle equazioni generate. Un piano di ricerca per UKB dovrebbe dare alla Semplificazione la priorità più alta tra le regole di inferenza, in modo che l'obbiettivo e la presentazione siano sempre semplificate il più possibile. Un piano di ricerca con questa proprietà è detto *Simplification-first* [48]. Se la Semplificazione non viene applicata, la regola di Deduzione satura rapidamente lo spazio di memoria con equazioni, rendendo impossibile ottenere una prova in tempo ragionevole.

Al fine di caratterizzare la procedura UKB come una procedura di completamento, definiamo un ordinamento su prove $>_{UKB}$ per confrontare le prove $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i)$. Usiamo l'ordinamento $>_u$ introdotto nell'Esempio 2.2.2. Quindi $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(E_j, \hat{s}_j \simeq \hat{t}_j)$ se e solo se $(\{\hat{s}_i, \hat{t}_i\}, \hat{s}_i \leftrightarrow^*_{E_i} \hat{t}_i) >_u (\{\hat{s}_j, \hat{t}_j\}, \hat{s}_j \leftrightarrow^*_{E_j} \hat{t}_j)$.

Lemma 6.1.1 *Le regole di inferenza sulla presentazione della procedura UKB sono riducenti.*

Dimostrazione: Deduzione e Semplificazione sono prova-riducenti, Eliminazione e Sussunzione funzionale eliminano equazioni ridondanti:

- la prova per Deduzione è stata data nell'Esempio 2.4.2.
- La prova per Semplificazione è stata data nell'Esempio 2.4.3.

- Una equazione banale $l \simeq l$ è ridondante: nessuna prova minimale contiene un passo $s \leftrightarrow_{l \simeq l} s$ dal momento che la sottoprova data dal solo termine s è più piccola: $\{(s, l, s)\} >^{e}_{mul} \{\epsilon\}$, dove la tripla vuota ϵ è la complessità di s.
- La prova per Sussunzione funzionale è stata data nell'Esempio 2.4.4. □

Lemma 6.1.2 *Le regole di inferenza sull'obbiettivo della procedura UKB sono strettamente prova-riducenti.*

Dimostrazione: la prova per Semplificazione è stata data nell'Esempio 2.4.1. Per un passo di Eliminazione, abbiamo $\{\hat{s}_i, \hat{t}_i\} \succ_{mul} \{true\}$, poiché *true* è minore di ogni altro termine. Dunque $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(E_i, \hat{s}_{i+1} \simeq \hat{t}_{i+1})$. □

Possiamo quindi mostrare che UKB è una procedura di completamento:

Teorema 6.1.1 *La procedura Unfailing Knuth-Bendix è una procedura di completamento sul dominio $\mathcal{T}$ su tutte le equazioni ground.*

Dimostrazione: per tutte le presentazioni equazionali E_0 e per tutti gli obbiettivi $\hat{s}_0 \simeq \hat{t}_0$ la derivazione

$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$

ha le proprietà di monotonicità, rilevanza e riduzione. Monotonicità e rilevanza seguono dalla soundness delle regole di inferenza, provata in [52, 8, 11]. Riduzione segue dal Lemma 6.1.1 e dal Lemma 6.1.2. □

Se il piano di ricerca è *fair*, la procedura UKB è una procedura di semidecisione per teorie equazionali:

Teorema 6.1.2 (Hsiang and Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989) [14] *Un'equazione $\forall \bar{x} s \simeq t$ è un teorema di una teoria equazionale E se e solo se la procedura Unfailing Knuth-Bendix deriva* true *da $(E; \hat{s} \simeq \hat{t})$.*

Implementazioni della procedura Unfailing Knuth-Bendix sono descritte in [69, 1, 2, 18, 3, 5].

6.2 Estensioni: AC-UKB e le leggi di cancellazione

Operatori associativi e commutativi (AC) compaiono in molti problemi equazionali. Una funzione f è AC se soddisfa le equazioni

$f(f(x, y), z) \simeq f(x, f(y, z))$ *(associatività)* e

$f(x, y) \simeq f(y, x)$ *(commutatività).*

Gestire l'associatività e la commutatività come equazioni qualsiasi risulta essere molto inefficiente. La legge di commutatività non può essere orientata e quindi può generare, per mezzo della regola

di Deduzione, un numero molto alto di equazioni, che non è possibile semplificare. L'assioma di associatività può essere orientato in $f(f(x,y),z) \to f(x,f(y,z))$, ma ciò nonostante, può attivare la generazione di un numero molto alto, perfino infinito, di equazioni, come mostra l'esempio seguente:

Esempio 6.2.1 *Data la regola di associatività*

$$f(f(x,y),z) \to f(x,f(y,z))$$

e la regola di riscrittura

$$f(a,f(x,b)) \to f(x,b),$$

l'associatività si sovrappone sul sottotermine $f(x,b)$ nel lato sinistro della seconda regola, generando la coppia critica

$$f(f(x,y),b) \simeq f(a,f(x,f(y,b))),$$

che può essere semplificata ed orientata in

$$f(a,f(x,f(y,b))) \to f(x,f(y,b)).$$

L'associatività si sovrappone ancora sul sottotermine $f(y,b)$ della nuova regola, generando

$$f(a,f(x,f(y,f(z,b)))) \to f(x,f(y,f(z,b))).$$

Ora l'associatività si sovrappone su $f(z,b)$: in questo modo si può generare un numero infinito di coppie critiche.

E' possibile migliorare notevolmente l'efficienza della strategia UKB trattando associatività e commutatività non come equazioni in input, ma direttamente nelle regole di inferenza. La procedura UKB con associatività e commutatività *built in* nelle regole di inferenza è detta *AC-UKB* [1]. L'idea fondamentale è di sostituire l'identità sintattica con l'uguaglianza *modulo AC*. Sia AC un insieme di assiomi di associatività e commutatività. Due termini s e t sono uguali modulo AC, se $s \simeq t$ è un teorema di AC, scritto $s \leftrightarrow^*_{AC} t$. le regole di inferenza di UKB vengono modificate in modo tale che due termini uguali modulo AC sono considerati identici.

La prima modificazione consiste nel richiedere che l'ordinamento di semplificazione completo $\succ$ sia in un certo senso "compatibile" con l'uso dell'uguaglianza modulo AC al posto dell'identità. Più precisamente, questo requisito di "compatibilità" è una proprietà di *commutazione*. Date due relazioni R ed S, diciamo che R *commuta* con S se $S \circ R \subseteq R \circ S$, dove $\circ$ è la composizione di relazioni. Innanzitutto si richiede che l'ordinamento di semplificazione completo $\succ$ commuti con $\leftrightarrow^*_{AC}$: questo significa che per ogni due termini s e t, se esiste un terzo termine r tale che $s \leftrightarrow^*_{AC} r$ ed $r \succ t$, esiste anche un termine r' tale che $s \succ r'$ ed $r' \leftrightarrow^*_{AC} t$. In secondo luogo, si usano AC-matching ed AC-unificazione al posto di matching ed unificazione. Un termine t è un'istanza di un termine s *modulo AC* se c'è una sostituzione σ tale che $s\sigma \leftrightarrow^*_{AC} t$. Similmente, due termini s e t unificano *modulo AC* se c'è una sostituzione σ tale che $s\sigma \leftrightarrow^*_{AC} t\sigma$. Infine, l'ordinamento per contenimento stretto $\rhd$ è sostituito dall'ordinamento $\rhd_{AC}$, tale che $s \rhd_{AC} t$ se e solo se $s \rhd r$ ed $r \leftrightarrow^*_{AC} t$ per qualche termine r.

L'insieme della regole di inferenza della procedura UKB viene dunque modificato come segue:

- *Regole di inferenza sulla presentazione*:

- *Semplificazione*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \begin{array}{l} p|u \leftrightarrow^*_{AC} l\sigma \\ p \triangleright_{AC} l \vee q \succ p[r\sigma]_u \end{array} \qquad p \succ p[r\sigma]_u$$
- *Deduzione*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma \leftrightarrow^*_{AC} l\sigma$$
- *Estensione*:
$$\frac{(E \cup \{f(p,q) \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p,q) \simeq r, f(p,q,z) \simeq f(r,z)\}; \hat{s} \simeq \hat{t})} \begin{array}{l} f \text{ is } AC \\ f(p,q) \not\preceq r \end{array}$$
- *Eliminazione*:
$$\frac{(E \cup \{l \simeq l\}; \hat{s} \simeq \hat{t})}{(E; \hat{s} \simeq \hat{t})}$$
- *Sussunzione funzionale*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright_{AC} (l \simeq r)$$

- *Regole di inferenza sull'obbiettivo*:
  - *Semplificazione*:
$$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \begin{array}{l} \hat{s}|u \leftrightarrow^*_{AC} l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$
  - *Eliminazione*:
$$\frac{(E; \hat{s} \simeq \hat{s})}{(E; true)}$$

Si ottiene questo insieme di regole di inferenza da quello di UKB, usando l'uguaglianza modulo AC al posto dell'identità e aggiungendo una nuova regola di inferenza, detta *Estensione*. La regola di *Estensione* è una versione specializzata della regola di *Deduzione*, che computa sovrapposizioni delle equazioni in E sugli assiomi di associatività. In particolare, se $f(p,q) \simeq r$ è un'equazione in E, f è AC ed $f(p,q) \not\preceq r$, l'equazione $f(p,q) \simeq r$ si sovrappone banalmente sulla legge di associatività $f(f(x,y),z) \simeq f(x,f(y,z))$, generando la coppia critica $f(p,f(q,z)) \simeq f(r,z)$, che si scrive in forma *appiattita (flattened)* come $f(p,q,z) \simeq f(r,z)$. Queste coppie critiche sono dette *regole estese*. Generare le regole estese è sufficiente ad assicurare la completezza della procedura AC-UKB: non c'è bisogno di computare nessun'altra coppia critica tra E ed AC [70].

L'estensione di UKB ad AC-UKB è fattibile perchè si conoscono algoritmi per AC-matching ed AC-unificazione. Un algoritmo di AC-unificazione, il suo uso in una procedura di completamento e le regole estese apparvero per la prima volta in [76, 77, 70]. La correttezza dell'algoritmo di AC-unificazione venne provata in [37]. Trattamenti teorici generali per lavorare con equazioni modulo un insieme di assiomi A sono stati dati in [55] e in [9]. Per una rassegna di questi risultati rimandiamo a [34], ed a [57] per problemi di unifcazione.

E' possibile arricchire ulteriormente la procedura UKB o AC-UKB aggiungendo speciali regole di inferenza che implementano le *leggi di cancellazione*. Una funzione f è *cancellabile a destra* se soddisfa la *legge di cancellazione destra*

$$\forall x, y, z \;\; f(x,y) = f(z,y) \;\supset\; x = z$$

La *legge di cancellazione sinistra* è definita simmetricamente. Le leggi di cancellazione possono ridurre considerevolmente la lunghezza delle equazioni. Esse sono implementate come regole di

inferenza nel modo seguente [49]:

Cancellazione 1:

$$\frac{(E \cup \{f(p,u) \simeq f(q,v)\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p,u) \simeq f(q,v), p\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})}\ u\sigma = v\sigma$$

Cancellazione 2:

$$\frac{(E \cup \{f(d_1,d_2) \simeq y\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(d_1,d_2) \simeq y, d_1\sigma \simeq x\}; \hat{s} \simeq \hat{t})}\ \begin{array}{ll} y \in V(d_1) & \sigma = \{y \mapsto f(x,d_2)\} \\ y \notin V(d_2) & x \text{ è una nuova variabile} \end{array}$$

Cancellazione 3:

$$\frac{(E \cup \{f(p_1,q_1) \simeq r_1, f(p_2,q_2) \simeq r_2\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p_1,q_1) \simeq r_1, f(p_2,q_2) \simeq r_2, p_1\sigma \simeq p_2\sigma\}; \hat{s} \simeq \hat{t})}\ \begin{array}{l} q_1\sigma = q_2\sigma \\ r_1\sigma = r_2\sigma \end{array}$$

Cancellazione 4:

$$\frac{(E \cup \{f(p,u) \simeq f(q,u)\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q\}; \hat{s} \simeq \hat{t})}$$

dove la funzione f è cancellabile a destra. Nella regola *Cancellazione 2*, se si applica la sostituzione $\sigma = \{y \mapsto f(x,d_2)\}$ alla data equazione, si ottiene $f(d_1\sigma, d_2) \simeq f(x,d_2)$, dal momento che y non compare in d_2. La legge di cancellazione riduce questa equazione a $d_1\sigma \simeq x$.

La regola *Cancellazione 4* non è necessaria per la completezza della strategia, poichè lo stesso effetto può essere ottenuto da un passo di *Cancellazione 1* con mgu vuoto seguito da un passo di Sussunzione funzionale. La si aggiunge per ragioni di efficienza.

Al fine di mostrare che la procedura UKB con le regole di inferenza per le leggi di cancellazione è una procedura di completamento, occorre provare che le regole di Cancellazione sono prova-riducenti. Adottiamo l'ordinamento su prove $>_{UKBC}$ definito come segue: un passo equazionale ground $s \simeq t$ giustificato da un'equazione $l \simeq r$ ha misura di complessità $(s, l\sigma, l, t)$, se s è $c[l\sigma]$, t è $c[r\sigma]$ ed $s \succ t$. Le misure di complessità sono confrontate dalla combinazione lessicografica $>^{ec}$ degli ordinamenti $\succ$, $\trianglerighteq$, $\trianglerighteq$ e $\succ$. Si confrontano le prove con la combinazione lessicografica $>_{uc}$ delle estensioni ai multiinsiemi $\succ_{mul}$ e $>^{ec}{}_{mul}$: $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKBC} \Pi(E_j, \hat{s}_j \simeq \hat{t}_j)$ se e solo se $(\{\hat{s}_i, \hat{t}_i\}, \hat{s}_i \leftrightarrow^*_{E_i} \hat{t}_i) >_{uc} (\{\hat{s}_j, \hat{t}_j\}, \hat{s}_j \leftrightarrow^*_{E_j} \hat{t}_j)$. La prova del Lemma 6.1.1 non viene alterata se si usa $>_{UKBC}$ al posto di $>_{UKB}$.

Lemma 6.2.1 *Le regole di* Cancellazione *sono prova-riducenti.*

Dimostrazione: mostriamo che se $(E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} (E_{i+1}; \hat{s}_i \simeq \hat{t}_i)$ è un passo di Cancellazione, allora se $\Pi(E_i, \hat{s} \simeq \hat{t}) \neq \Pi(E_{i+1}, \hat{s} \simeq \hat{t})$, cioè il passo di inferenza altera la prova di $\hat{s} \simeq \hat{t}$, $\Pi(E_i, \hat{s} \simeq \hat{t}) >^{ec}{}_{mul} \Pi(E_{i+1}, \hat{s} \simeq \hat{t})$.

- Un'applicazione della regola Cancellazione 1 a un'equazione $f(p,u) \simeq f(q,v)$ modifica ogni prova minimale in E_i che contenga un passo $s \leftrightarrow t$ tale che $s = c[f(p,u)\tau]$, $t = c[f(q,v)\tau]$ e $\tau \trianglerighteq \sigma$, dove $\trianglerighteq$ è l'ordinamento per sussunzione e σ è l'mgu tale che $u\sigma = v\sigma$ dell'applicazione di Cancellazione 1. Il passo $s \leftrightarrow_{f(p,u) \simeq f(q,v)} t$ ha complessità $(s, f(p,u)\tau, f(p,u), t)$, se $s \succ t$.

Nelle prove minimali in E_{i+1} il passo $s \leftrightarrow_{f(p,u) \simeq f(q,v)} t$ viene sostituito da un passo $s \leftrightarrow_{p\sigma \simeq q\sigma} t$ giustificato dalla nuova equazione $p\sigma \simeq q\sigma$ generata dall'applicazione di Cancellazione 1. Il passo $s \leftrightarrow_{p\sigma \simeq q\sigma} t$ ha complessità $(s, p\tau, p\sigma, t)$. Dal momento che $f(p,u)\tau \rhd p\tau$, segue che $\{(s, f(p,u)\tau, f(p,u), t)\} >^{ec}_{mul} \{(s, p\tau, p\sigma, t)\}$. Il caso in cui è $t \succ s$ è simmetrico.

- Un'applicazione della regola Cancellazione 2 ad un'equazione $f(d_1, d_2) \simeq y$ altera ogni prova minimale in E_i che contenga un passo $s \leftrightarrow t$ tale che $s = c[f(d_1, d_2)\tau]$, $t = c[y\tau]$ e $\tau \succeq \sigma$, dove σ è $\{y \mapsto f(x, d_2)\}$. Poiché $y \in V(d_1)$, abbiamo $f(d_1, d_2)\tau \succ y\tau$ per la proprietà sottotermine e quindi $s \succ t$ per monotonicità, così che il passo $s \leftrightarrow t$ ha complessità $(s, f(d_1, d_2)\tau, f(d_1, d_2), t)$. Nelle prove minimali in E_{i+1} il passo $s \leftrightarrow t$ viene rimpiazzato da un passo $s \leftrightarrow_{d_1\sigma \simeq x} t$ giustificato dalla nuova equazione $d_1\sigma \simeq x$ generata dall'applicazione di Cancellazione 2. Il passo $s \leftrightarrow_{d_1\sigma \simeq x} t$ ha complessità $(s, d_1\tau, d_1\sigma, t)$. Siccome $f(d_1, d_2)\tau \rhd d_1\tau$, segue che $\{(s, f(d_1, d_2)\tau, f(d_1, d_2), t)\} >^{ec}_{mul} \{(s, d_1\tau, d_1\sigma, t)\}$.
- Un'applicazione della regola Cancellazione 3 a due equazioni $f(p_1, q_1) \simeq r_1$ ed $f(p_2, q_2) \simeq r_2$ modifica ogni prova minimale in E_i che contenga una sottoprova $s \leftrightarrow u \leftrightarrow t$ tale che $s = c[f(p_1, q_1)\tau]$, $u = c[r_1\tau]$, $t = c[f(p_2, q_2)\tau]$ e $\tau \succeq \sigma$, dove σ è l'mgu tale che $q_1\sigma = q_2\sigma$ ed $r_1\sigma = r_2\sigma$ dell'applicazione di Cancellazione 3. Segue che $q_1\tau = q_2\tau$ e anche $r_1\tau = r_2\tau$. la sottoprova $s \leftrightarrow u \leftrightarrow t$ viene rimpiazzata in ogni prova minimale in E_{i+1} da un singolo passo $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ giustificato dalla nuova equazione $p_1\sigma \simeq p_2\sigma$ generata dall'applicazione di Cancellazione 3.
  1. Se $s \succ t \succ u$, la sottoprova $s \leftrightarrow u \leftrightarrow t$ ha complessità $\{(s, f(p_1, q_1)\tau, f(p_1, q_1), u), (t, f(p_2, q_2)\tau, f(p_2, q_2), u)\}$ e il passo $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ ha complessità $(s, p_1\tau, p_1\sigma, t)$. Dal momento che $f(p_1, q_1)\tau \rhd p_1\tau$, segue il risultato desiderato. Il caso in cui è $t \succ s \succ u$ è simmetrico.
  2. Se $s \succ u \succ t$, la sottoprova $s \leftrightarrow u \leftrightarrow t$ ha complessità $\{(s, f(p_1, q_1)\tau, f(p_1, q_1), u), (u, r_1\tau, r_1, t)\}$ e il passo $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ ha complessità $(s, p_1\tau, p_1\sigma, t)$. Poiché $f(p_1, q_1)\tau \rhd p_1\tau$, abbiamo il risultato. Il caso in cui è $t \succ u \succ s$ è simmetrico.
  3. Se $u \succ s \succ t$, la sottoprova $s \leftrightarrow u \leftrightarrow t$ ha complessità $\{(u, r_1\tau, r_1, s), (u, r_1\tau, r_1, t)\}$ e il passo $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ ha complessità $(s, p_1\tau, p_1\sigma, t)$. Siccome $u \succ s$, il risultato desiderato segue banalmente. Il caso in cui è $u \succ t \succ s$ è simmetrico. □

La procedura UKB o AC-UKB arricchita con le regole per la cancellazione è completa [49]. La maggior parte dei risultati sperimentali riportati in [1, 2, 3, 5] è stata ottenuta da AC-UKB con le regole di inferenza per la cancellazione.

6.3 La Inequality Ordered Saturation Strategy

La procedura UKB è completa, ma non è molto efficiente in generale. La principale causa di inefficienza è la regola di inferenza Deduzione, cioè la componente di forward reasoning di UKB. Questo problema è stato considerato dapprima nell'ambito dell'applicazione della procedura di Knuth-Bendix alla generazione di sistemi confluenti. Generare tutte le coppie critiche è una condizione sufficiente, ma non necessaria affinché il limite di una derivazione sia confluente. Un *criterio per coppie critiche* è un criterio per stabilire che certe coppie critiche non sono necessarie per ottenere un sistema confluente. Questi criteri sono stati studiati in [12].

Qui analizziamo invece il problema dal punto di vista della dimostrazione di teoremi. Criteri come quelli in [12] sono utili anche in dimostrazione di teoremi, ma non sono soddisfacenti perchè non sono orientati all'obbiettivo.

Tutti i passi di backward reasoning in una derivazione di UKB sono passi di Semplificazione, cioè passi strettamente prova-riducenti. D'altra parte, è garantito che un passo di Deduzione riduca la prova di qualche teorema, ma non necessariamente la prova dell'obbiettivo. La procedura UKB è inefficiente perché genera molte coppie critiche che non contribuiscono a provare l'obbiettivo.

Dunque, il goal è ridurre il numero delle coppie critiche generate o equivalentemente eseguire meno forward reasoning e più backward reasoning.

Per la parte di forward reasoning, un approccio potenzialmente interessante a questo problema consiste nel progettare piani di ricerca che generino prima quelle coppie critiche che si stima abbiano probabilità più alta di ridurre la prova dell'obbiettivo. Poichè l'effetto di una coppia critica sull'obbiettivo non può essere completamente determinato a priori, siffatti piani di ricerca si basano su *criteri euristici* che danno una misura di quanto ci si aspetta che una coppia critica sia utile rispetto al compito di semplificare l'obbiettivo. Alcuni esempi di queste euristiche sono dati in [3, 4].

Per la parte di backward reasoning, osserviamo che se l'obbiettivo $\hat{s}_i \simeq \hat{t}_i$ è semplificato il più possibile rispetto a E_i, $\hat{s}_i \simeq \hat{t}_i$ è minimale nell'ordinamento $\succ_{mul}$ tra tutte le equazioni ground E-equivalenti all'obbiettivo in input $s_0 \simeq t_0$, dove $E = \bigcup_{0 \leq j \leq i} E_j$. Se si adotta un piano di ricerca Simplification-first, UKB mantiene un'obbiettivo minimale. In conseguenza, può sembrare che non sia possibile ottenere alcun miglioramento dal lato dell'obbiettivo. Tuttavia, vedremo che non è così.

La nozione di obbiettivo minimale è relativa all'insieme parzialmente ordinato (poset) di obbiettivi che si prende in considerazione. Se assumiamo il poset delle equazioni ground ordinate da $\succ_{mul}$, $\hat{s}_i \simeq \hat{t}_i$ è minimale tra tutte le equazioni ground E-equivalenti all'obbiettivo in input $s_0 \simeq t_0$. La situazione cambia se assumiamo come poset degli obbiettivi il poset delle disgiunzioni di equazioni ground ordinate da un ordinamento $\succ'_{mul}$ definito come segue: $N_1 \succ'_{mul} N_2$ se $min(N_1) \succ_{mul} min(N_2)$, dove N_1 ed N_2 sono disgiunzioni di equazioni ground e $min(N)$ è la minima equazione in N secondo $\succ_{mul}$. Siccome le equazioni sono ground e l'ordinamento di semplificazione è totale sui termini ground, esiste un elemento minimo in una disgiunzione e questo ordinamento è ben definito. Inoltre, il poset delle equazioni è *monomorfo*[1] al poset delle disgiunzioni.

Mostriamo perché non è garantito che la componente di backward reasoning di UKB computi un obbiettivo minimale se si assume il poset delle disgiunzioni. Sia $(E_i; \hat{s}_i \simeq \hat{t}_i)$ lo stadio corrente in una derivazione di UKB e sia $l \simeq r$ una equazione non orientabile in E_i, tale che $\hat{s}_i|u = l\sigma$ per qualche posizione u e sostituzione σ, ma $\hat{s}_i \prec \hat{s}_i[r\sigma]_u$. In altre parole, un sottotermine di

[1]Dati due posets $\mathcal{P}_1 = (D_1, >_1)$ e $\mathcal{P}_2 = (D_2, >_2)$, un *monomorfismo* $h\colon \mathcal{P}_1 \to \mathcal{P}_2$ è una funzione iniettiva $h\colon D_1 \to D_2$ che preserva l'ordinamento: per tutti gli $x, y \in D_1$, $x >_1 y$ implica $h(x) >_2 h(y)$. La funzione che associa a un'equazione ground la disgiunzione data dall'equazione ground stessa è chiaramente un monomorfismo del poset delle equazioni ground nel poset delle disgiunzioni di equazioni ground, dal momento che l'elemento minimo in una disgiunzione data da una singola equazione è l'equazione stessa.

$\hat{s}_i$ è un'istanza di l, ma non si può applicare la Semplificazione perché $\hat{s}_i$ non sarebbe ridotto a un termine più piccolo. Tuttavia, assumiamo che l'obbiettivo $\hat{s}_i[r\sigma]_u \simeq \hat{t}_i$ venga generato ciò nonostante e che sia ridotto mediante semplificazione a un'equazione più piccola di $\hat{s}_i \simeq \hat{t}_i$, cioè $\hat{s}_i[r\sigma]_u \to^*_{E_i} \hat{s}'$, $\hat{t}_i \to^*_{E_i} \hat{t}'$ e $\{\hat{s}', \hat{t}'\} \prec_{mul} \{\hat{s}_i, \hat{t}_i\}$. Se queste condizioni valgono, abbiamo che la disgiunzione $\hat{s}_i \simeq \hat{t}_i \vee \hat{s}' \simeq \hat{t}'$ è più piccola della disgiunzione data da $\hat{s}_i \simeq \hat{t}_i$ solamente, nel poset delle disgiunzioni sopra definito. Dunque, se assumiamo il poset delle disgiunzioni come poset degli obbiettivi, non è vero che UKB mantiene un obbiettivo minimale.

Intuitivamente, considerare disgiunzioni di equazioni piuttosto che equazioni significa considerare più di un'equazione obbiettivo, e questo offre maggiori opportunità di trovare una prova in breve tempo. Al fine di lavorare su disgiunzioni di equazioni, bisogna aggiungere alla procedura UKB una regola di espansione, che prima o poi espanda l'obbiettivo in una disgiunzione di equazioni ground. Una regola di espansione siffatta deve soddisfare il requisito di rilevanza, così che provare la validità di una qualsiasi delle equazioni nella disgiunzione sia equivalente a provare l'obbiettivo in input $s_0 \simeq t_0$. Inoltre, l'applicazione di tale regola deve essere ristretta, in modo da evitare la generazione di un numero elevato di equazioni obbiettivo, che potrebbe rallentare, invece che facilitare, la ricerca della soluzione.

Questa nuova regola di inferenza consiste nella sovrapposizione di una equazione non orientabile su una equazione obbiettivo $\hat{s} \simeq \hat{t}$ al fine di generare una nuova equazione obbiettivo. Una nuova equazione obbiettivo appena generata viene prima semplificata il più possibile e tenuta solo se è minore di $\hat{s} \simeq \hat{t}$:

Saturazione ordinata:

$$\frac{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}\})}{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}, \hat{s}' \simeq \hat{t}'\})} \quad \hat{s}|u = l\sigma \quad \hat{s}[r\sigma]_u \to^*_E \hat{s}' \quad \hat{t} \to^*_E \hat{t}' \quad \{\hat{s}', \hat{t}'\} \prec_{mul} \{\hat{s}, \hat{t}\}$$

La *Saturazione ordinata* si applica se $\hat{s} \prec \hat{s}[r\sigma]_u$, dal momento che se $\hat{s} \succ \hat{s}[r\sigma]_u$ vale, si applicherebbe la Semplificazione. L'equazione obbiettivo $\hat{s}' \simeq \hat{t}'$ potrebbe avere una prova più corta delle altre equazioni obbiettivo. Non sappiamo quale ha la prova più corta. Le teniamo tutte per accrescere le nostre possibilità di raggiungere la prova il più presto possibile.

Dobbiamo modificare anche la regola di *Eliminazione*, poiché la computazione termina con successo non appena un'equazione nella disgiunzione è stata ridotta a un'equazione banale:

Eliminazione:

$$\frac{(E; N \cup \{\hat{s} \simeq \hat{s}\})}{(E; true)}$$

La procedura ottenuta aggiungendo la Saturazione ordinata ad UKB e modificando la regola di Eliminazione in questo modo, è detta *Inequality Ordered-Saturation strategy* (IOS) [3]. Una derivazione prodotta dalla strategia IOS ha la forma

$$(E_0; N_0) \vdash_{IOS} (E_1; N_1) \vdash_{IOS} \ldots \vdash_{IOS} (E_i; N_i) \vdash_{IOS} \ldots$$

dove l'insieme N_0 contiene il goal iniziale $\hat{s}_0 \simeq \hat{t}_0$ e allo stadio i, N_i è l'insieme corrente di equazioni obbiettivo. La derivazione termina a uno stadio k se N_k contiene un obbiettivo $\hat{s}_i \simeq \hat{t}_i$ tale che $\hat{s}_i$ e $\hat{t}_i$ sono identici e la clausola in N_k si riduce a $true$.

Al fine di mostrare che la strategia IOS è una procedura di completamento, assumiamo un ordinamento su prove $>_{IOS}$ definito nel modo seguente: $\Pi(E;N) >_{IOS} \Pi(E';N')$ se e solo se $\Pi(E;min(N)) >_{UKB} \Pi(E';min(N'))$. In altre parole la prova di una disgiunzione è rappresentata dalla prova dell'obbiettivo minimo nella disgiunzione.

Lemma 6.3.1 *La regola di* Saturazione ordinata *è prova-riducente.*

Dimostrazione: mostriamo che se $(E_i;N_i)\vdash_{IOS}(E_i;N_{i+1})$ è un passo di Saturazione ordinata, allora $\Pi(E_i,N_i) \geq_{IOS} \Pi(E_i,N_{i+1})$. Dal momento che $N_i \subset N_{i+1}$, $min(N_i) \succeq_{mul} min(N_{i+1})$ e abbiamo il risultato desiderato. □

Teorema 6.3.1 *La strategia Inequality Ordered-Saturation è una procedura di completamento.*

Dimostrazione: segue dal Teorema 6.1.1 e dal Lemma 6.3.1. □

La strategia IOS è stata implementata e si è osservato che ha prestazioni migliori della procedura UKB [3]. In pratica, poche equazioni obbiettivo vengono tenute, così che l'extra lavoro speso per gestirle risulta irrilevante rispetto al vantaggio di tenere più di un obbiettivo.

6.4 La S-strategy

La *S-strategy* [48] è un'estensione della procedura UKB alla logica dell'uguaglianza e dell'inuguaglianza. Una presentazione è un insieme di equazioni E_0 e un teorema φ è una formula $\bar{Q}\bar{x}\ s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$, dove $\bar{Q}\bar{x}$ è una qualsiasi sequenza di coppie quantificatore-variabile. Un teorema φ in questa forma viene trasformato, sostituendo tutte le variabili quantificate universalmente con costanti e lasciando cadere i quantificatori, in un obbiettivo $N_0 = s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$, dove tutte le variabili sono implicitamente quantificate esistenzialmente. Se φ è $\forall\bar{x} s_0 \simeq t_0$, N_0 è $\hat{s}_0 \simeq \hat{t}_0$ e la S-strategy si riduce alla procedura UKB. Una computazione ha la forma

$$(E_0;N_0)\vdash_S(E_1;N_1)\vdash_S \ldots \vdash_S(E_i;N_i)\vdash_S \ldots$$

dove $\forall i \geq 0$, E_i è un insieme di equazioni ed N_i è una disgiunzione di equazioni obbiettivo con variabili quantificate esistenzialmente. Una derivazione termina allo stadio k se N_k contiene un obbiettivo $s_i \simeq t_i$ i cui lati sono unificabili. L'insieme delle regole di inferenza di UKB viene modificato come segue:

- *Regole di inferenza sulla presentazione*:
  - *Semplificazione*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; N)} \quad \begin{array}{l} p|u = l\sigma \qquad\qquad p \succ p[r\sigma]_u \\ p \rhd l \vee q \succ p[r\sigma]_u \end{array}$$
  - *Deduzione*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; N)} \quad \begin{array}{l} p|u \notin X \qquad (p|u)\sigma = l\sigma \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array}$$
  - *Eliminazione*:
    $$\frac{(E \cup \{l \simeq l\}; N)}{(E; N)}$$

- *Sussunzione funzionale*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{l \simeq r\}; N)} \; (p \simeq q) \triangleright (l \simeq r)$$

- *Regole di inferenza sull'obbiettivo*:
  - *Semplificazione*:
$$\frac{(E \cup \{l \simeq r\}; N \cup \{s \simeq t\})}{(E \cup \{l \simeq r\}; N \cup \{s[r\sigma]_u \simeq t\})} \quad \begin{array}{l} s|u = l\sigma \\ s \succ s[r\sigma]_u \end{array}$$
  - *Deduzione*:
$$\frac{(E \cup \{l \simeq r\}; N \cup \{s \simeq t\})}{(E \cup \{l \simeq r\}; N \cup \{s \simeq t, s[r]_u\sigma \simeq t\sigma\})} \quad \begin{array}{l} s|u \notin X \qquad (s|u)\sigma = l\sigma \\ s\sigma \not\preceq s[r]_u\sigma \end{array}$$
  - *Eliminazione*:
$$\frac{(E; N \cup \{s \simeq t\})}{(E; true)} \; s\sigma = t\sigma$$

La regola di *Deduzione* si applica sia alla presentazione che all'obbiettivo. La regola di *Eliminazione* sull'obbiettivo è modificata perchè l'obbettivo contiene variabili: si ha una contraddizione quando i due lati di una equazione obbiettivo sono unificabili.

Al fine di caratterizzare la S-strategy come una procedura di completamento, definiamo l'ordinamento seguente: $\Pi(E_i, N_i) >_S \Pi(E_{i+1}, N_{i+1})$ se e solo se $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(\hat{E}_{i+1}, \hat{s}_{i+1} \simeq \hat{t}_{i+1})$, dove $\forall i \geq 0$, $\hat{E}_i \cup \{\hat{s}_i \neq \hat{t}_i\}$ è il minimo tra gli insiemi finiti insoddisfacibili di istanze ground di clausole in $E_i \cup \neg N_i$. Mostriamo che questo ordinamento è ben definito. Innanzitutto vediamo come si sceglie la coppia $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$. N_i è un teorema di E_i se e solo se $E_i \cup \neg N_i$ è insoddisfacibile, dove N_i è una disgiunzione di equazioni $s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$ con variabili quantificate esistenzialmente e quindi $\neg N_i$ è una congiunzione di inequazioni $s_0 \neq t_0 \wedge \ldots \wedge s_n \neq t_n$ con variabili quantificate universalmente. Per il Teorema di Herbrand [24], l'insieme $E_i \cup \neg N_i$ è insoddisfacibile se e solo se esiste un sottoinsieme finito di istanze ground di clausole in $E_i \cup \neg N_i$ che è insoddisfacibile. Dal momento che $\neg N_i$ è un insieme di inequazioni con variabili quantificate universalmente, un'istanza ground insoddisfacibile di $E_i \cup \neg N_i$ contiene almeno un'inequazione ground: $\hat{E}_i \cup \{\hat{s}_i \neq \hat{t}_i\}$ è il minimo siffatto insieme rispetto all'ordinamento $\succ_{mul}$. Poiché $\succ$ è totale sui termini ground, esiste un insieme minimo.

La suddetta definizione dell'ordinamento $>_S$ stabilisce che la complessità della prova $\Pi(E_i, N_i)$ è misurata dalla complessità della prova ground $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i)$ e che l'impatto dei passi di inferenza su $\Pi(E_i, N_i)$ è misurato dal loro impatto su $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i)$. Questo approccio è corretto se ad ogni passo di inferenza su $(E_i; N_i)$ corrisponde un passo di inferenza su $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ e vice versa. Per provarlo, ci occorre il lemma seguente, che riformula per la S-strategy il *Paramodulation Lifting Lemma*. Si ricorda che una sostituzione ground è E-irriducibile se non contiene nessuna coppia $\{x \mapsto t\}$ tale che t può essere semplificato da un'equazione in E.

Lemma 6.4.1 (Peterson 1983) [71], (Hsiang and Rusinowitch 1987) [50] *Se σ è una sostituzione ground, E-irriducibile, allora per tutte le regole di inferenza f di S-strategy, se $(E\sigma; s\sigma \simeq t\sigma) \vdash_f (E'; s' \simeq t')$, allora $(E; s \simeq t) \vdash_f (E''; s'' \simeq t'')$, dove E' ed $s' \simeq t'$ sono istanze di E'' ed $s'' \simeq t''$ rispettivamente.*

Lemma 6.4.2 *$(E_i; N_i) \vdash_S (E_{i+1}; N_{i+1})$ se e solo se $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i) \vdash_S (\hat{E}_{i+1}; \hat{s}_{i+1} \simeq \hat{t}_{i+1})$.*

Dimostrazione:

$\Rightarrow$) Un passo di inferenza su $(E_i; N_i)$ è banalmente un passo di inferenza su $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$, dal momento che un passo di inferenza su clausole non ground è banalmente un passo di inferenza su tutte le loro istanze.

$\Leftarrow$) Poiché $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ è minimale, $\hat{E}_i \subseteq E_i\sigma$ ed $\hat{s}_i \simeq \hat{t}_i \in N_i\sigma$ per una sostituzione ground, E-irriducibile σ. Dunque, per il Lemma 6.4.1, un passo di inferenza su $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ è un passo di inferenza su $(E_i; N_i)$. □

Possiamo formulare finalmente il seguente teorema:

Teorema 6.4.1 *La S-strategy è una procedura di completamento sul dominio $\mathcal{T}$ di tutte le equazioni ground.*

Dimostrazione: monotonicità e rilevanza seguono dalla soundness delle regole di inferenza [48]. Per la definizione dell'ordinamento $>_S$, le regole di inferenza della S-strategy sono prova-riducenti se lo sono sulle prove ground rispetto all'ordinamento $>_{UKB}$. Questo segue dal Lemma 6.1.1 e dal Lemma 6.1.2, dal momento che Deduzione sull'obbiettivo si riduce a Semplificazione dell'obbiettivo se l'obiettivo è ground. □

Se il piano di ricerca è *fair*, la S-strategy è una procedura di semidecisione per teorie nella logica dell'uguaglianza e dell'inuguaglianza:

Teorema 6.4.2 (Hsiang and Rusinowitch 1987) [48] *Una formula $\bar{Q}\bar{x}\ s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$ è un teorema di una teoria equazionale E se e solo se la S-strategy deriva* true *da $(E; s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n)$.*

Capitolo 7

Confutazione di congetture induttive

La procedura di completamento di Knuth-Bendix è stata anche applicata a *confutare congetture induttive* in teorie equazionali. Questo metodo è stato chiamato *induzione senza induzione (inductionless induction)*, *prova per consistenza* o *prova per mancanza di inconsistenza* da diversi autori [68, 40, 53, 66, 54, 38, 60, 61, 13, 56]. Estensioni di questo metodo alla logica di Horn con uguaglianza sono state esplorate in [63].

7.1 Procedure di semidecisione per confutare congetture induttive

Una clausola φ è un *teorema induttivo* di S, scritto $S \models_{Ind} \varphi$, se e solo se per tutte le sostituzioni ground σ, $\varphi\sigma \in Th(S)$. Denotiamo con $Ind(S)$ l'insieme di tutti i teoremi induttivi di S, $Ind(S) = \{\varphi |\ S \models_{Ind} \varphi\}$, con $GTh(S)$ l'insieme di tutti i teoremi ground di S, $GTh(S) = \{\varphi |\ \varphi \in Th(S), \varphi\ ground\}$ e con $G(\varphi)$ l'insieme di tutte le istanze ground di φ, $G(\varphi) = \{\varphi\sigma |\ \varphi\sigma\ ground\}$.

Una clausola φ non è un teorema induttivo di S se esiste una sostituzione ground σ tale che $\varphi\sigma \notin GTh(S)$. Se $GTh(S)$ è decidibile, il complemento di $Ind(S)$ è semidecidibile. D'altra parte $Ind(S)$ può non essere semidecidibile anche se $GTh(S)$ è decidibile. Se $Ind(S)$ fosse semidecidibile, sia $Ind(S)$ che il suo complemento sarebbero semidecidibili, ovvero il problema sarebbe decidibile.

Se $\varphi \notin Ind(S)$, allora $GTh(S \cup \{\varphi\}) \neq GTh(S)$, dal momento che $\varphi\sigma \in GTh(S \cup \{\varphi\})$ per tutte le istanze ground $\varphi\sigma$. Quindi, possiamo provare che φ non è un teorema induttivo di S provando l'obbiettivo seguente:

$\Phi_0 = \exists\sigma\ \exists\psi \in S \cup \{\varphi\}$ *tale che* σ *è ground e* $\psi\sigma \in GTh(S \cup \{\varphi\}) - GTh(S)$.

Se esiste un oracolo $\mathcal{O}$ per decidere un obbiettivo siffatto, una procedura di completamento $\mathcal{C} =< I_p, I_t; \Sigma; \mathcal{O} >$ arricchita con l'oracolo $\mathcal{O}$ sarà una *procedura di semidecisione per confutare congetture induttive*. Una derivazione ha la forma

$(S \cup \{\varphi\}; \Phi_0) \vdash_{\mathcal{C}} (S_1; \Phi_1) \vdash_{\mathcal{C}} \ldots (S_i; \Phi_i) \vdash_{\mathcal{C}} \ldots$.

A ogni passo l'obbiettivo è

$\Phi_i = \exists\sigma\ \exists\psi \in S_i$ *tale che* σ *è ground e* $\psi\sigma \in GTh(S_i) - GTh(S)$,

dove σ *ground* indica una sostituzione che associa alle variabili termini ground sulla segnatura di S. Nessun passo di inferenza si applica direttamente all'obbiettivo: la procedura prende in input la presentazione $S \cup \{\varphi\}$ data dalla presentazione originale e dalla congettura induttiva e procede applicando regole di inferenza alla presentazione finché ottiene una presentazione S_k tale che l'oracolo applicato a S_k risponde positivamente e riduce Φ_k a $true$.

Nel caso equazionale, Φ_i è

$\Phi_i = \exists\sigma\ \exists s_i \simeq t_i \in E_i$ *tale che* σ *è ground e* $(s_i \simeq t_i)\sigma \in GTh(E_i) - GTh(E)$.

Un oracolo per decidere Φ_i è disponibile sotto l'assunzione che l'insieme di equazioni in input E sia ground confluente. Sotto questa ipotesi, Φ_i è vero se e solo se ci sono due termini ground E-irriducibili s e t tali che $s_i\sigma \rightarrow_E^* s$, $t_i\sigma \rightarrow_E^* t$ e $s \simeq t \in GTh(E_i)$. In conseguenza, possiamo restringere la nostra attenzione ai termini ground E-irriducibili.

Il primo oracolo è stato dato in [53] per presentazioni equazionali che soddisfano il *principio di definizione*:

Definizione 7.1.1 (Huet and Hullot 1982) [53] *Una presentazione E di una teoria equazionale soddisfa il* principio di definizione *se*

- *la segnatura F di E è data dall'unione disgiunta $F = C \uplus D$, di due insiemi di simboli di funzione, l'insieme C dei* costruttori *e l'insieme D dei* simboli definiti,
- *l'insieme $T(C)$ di tutti i termini ground di costruttori è* libero, *cioè per nessuna coppia di termini t_1 e t_2 in $T(C)$ vale $t_1 \leftrightarrow_E^* t_2$ e*
- *tutti i simboli di funzione in D sono* completamente definiti *su C, ovvero per tutti i termini ground $t \in T(F)$, esiste un unico termine ground di costruttori $t' \in T(C)$ tale che $t \leftrightarrow_E^* t'$.*

Se una presentazione E soddisfa il principio di definizione, i termini ground E-irriducibili sono i termini ground di costruttori. Quindi, Φ_i è vero se e solo se ci sono due termini ground di costruttori t_1 e t_2 tali che $t_1 \leftrightarrow_{E_i}^* t_2$. Le regole di inferenza seguenti implementano questo test [53]:

- *Confutazione 1*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq g(s_1 \dots s_n)\}; \Phi)}{(E \cup \{f(t_1 \dots t_n) \simeq g(s_1 \dots s_n)\}; true)}\ f, g \in C,\ f \neq g$
- *Confutazione 2*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq x\}; \Phi)}{(E \cup \{f(t_1 \dots t_n) \simeq x\}; true)}\ f \in C$
- *Decomposizione*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq f(s_1 \dots s_n)\}; \Phi)}{(E \cup \{t_1 \simeq s_1 \dots t_n \simeq s_n\}; \Phi')}\ f \in C$

dove Φ' è Φ_i per $E_i = E \cup \{t_1 \simeq s_1 \dots t_n \simeq s_n\}$. Le due regole di *Confutazione* segnalano che sono state derivate uguaglianze tra termini costruttori. Per il principio di definizione, la teoria di E_0 non contiene tali uguaglianze. Esse sono una conseguenza dell'aver aggiunto la congettura induttiva $s \simeq t$, che viene quindi confutata. Aggiungiamo la regola di *Decomposizione* per motivi di efficienza. Essa sostituisce un'equazione $f(t_1 \dots t_n) \simeq f(s_1 \dots s_n)$, dove f è un costruttore, con

le equazioni $t_1 \simeq s_1 \ldots t_n \simeq s_n$: siccome f è un costruttore, per il principio di definizione due termini $f(t_1 \ldots t_n)$ ed $f(s_1 \ldots s_n)$ possono essere uguali solo se i loro argomenti sono uguali.

Teorema 7.1.1 (Huet and Hullot 1982) [53], (Bachmair 1988) [13] *Se E è una presentazione equazionale ground confluente, che soddisfa il principio di definizione, la procedura di completamento di Unfailing Knuth-Bendix arricchita con le regole di inferenza* Decomposizione, Confutazione 1 *e* Confutazione 2 *è una procedura di semidecisione per il complemento di $Ind(E)$.*

Questo risultato è stato ottenuto assumendo un piano di ricerca uniformemente fair sul dominio di tutte le equazioni ground.

Un oracolo più generale è stato proposto in [54] per la procedura di completamento di Knuth-Bendix ed esteso alla procedura UKB in [13]. Questo test è basato sulla *riducibilità ground*: un termine t è *ground E-riducibile* se per tutte le sostituzioni ground σ, $t\sigma$ è E-riducibile. La ground E-riducibilità è decidibile solo se E è un sistema di riscrittura ground confluente [72]. Perciò, il test in [54, 13] è applicabile solo se la presentazione in input E è ground confluente e tutte le sue equazioni possono essere orientate in regole di riscrittura.

Assumiamo che E abbia queste proprietà e lo ribattezziamo R. Un'equazione $l \simeq r$ è *ground R-riducibile* se per tutte le sostituzioni ground σ, tali che $l\sigma$ ed $r\sigma$ sono distinti, o $l\sigma$ o $r\sigma$ è R-riducibile. Se a un certo stadio i viene derivata da $R \cup \{s \simeq t\}$ un'equazione $l \simeq r$ che non è ground R-riducibile, allora c'è un'istanza ground $l\sigma \simeq r\sigma$ dell'equazione tale che $l\sigma$ ed $r\sigma$ sono distinti ed R-irriducibili, ma $l\sigma \simeq r\sigma \in GTh(E_i)$. Questo significa che Φ_i è vero e la congettura induttiva è confutata. La seguente regola di inferenza implementa questo test:

Confutazione 3:

$$\frac{(E \cup \{l \simeq r\}; \Phi)}{(E \cup \{l \simeq r\}; true)}\ l \simeq r \text{ non è ground } R - \text{riducibile}$$

Teorema 7.1.2 (Jouannaud and Kounalis 1986) [54], (Bachmair 1988) [13] *Se R è un sistema di riscrittura ground confluente, la procedura di completamento Unfailing Knuth-Bendix arricchita con la regola di inferenza* Confutazione 3 *è una procedura di semidecisione per il complemento di $Ind(R)$.*

Come nel risultato precedente, si assume un piano di ricerca uniformemente fair sul dominio di tutte le equazioni ground.

Il test di ground riducibilità non è una soluzione pratica al problema della dimostrazione di teoremi induttivi, perchè la sua complessità è molto alta. Inoltre, i due risultati suddetti sulla procedura UKB per confutare teoremi induttivi sono stati ottenuti in un contesto in cui il completamento era considerato come generazione di sistemi confluenti e la capacità di confutare congetture induttive era soltanto un effetto collaterale. Questo spiega come mai entrambi i risultati sono stati ottenuti sotto l'ipotesi che il piano di ricerca sia uniformlemente fair.

D'altra parte, abbiamo mostrato che confutare una congettura induttiva è un processo di semidecisione con uno specifico obbiettivo. Quindi, occorre ridurre soltanto la prova dell'obbiettivo. Definiamo la prova dell'obbiettivo Φ_i nel modo seguente:

$\Pi(S_i, \Phi_i) = \Pi(S_i, min\{\psi\sigma |\ \psi \in S_i, \psi\sigma \in GTh(S_i) - GTh(S)\})$,

ovvero la prova dell'obbiettivo è la prova della minima istanza ground di qualche clausola in S_i che è un teorema di S_i ma non di S.

In logica equazionale, una procedura di completamento che genera prima o poi un insieme ground confluente di equazioni, è in grado di ridurre le prove di tutti i teoremi ground e quindi la prova dell'obbiettivo. Tuttavia, questo non è necessario. Dal momento che la prova dell'obbiettivo è la prova del minimo teorema ground che non è un teorema della presentazione originale, possiamo restringere la nostra attenzione a un insieme più piccolo di teoremi ground:

Definizione 7.1.2 (Fribourg 1986) [38] *Data una presentazione ground confluente E, un insieme di sostituzioni H è* E-induttivamente completo *se per tutte le sostituzioni ground ρ, esistono una sostituzione $\sigma \in H$ e una sostituzione ground τ tali che $\rho \rightarrow_E^* \sigma\tau$.*

Per esempio, se E include gli assiomi $0{+}x \simeq x$ e $succ(x){+}y \simeq succ(x{+}y)$, un insieme di sostituzioni H è E-induttivamente completo se contiene due sostituzioni σ_1 e σ_2 tali che $\{x \mapsto 0\} \in \sigma_1$ e $\{x \mapsto succ(y)\} \in \sigma_2$. Infatti, tutti i termini ground si riducono o a 0 o a un termine $succ^n(0)$, così che un insieme di sostituzioni che copre le istanze $x \mapsto 0$ e $x \mapsto succ(y)$ copre tutte le istanze.

Chiaramente, siamo interessati a insiemi minimali E-induttivamente completi. Tutti gli insiemi siffatti sono equivalenti, dal momento che hanno tutti la proprietà di coprire tutte le sostituzioni ground. Denotiamo con H_E un tale insieme minimale e con $\mathcal{IT}_E$ il dominio delle equazioni ground istanze di sostituzioni in H_E, cioè $\mathcal{IT}_E = \{(l \simeq r)\sigma\tau \mid \sigma \in H_E, (l \simeq r)\sigma\tau \text{ è } ground\}$. La prova dell'obbiettivo è la prova del minimo teorema ground che non è un teorema della presentazione originale. Questo minimo teorema è nel dominio $\mathcal{IT}_E$ e quindi ridurre le prove dei teoremi in $\mathcal{IT}_E$ è sufficiente a garantire che la prova dell'obbiettivo venga ridotta, come venne provato originariamente in [38] per l'applicazione del completamento di Knuth-Bendix alla confutazione di congetture induttive in teorie equazionali:

Teorema 7.1.3 (Fribourg 1986) [38] *Una procedura di completamento $\mathcal{C} =< I_p, I_t; \Sigma; \mathcal{O} >$ sul dominio $\mathcal{IT}_E$, con regole di inferenza complete e piano di ricerca fair è una procedura di semidecisione per il complemento di $Ind(E)$ per tutte le presentazioni equazionali E, per cui l'oracolo $\mathcal{O}$ è computabile.*

In conseguenza, la regola di Deduzione di UKB può essere ristretta in modo tale che allo stadio i della derivazione, le sovrapposizioni su $p \simeq q$ alla posizione u vengano eseguite solo se l'insieme di mgus $\{\sigma | l\sigma = (p|u)\sigma, l \simeq r \in E_i\}$ è E-induttivamente completo. La posizione u è detta *completamente sovrapponibile* in [38]:

Deduzione su posizioni completamente sovrapponibili:

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \Phi)}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \Phi')} \quad \begin{array}{l} p|u \notin X,\ (p|u)\sigma = l\sigma \\ u \text{ è completamente sovrapponibile} \end{array} \qquad p\sigma \not\preceq q\sigma, p[r]_u\sigma$$

dove Φ' è Φ_i per $E_i = E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}$. Inoltre, per tutte le equazioni $p \simeq q$, generate durante la derivazione, è sufficiente eseguire le sovrapposizioni su soltanto una posizione

completamente sovrapponibile in $p \simeq q$. In altre parole, un piano di ricerca che seleziona una posizione completamente sovrapponibile in ogni equazione è fair. Possiamo riassumere questi risultati nel teorema seguente:

Teorema 7.1.4 (Fribourg 1986) [38] *Il Teorema 7.1.1 e il Teorema 7.1.2 valgono anche se la regola di Deduzione è ristretta a posizioni completamente sovrapponibili e il piano di ricerca seleziona una posizione completamente sovrapponibile in ogni equazione generata.*

Questo risultato richiede un algoritmo che riconosca le posizioni completamente sovrapponibili. Una caratterizzazione equivalente è la seguente: una posizione u in p è completamente sovrapponibile se per tutte le istanze ground $(p|u)\rho$ esiste un'equazione $l \simeq r$ in E tale che $(p|u)\rho = l\sigma$ e $l\sigma \succ r\sigma$. Il problema di riconoscere le posizioni completamente sovrapponibili si riduce al problema della riducibilità ground. Tuttavia, se la presentazione soddisfa il principio di definizione, una posizione u è completamente sovrapponibile se $p|u$ è un termine che ha un simbolo definito alla radice e solo simboli costruttori e variabili alle posizioni sotto la radice. Quindi, il teorema suddetto ha applicazione pratica nel caso di presentazioni che soddisfano il principio di definizione.

Capitolo 8

Programmazione logica

In questo capitolo presentiamo una procedura di completamento per la programmazione logica, chiamata *Linear Completion*. I programmi logici interpretati da Linear Completion sono sistemi di riscrittura o *programmi a riscrittura*. I programmi a riscrittura differiscono dai programmi Prolog perché permettono di distinguere tra i predicati che sono *definiti in modo mutualmente esclusivo* e quelli che non lo sono. Un predicato è definito in modo mutualmente esclusivo se la testa di ciascuna delle sue clausole è logicamente equivalente al suo corpo. Un esempio tipico di tale definizione è la usuale definizione di *append*. Nei programmi a riscrittura, i predicati definiti in modo mutualmente esclusivo sono definiti da equivalenze logiche, mentre i predicati che non sono definiti in modo mutualmente esclusivo sono definiti da implicazioni come in Prolog. Mostriamo attraverso alcuni esempi come i programmi a riscrittura possono risultare più accurati dei programmi Prolog nel catturare la semantica intesa dal programmatore, proprio grazie a questa caratteristica addizionale.

La nostra procedura di Linear Completion differisce dalla precedente [29, 30], perché permettiamo la *semplificazione* dei goals da parte dei loro antenati. L'interprete per il Prolog puro non ha regole di contrazione, poiché la risoluzione è una regola di espansione. Siamo quindi interessati a studiare gli effetti di una regola di contrazione nell'interpretazione dei programmi logici.

La semplificazione riduce lo spazio di ricerca e la quantità di backtracking eseguita dall'interprete. Essa previene certe esecuzioni infinite che sono altrimenti inevitabili nel Prolog puro. Se si consente la semplificazione, i programmi di equivalenze mostrano un comportamento diverso da quello dei programmi di implicazioni. Più precisamente, i programmi a riscrittura dove i predicati sono definiti da equivalenze possono dare meno risposte dei programmi Prolog dove gli stessi predicati sono definiti da implicazioni, perché i processi di generazione di alcune risposte vengono interrotti dalla semplificazione. Questo accade nonostante i programmi a riscrittura siano *denotazionalmente equivalenti* ai programmi Prolog al livello ground.

Diamo una caratterizzazione a punto fisso dei programmi a riscrittura e mostriamo che la semantica operazionale data da Linear Completion, la semantica logica e la semantica a punto fisso sono tutte equivalenti. I programmi a riscrittura e i programmi Prolog per gli stessi predicati hanno lo stesso punto fisso, cioè lo stesso insieme di fatti ground veri. Può accadere che i programmi a riscrittura diano meno risposte perché se i predicati sono definiti da equivalenze,

distinte risposte Prolog risultano equivalenti rispetto al programma a riscrittura e "collassano" su una sola risposta. Tuttavia, è garantito che i programmi a riscrittura non perdano nessuna risposta necessaria: per ogni risposta Prolog c'è una risposta equivalente data dal programma a riscrittura. Questo spiega come mai un insieme di risposte più piccolo copre lo stesso insieme di fatti ground veri.

Concludiamo con un confronto tra gli effetti della simplificazione e quelli di alcuni meccansimi di controllo dei cicli in Prolog, basati sulla sussunzione, che sono stati presentati in [7, 17].

8.1 Programmi a riscrittura

I sistemi a riscrittura sono stati ampiamente applicati in programmazione funzionale [23, 39, 41, 43] e in un grado minore in programmazione logica [29, 30, 31, 73]. Nel caso dei programmi funzionali il meccanismo di valutazione è la riduzione e una computazione consiste nel ridurre un termine ground in input a un termine irriducibile, che rappresenta l'output. Per ottenere la programmazione logica si estende il meccanismo di valutazione a includere sia la riduzione che una forma lineare di sovrapposizione. Questo corrisponde a una versione molto ristretta del completamento di Knuth-Bendix, detta *Linear Completion* [30]. Una computazione consiste nel generare una sostituzione risposta per un quesito non ground come in Prolog.

Nonostante vari approcci suggeriti finora, è diffuso il falso concetto che i programmi a riscrittura abbiano la stessa semantica dei programmi Prolog, pur con un diverso meccanismo di valutazione. Abbastanza sorprendentemente, questo non è vero in generale. Presentiamo una semantica più precisa, sia operazionale che denotazionale, per i programmi a riscrittura e mostriamo come essi possono evitare certe esecuzioni infinite che si verificano in simili programmi Prolog.

La ragione principale del diverso comportamento dei programmi a riscrittura è l'utilizzazione di una regole di inferenza per la semplificazione. Mostriamo il suo uso con un semplice esempio. Il seguente programma Prolog:

$append([\,], L, L).$

$append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).$

con il quesito

$?- append(X, [b|Y], [a, b, c|Z]).$

genera un insieme infinito di soluzioni come si vede in Fig. 8.1.

Il programma a riscrittura, d'altro canto, è definito da

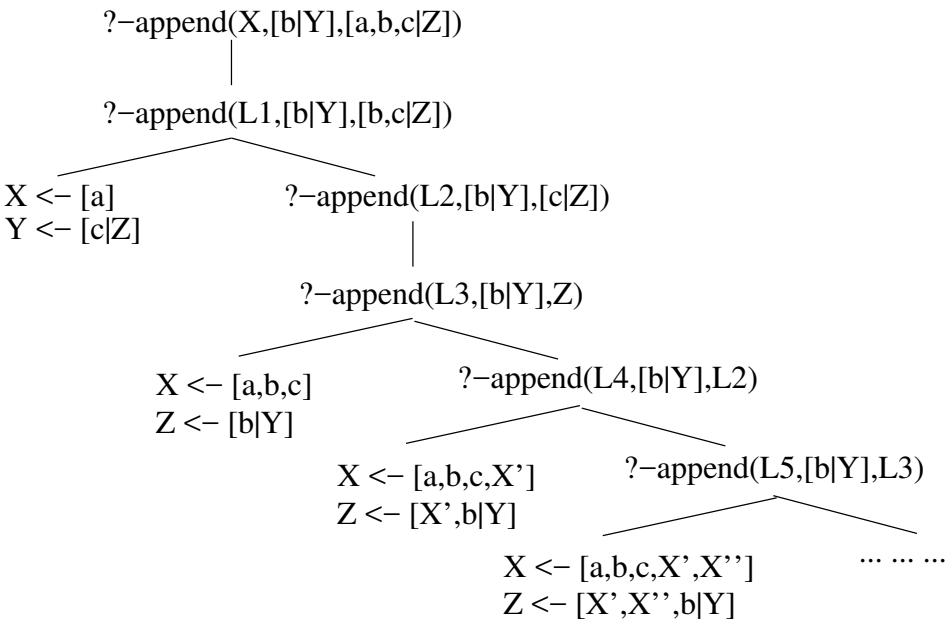
$append([\,], L, L) \rightarrow true$

$append([X|L1], Y, [X|L2]) \rightarrow append(L1, Y, L2)$

e il quesito è

$append(X, [b|Y], [a, b, c|Z]) \rightarrow answer(X, Y, Z).$

Se eseguiamo questo programma mediante Linear Completion otteniamo soltanto le prime due



$\{X \mapsto [a],\ Y \mapsto [c|Z]\}$
$\{X \mapsto [a,b,c],\ Z \mapsto [b|Y]\}$
$\{X \mapsto [a,b,c,X'],\ Z \mapsto [X',b|Y]\}$
$\{X \mapsto [a,b,c,X',X''],\ Z \mapsto [X',X'',b|Y]\}$
. . . . . .
$\{X \mapsto [a,b,c,X',X'',\ldots,X^n],\ Z \mapsto [X',X'',\ldots,X^n,b|Y]\}$
. . . . . .

Figura 8.1: Prolog genera un insieme infinito di soluzioni

risposte come mostrato in Fig. 8.2.

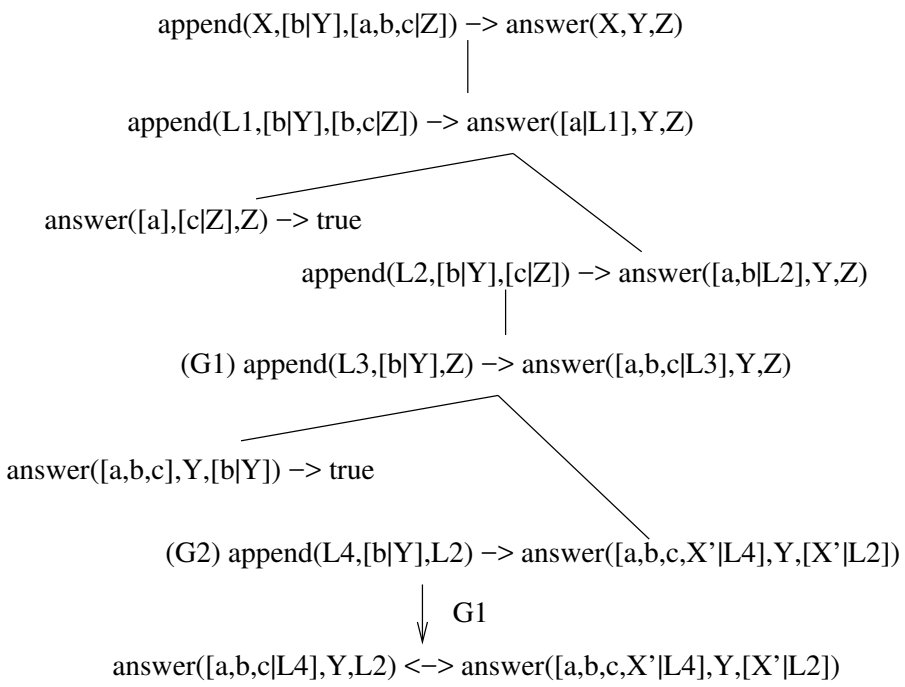
L'ultimo passo, etichettato con ↓, è un passo di semplificazione dove un goal antenato, etichettato con (G1) in figura, riscrive il goal corrente (G2) a un goal banale di forma $answer(\ ) \leftrightarrow answer(\ )$. Poiché nessuna inferenza può essere applicata a questo goal, l'esecuzione si arresta con soltanto due risposte.

La causa di questo diverso comportamento è che il programma Prolog e il programma a riscrittura danno due diverse definizioni di *append*. Le "unità elementari" in un programma a riscrittura e in un programma Prolog sono interpretate in modo diverso. In Prolog, ogni unità è una clausola, dove ": −" indica un *se* logico. Nei programmi a riscrittura, il connettivo "→" significa *se e solo se*. Le infinite risposte di forma

$$\{X \mapsto [a,b,c,X',X'',\ldots,X^n],\ Z \mapsto [X',X'',\ldots,X^n,b|Y]\}$$

date da Prolog non sono equivalenti se *append* è definito da implicazioni, ma collassano tutte sulla seconda soluzione

$$\{X \mapsto [a,b,c],\ Z \mapsto [b|Y]\}$$



$\{X \mapsto [a],\ Y \mapsto [c|Z]\}$
$\{X \mapsto [a,b,c],\ Z \mapsto [b|Y]\}$

Figura 8.2: Linear completion genera soltanto due risposte

se *append* è definito da bi-implicazioni. La prima risposta del programma a riscrittura corrisponde alla prima risposta del programma Prolog. La seconda risposta del programma a riscrittura corrisponde alla seconda risposta del programma Prolog e a tutte le seguenti.

Possiamo vedere perché questo accade instanziando il quesito con le sostituzioni risposta. Se il quesito viene instanziato con la prima sostituzione risposta, viene ridotto ad $append([\,], [b, c|Z], [b, c|Z])$ e poi a $true$. Se viene instanziato con la seconda risposta o una delle successive, viene ridotto ad $append([\,], [b|Y], [b|Y])$ e poi a $true$. Tutte le risposte eccetto la prima si riducono allo stesso fatto vero se si applica la semplificazione, cioè se *append* è definito da equivalenze. Tutte quelle risposte sono equivalenti alla seconda rispetto al programma a riscrittura e così non vengono generate.

Dal momento che la definizione intesa di *append* è proprio

$append([X|L1], Y, [X|L2])$ *se e solo se* $append(L1, Y, L2)$,

il programma a riscrittura sembra essere più vicino del programma Prolog al significato inteso del programma.

L'interpretazione delle unità del programma come equivalenze logiche può anche aiutare nel risolvere certi cicli che possono verificarsi in Prolog. Per esempio, si consideri il seguente programma Prolog:

$\ldots$

$P(X, Y, Z) :- append(X, [b|Y], [a, b, c|Z]), non-member(a, X).$

$\ldots$

$append([\,], L, L).$

$append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).$

$\ldots$

con il quesito $?- P(X, Y, Z)$.

Prolog cade in un ciclo infinito quando valuta la prima clausola per P, poiché a è un elemento di X in tutte le infinite soluzioni di $append(X, [b|Y], [a, b, c|Z])$. Questi cicli potenziali non possono essere evitati nemmeno usando il *cut*. Il programma a riscrittura non va in ciclo e valuta la seconda clausola di P, giacché ci sono soltanto due risposte dalla valutazione del sottogoal *append* e nessuna di esse soddisfa il sottogoal $non-member$.

Si potrebbe sospettare che la semplificazione elimini troppe risposte e cambi la semantica intesa. Per esempio, consideriamo il programma:

$Q(X, Y, Z) :- append(X, [b|Y], [a, b, c|Z]), size(X) > 3.$

con il quesito $?- Q(X, Y, Z)$.

Siccome nelle due risposte al sottogoal *append* generate negli esempi precedenti il numero degli elementi di X è minore o uguale a tre, ci si potrebbe aspettare che non venga generata nessuna soluzione. Ma non è così. Quando si dà il quesito $Q(X, Y, Z) \rightarrow answer(X, Y, Z)$, l'esecuzione genera prima le due soluzioni del sottogoal *append*

$size([a]) > 3 \rightarrow answer([a], [c|Z], Z)$

$size([a, b, c]) > 3 \rightarrow answer([a, b, c], Y, [b|Y]),$

nessuna delle quali dà una soluzione al problema Q. Poi l'esecuzione continua con il goal

$append(L1, [b|Y], L2), size([a, b, c, X'|L1]) > 3 \rightarrow answer([a, b, c, X'|L1], Y, [X'|L2]).$

Assumendo che *size* sia definito come è desiderabile, $size([a, b, c, X'|L1]) > 3$ si semplifica in *true* e il goal è ridotto a

$append(L1, [b|Y], L2) \rightarrow answer([a, b, c, X'|L1], Y, [X'|L2])$ (G2).

Si noti che questo è il medesimo goal (G2) generato nella esecuzione precedente per il quesito *append*. In quella esecuzione questo goal viene riscritto dal suo antenato (G1) e non dà nessuna risposta. Nella esecuzione per il quesito Q tutti gli antenati contegono anche un letterale *size*, così che non si applicano a semplificare il goal (G2), che offre una corretta soluzione

$answer([a, b, c, X'], Y, [X', b|Y]) \rightarrow true.$

Infine la computazione si arresta quando il nuovo goal

$$append(L1, [b|Y], L3) \rightarrow answer([a, b, c, X', X''|L1], Y, [X', X''|L3])$$

viene ridotto dal suo antenato (G2) a

$$answer([a, b, c, X'|L1], Y, [X'|L3]) \leftrightarrow answer([a, b, c, X', X''|L1], Y, [X', X''|L3])\ ^{1}.$$

Finora abbiamo visto esempi dove è desiderabile definire i predicati con equivalenze. Certamente, questo non è vero per tutte le relazioni. Tuttavia, i programmi a riscrittura ci permettono di definire predicati mediante implicazioni. Per esempio, per la relazione *ancestor*

$$ancestor(X, Y) :- parent(X, Y).$$
$$ancestor(X, Y) :- parent(Z, Y),\ ancestor(X, Z).$$

entrambe le clausole sono implicazioni. Come già osservato in [30], si possono scrivere le implicazioni come bi-implicazioni ricordando che $P :- Q$ è equivalente a $P \wedge Q \rightarrow Q$. Se si aggiungono alcuni fatti e un quesito, abbiamo:

$$parent(jb, lc) \rightarrow true$$
$$parent(jb, gg) \rightarrow true$$
$$parent(gg, wm) \rightarrow true$$
$$ancestor(X, Y),\ parent(X, Y) \rightarrow parent(X, Y)$$
$$ancestor(X, Y),\ parent(Z, Y),\ ancestor(X, Z) \rightarrow parent(Z, Y),\ ancestor(X, Z)$$
$$ancestor(jb, Z) \rightarrow answer(Z).$$

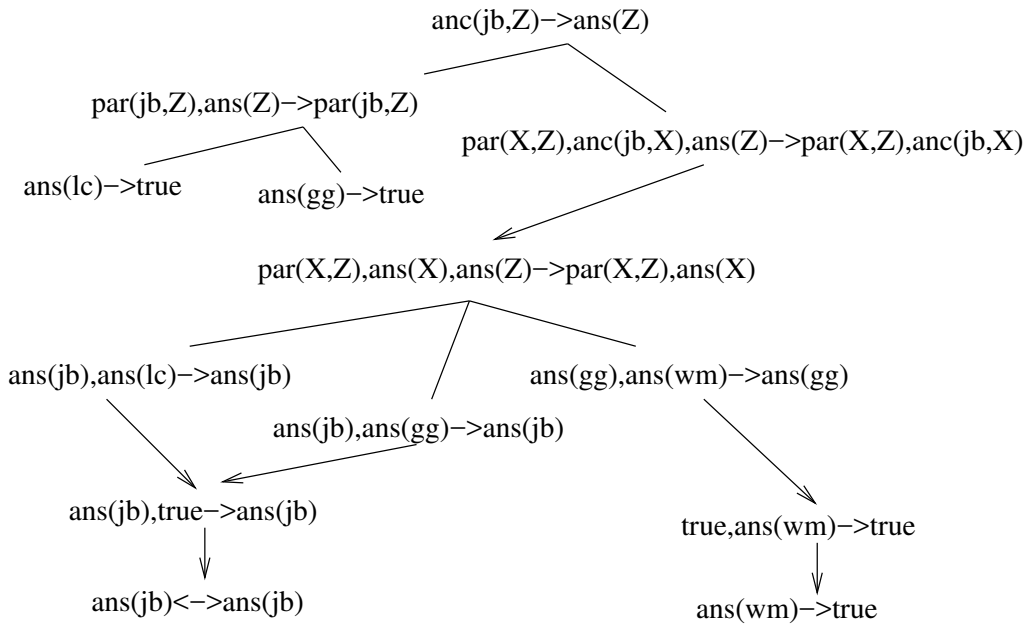
Questo programma dà le stesse risposte di Prolog, ma la computazione è ottimizzata dalla semplificazione dei goals come illustrato in Fig. 8.3.

La generazione delle prime due risposte avviene come in Prolog. Per la terza risposta è diverso. Avendo il goal $parent(X, Z), ancestor(jb, X)$, Prolog genera prima $ancestor(jb, jb)$ due volte, fallisce due volte, poi genera il goal $ancestor(jb, gg)$, che dà la risposta $z \mapsto wm$, e una terza computazione fallimentare per il goal $ancestor(jb, jb)$. Questi cammini fallimentari vengono eliminati dalla riscrittura. La semplificazione mediante risposte generate in precedenza riduce il numero di applicazioni ricorsive della definizione di *ancestor* e la quantità di backtracking eseguita dall'interprete.

8.2 Linear Completion: regole di inferenza e piano di ricerca

Abbiamo visto nella sezione precedente, che i programmi a riscrittura ci permettono di definire predicati o mediante equivalenze o mediante implicazioni. Un predicato è *definito in modo mu-*

[1] Vedremo nel seguito che le regole di inferenza di Linear Completion includono la sovrapposizione tra il goal corrente e le risposte generate in precedenza. In questo caso il goal corrente viene orientato da destra a sinistra e nessuna delle risposte già generate si sovrappone sul lato maggiore. Quindi nessun altro passo è possibile.



$\{Z \mapsto lc\}$
$\{Z \mapsto gg\}$
$\{Z \mapsto wm\}$

Figura 8.3: Ottimizzazione della computazione via semplificazione

tualmente esclusivo se è definito da un insieme di clausole dove non ci sono due teste unificabili[2]. Se un predicato è definito in modo mutualmente esclusivo, le sue regole di riscrittura sono equivalenze, altrimenti sono implicazioni. Più precisamente, se un predicato A è definito da un insieme di clausole

$A(\bar{t}_1).$

$\ldots$

$A(\bar{t}_m).$

$A(\bar{t}_{m+1}) :- B_{11} \ldots B_{1p_1}.$

$\ldots$

$A(\bar{t}_{m+n}) :- B_{n1} \ldots B_{np_n}.$

il suo programma a riscrittura contiene le regole

$A(\bar{t}_1) \to true.$

$\ldots$

[2]In base a questa definizione, le clausole $p(x) :- x > 0 \ldots$ e $p(x) :- x \leq 0 \ldots$ non sono mutualmente esclusive e chiaramente questo non è completamente soddisfacente. Tuttavia, questa definizione è sufficiente per gli scopi di questa discussione. Si può trovare una nozione più generale di clausole mutualmente esclusive in [25].

$A(\bar{t}_m) \to true.$

$A(\bar{t}_{m+1}) \to B_{11} \dots B_{1p_1}.$

$\dots$

$A(\bar{t}_{m+n}) \to B_{n1} \dots B_{np_n}.$

se A è definito in modo mutualmente esclusivo e $\forall i, 1 \leq i \leq n, A(\bar{t}_{m+i}) \succ B_{i1} \dots B_{ip_i}$, dove $\succ$ è un ordinamento di semplificazione completo. Altrimenti A viene trasformato in

$A(\bar{t}_1) \to true.$

$\dots$

$A(\bar{t}_m) \to true.$

$A(\bar{t}_{m+1})B_{11} \dots B_{1p_1} \to B_{11} \dots B_{1p_1}.$

$\dots$

$A(\bar{t}_{m+n})B_{n1} \dots B_{np_n} \to B_{n1} \dots B_{np_n}.$

Assumiamo un ordinamento di semplificazione completo $\succ$ su termini e letterali tale che $P \succ answer(\bar{t}) \succ true$ per tutti gli atomi P il cui simbolo di predicato è diverso da *answer* e da *true* e per tutti i termini $\bar{t}$. Chiamiamo le regole di riscrittura che rappresentano fatti, implicazioni e bi-implicazioni *regole fatti*, *regole se* e *regole sse*. Un *programmma a riscrittura* è un sistema di riscrittura di regole se, regole sse e regole fatti. Se un programma ha solo regole se e regole fatti, lo chiamiamo un *programma se*. Si noti che ogni programma Prolog può essere trasformato in un programma se. Altrimenti lo chiamiamo un *programma sse*.

I programmi a riscrittura sono interpretati da *Linear Completion*. I dati in input per Linear Completion sono un programmma a riscrittura, la presentazione, e un quesito, l'obbiettivo o goal.

Un quesito $\exists \bar{x} Q_1 \dots Q_m$ viene negato in $Q_1 \dots Q_m \to false$ e scritto come una *regola quesito* $Q_1 \dots Q_m \to answer(\bar{x})$, dove $\bar{x}$ contiene tutte le variabili libere in $Q_1 \dots Q_m$. Quando una regola di forma $answer(\bar{x})\sigma \to true$ viene dedotta, $\bar{x}\sigma$ è una soluzione al quesito. Il letterale *answer* venne introdotto originariamente in [42] e usato in [30].

Una regola risposta significa una contraddizione in senso refutazionale, poiché *answer* significa $false$ logicamente. L'interprete costruisce una refutazione a partire dal quesito:

$E \cup \{Q_1 \dots Q_m \to answer(\bar{x})\} \vdash_{LC} answer(\bar{x})\sigma \to true$

Usiamo anche la notazione $E \vdash_{LC} Q_1 \dots Q_m\sigma$, a significare che $Q_1 \dots Q_m\sigma$ è stato provato dal programma E mediante Linear Completion. Se si dà un quesito ground, la sostituzione risposta è vuota e scriviamo $E \vdash_{LC} Q_1 \dots Q_m$.

Una computazione di Linear Completion ha la forma

$(E; Q_1 \dots Q_m \to answer(\bar{x}); \emptyset) \vdash_{LC} \dots \vdash_{LC} (E_i; \bar{L}_i \to \bar{R}_i; S_i) \vdash_{LC} \dots$

dove E è il programma, $\bar{L}_i \to \bar{R}_i$ è il goal corrente e S_i contiene tutti gli *antenati* del goal corrente. Usiamo $\bar{L}$, $\bar{R}$ per indicare congiunzioni di atomi. Un passo di computazione applica una delle regole di inferenza seguenti:

Semplificazione:

$$\frac{(E; L_1 \dots L_l \to R_1 \dots R_r; S)}{(E; L'_1 \dots L'_n \to R'_1 \dots R'_s; S)}$$

Deduzione:

$$\frac{(E; L_1 \dots L_l \to R_1 \dots R_r; S)}{(E; L'_1 \dots L'_n \to R'_1 \dots R'_s; S \cup \{L_1 \dots L_l \to R_1 \dots R_r\})}$$

Risposta:

$$\frac{(E; answer(\bar{x})\sigma \to true; S)}{(E \cup \{answer(\bar{x})\sigma \to true\}; —; S)}$$

Eliminazione:

$$\frac{(E; L_1 \dots L_l \leftrightarrow L_1 \dots L_l; S)}{(E; —; S)}$$

Nella *Semplificazione*, $L_1 \dots L_l \to R_1 \dots R_r$ viene semplificato nel nuovo goal $L'_1 \dots L'_n \to R'_1 \dots R'_s$ usando $E \cup S$ e $L_1 \dots L_l \to R_1 \dots R_r$ viene scartato. Semplificazione dei goals mediante goals è la fondamentale differenza tra la nostra definizione di Linear Completion e quella in [30]. La *Deduzione* genera un nuovo goal sovrapponendo una regola in E al goal corrente. Il nuovo goal rimpiazza quello corrente, che viene trasferito nell'insieme degli antenati. Nella regola *Risposta* si deriva una risposta e la regola risposta viene aggiunta al programma. In *Eliminazione* si elimina un goal che è un'identità.

Non si esegue nessuna sovrapposizione tra due regole del programma, nessuna sovrapposizione tra due regole goal e nessuna semplificazione delle regole del programma: tutti i passi di inferenza sono passi di inferenza sull'obbiettivo e quindi tutte le derivazioni sono derivazioni lineari input. Il nome Linear Completion mette in evidenza questa proprietà.

Descriviamo una computazione di Linear Completion come un processo di attraversamento di un I-albero con radice in $(E; Q_1 \dots Q_m \to answer(\bar{x}); \emptyset)$, dove I è l'insieme di regole di inferenza sopraddetto.

In dimostrazione di teoremi si applica una procedura di completamento a un problema di *validità*, come $E_0 \models \forall \bar{x} s_0 \simeq t_0$, dato in input come $(E_0; \hat{s}_0 \simeq \hat{t}_0)$. Perciò, la computazione termina con successo non appena una coppia $(E_k; true)$ viene raggiunta. Una computazione di successo è un cammino nell'I-albero dalla radice $(E_0; \hat{s}_0 \simeq \hat{t}_0)$ a un nodo $(E_k; true)$.

In programmazione logica, un quesito $\exists \bar{x} Q_1 \dots Q_m$ è un problema di *soddisfacibiltà*. Se siamo interessati a una sola risposta, una computazione di successo è un cammino nell'I-albero dalla radice $(E; \bar{Q} \to answer(\bar{x}); \emptyset)$ a un nodo $(E; answer(\bar{x})\sigma \to true; S_k)$. Tuttavia, in pratica un programmatore è interessato ad estrarre tutte le risposte dai dati disponibili. Quindi, la computazione si arresta con successo quando tutte le risposte sono state scoperte, cioè quando tutti i nodi $(E_k; answer(\bar{x})\sigma \to true; S_k)$ nell'I-albero sono stati raggiunti. Una computazione di successo è un sottoalbero finito nell'I-albero, tale che le foglie del sottoalbero sono tutte e sole le risposte al problema. Il piano di ricerca determina come un sottoalbero siffatto viene generato. Se la computazione raggiunge uno stadio dove nessuna regole di inferenza è applicabile, torna

indietro (backtracks). Questo è il caso per esempio quando l'obbiettivo è vuoto, dopo un passo di *Risposta* o di *Eliminazione*.

Anche un piano di ricerca per dimostrazione di teoremi può includere backtracking. Può seguire un cammino nell'I-albero, tornare indietro per uno o più passi e scegliere un altro ramo. Ma il risultato finale di una computazione di successo è un singolo cammino. Il backtracking può essere usato per determinare tale cammino. Nel caso di un piano di ricerca per programmazione logica occorre il backtracking perché il risultato finale di una computazione di successo è un sottoalbero con tutte le risposte.

Un piano di ricerca per Linear Completion tenta le regole di inferenza nell'ordine seguente: *Eliminazione, Risposta, Semplificazione* e *Deduzione*. Quindi, il goal corrente viene sempre semplificato il più possibile prima di eseguire un nuovo passo di Deduzione. Questa scelta assicura che l'interprete esegua un passo di sovrapposizione ed estenda lo spazio di ricerca solo se lo spazio di ricerca corrente è stato ridotto il più possibile.

Il piano di ricerca esegue backtracking soltanto sul goal e sull'insieme degli antenati. Le modifiche al programma sono conservate. Poiché la sola modifica al programma è l'aggiunta di regole di risposta, segue che l'interprete non dimentica le risposte già computate e le applica come regole mentre visita altri cammini dell'albero. L'esempio *ancestor* nella Sezione 8.1 mostra che tenere le risposte è necessario per generare tutte le sostituzioni risposta. Inoltre, la semplificazione mediante regole di risposta contribuisce a ridurre l'albero di ricerca e quindi a ottimizzare l'esecuzione.

I passi di sovrapposizione in LC sono simili ai passi di risoluzione in Prolog. Dato un goal $A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r$ e una regola di programma, si può usare uno dei tre tipi di sovrapposizione in dipendenza dal tipo della regola di programma:

Sovrapposizione con una regola se:

$$\frac{A(\bar{s})B_1 \dots B_n \to B_1 \dots B_n,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(B_1 \dots B_n L_1 \dots L_l)\sigma \leftrightarrow (B_1 \dots B_n R_1 \dots R_r)\sigma}$$

Sovrapposizione con una regola sse:

$$\frac{A(\bar{s}) \to B_1 \dots B_n,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(B_1 \dots B_n L_1 \dots L_l)\sigma \leftrightarrow (R_1 \dots R_r)\sigma}$$

Sovrapposizione con una regola fatto:

$$\frac{A(\bar{s}) \to true,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(L_1 \dots L_l)\sigma \leftrightarrow (R_1 \dots R_r)\sigma}$$

dove σ è l'mgu di $A(\bar{s})$ e di $A(\bar{u})$ e il goal generato viene orientato secondo l'ordinamento di semplificazione. Qui e nel seguito assumiamo che si selezioni il letterale più a sinistra in un goal[3]. Un passo di sovrapposziione rimpiazza un letterale nel goal con l'appropriata istanza della definizione del suo predicato. Se la formula che dà la definizione è un'implicazione, si aggiunge il nuovo insieme di sottogoals a entrambi i lati dell'equazione goal. Se è una bi-implicazione, lo si aggiunge al lato sinistro solamente. Una sovrapposizione con un fatto elimina un letterale nel

[3]E' ben noto che computazioni che differiscono nell'ordine di selezione dei letterali danno le stesse risposte a meno di ridenominazioni di variabili [24, 67].

goal.

Non occorre considerare nessuna sovrapposizione su un atomo diverso dalla testa di una regola. Se l'atomo $A(\bar{u})$ nel goal $A(\bar{u})\bar{L} \to \bar{R}$ unifica con mgu σ con un atomo $A(\bar{v})$ in una regola se $CA(\bar{v})\bar{B} \to A(\bar{v})\bar{B}$ il passo di sovrapposizione genera il goal

$$(C\bar{B}\bar{R})\sigma \to (A(\bar{v})\bar{B}\bar{L})\sigma$$

che viene ridotto dal suo predecessore $A(\bar{u})\bar{L} \to \bar{R}$ a

$$(C\bar{B}\bar{R})\sigma \to (\bar{B}\bar{R})\sigma.$$

Una successiva sovrapposizione sul letterale C tra questo nuovo goal e la stessa regola di programma $CA(\bar{v})\bar{B} \to A(\bar{v})\bar{B}$ darà l'identità $(A(\bar{v})\bar{B}\bar{R})\sigma \leftrightarrow (A(\bar{v})\bar{B}\bar{R})\sigma$.

La caratteristica chiave dei programmi a riscrittura è la *semplificazione* del goal corrente da parte delle regole di programma, dei goals antenati e delle regole risposta. Se una regola se $AB_1 \dots B_n \to B_1 \dots B_n$ o un fatto $A \to true$ semplifica un quesito $Q_1 \dots Q_m$, l'atomo Q_j tale che $A\sigma = Q_j$ è eliminato. Se si applica una regola sse $A \to B_1 \dots B_n$, l'atomo Q_j viene sostituito da $(B_1 \dots B_n)\sigma$. Si applicano implicitamente anche le regole di riscrittura $x \cdot true \to x$ e $x \cdot x \to x$. Esse ci permettono di eliminare ogni atomo ripetuto e ogni occorrenza di *true* in una congiunzione. Dal momento che il prodotto è associativo, commutativo ed idempotente, le congiunzioni di atomi sono *insiemi* di atomi: il lato sinistro $\bar{L}$ di una regola se o di un goal antenato semplifica un lato $\bar{R}$ del goal corrente se c'è un sottoinsieme di $\bar{R}$ che è un'istanza di $\bar{L}$. Non c'è bisogno di AC-matching, poichè tutte le operazioni di matching avvengono sotto il prodotto, tra coppie di letterali.

8.3 Una caratterizzazione a punto fisso dei programmi a riscrittura

I programmi a riscrittura hanno una semantica operazionale, una semantica logica e una semantica a punto fisso. Sia E un programma a riscrittura, sia $\mathcal{B}$ la sua base di Herbrand e sia $\mathcal{P}(\mathcal{B})$ l'insieme di tutti i sottoinsiemi di $\mathcal{B}$, cioè l'insieme di tutte le interpretazioni di Herbrand. La semantica operazionale di E è il suo *insieme di successo*, $\{G \mid G \in \mathcal{B},\ E \vdash_{LC} G\}$. La semantica logica è l'insieme $\{G \mid G \in \mathcal{B},\ E^* \models G \simeq true\}$, dove $E^* = E \cup \{x \cdot x \to x,\ x \cdot true \to x\}$. Se $E \vdash_{LC} Q_1 \dots Q_m\sigma$, diciamo che σ è una *sostituzione risposta corretta* per il quesito $Q_1 \dots Q_m$ se $E^* \models Q_1 \dots Q_m\sigma \simeq true$.

Definiamo una *semantica a minimo punto fisso* dei programmi a riscrittura sul reticolo $\mathbb{B} = \{I' \mid I' = I \cup \{true\}, I \subseteq \mathcal{B}\}$. $\mathbb{B}$ è $\mathcal{P}(\mathcal{B})$ dove l'elemento *true* è stato aggiunto a ogni sottoinsieme di $\mathcal{B}$. La relazione d'ordine su $\mathbb{B}$ è l'inclusione di insiemi $\subseteq$, l'operazione di estremo inferiore è l'intersezione $\bigcap$ e l'operazione di estremo superiore è l'unione $\bigcup$. L'elemento minimo è $\{true\}$ e l'elemento massimo è $\mathcal{B} \cup \{true\}$. Si associa una funzione $T_E : \mathbb{B} \to \mathbb{B}$ a un programma E nel modo seguente:

Definizione 8.3.1 *Dato un programma a riscrittura E, la sua funzione associata è la funzione*

$T_E : \mathbb{B} \to \mathbb{B}$ *tale che* $P \in T_E(I)$ *se e solo se esiste in* E *una regola* $A_1 \dots A_n \leftrightarrow B_1 \dots B_m$ $(n \geq 1, m \geq 1)$ *tale che* $P = A_i\sigma$ *e* $\{A_1\sigma, \dots, A_{i-1}\sigma, A_{i+1}\sigma, \dots, A_n\sigma, B_1\sigma, \dots, B_m\sigma\} \subseteq I$ *per qualche* i, $1 \leq i \leq n$, *e qualche sostituzione ground* σ. *(La doppia freccia* $\leftrightarrow$ *significa qui che non c'è nessuna distinzione tra il lato sinistro e il lato destro.)*

Lemma 8.3.1 *Dato un programma a riscrittura* E, T_E *è continua, cioè per ogni catena non decrescente* $X_1 \subseteq X_2 \subseteq \dots$ *di elementi di* $\mathbb{B}$, $T_E(\bigcup\{X_i \mid i < \omega\}) = \bigcup\{T_E(X_i) \mid i < \omega\}$.

Dimostrazione: sia $X_1 \subseteq X_2 \subseteq \dots$ una catena in $\mathbb{B}$.

1. $T_E(\bigcup\{X_i \mid i < \omega\}) \subseteq \bigcup\{T_E(X_i) \mid i < \omega\}$:
   $P \in T_E(\bigcup\{X_i \mid i < \omega\})$ se e solo se esiste un'istanza ground $A_1\sigma \dots A_{j-1}\sigma P A_{j+1}\sigma \dots A_n\sigma \leftrightarrow B_1\sigma \dots B_m\sigma$ di una regola in E e $\{A_1\sigma \dots A_{j-1}\sigma, A_{j+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq \bigcup\{X_i \mid i < \omega\}$. Allora $\{A_1\sigma \dots A_{j-1}\sigma, A_{j+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq X_i$ per qualche i, così che $P \in T_E(X_i)$ e $P \in \bigcup\{T_E(X_i) \mid i < \omega\}$.
2. $\bigcup\{T_E(X_i) \mid i < \omega\} \subseteq T_E(\bigcup\{X_i \mid i < \omega\})$:
   $P \in \bigcup\{T_E(X_i) \mid i < \omega\}$ se e solo se $P \in T_E(X_i)$ per qualche i se e solo se esiste un'istanza ground $A_1\sigma \dots A_{j-1}\sigma P A_{j+1}\sigma \dots A_n\sigma \leftrightarrow B_1\sigma \dots B_m\sigma$ e $\{A_1\sigma \dots A_{j-1}\sigma, A_{j+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq X_i$. Poiché $X_i \subseteq \bigcup\{X_i \mid i < \omega\}$, segue che $P \in T_E(\bigcup\{X_i \mid i < \omega\})$. □

Segue che T_E ha le proprietà delle funzioni continue su un reticolo. In particolare il minimo punto fisso di T_E è una potenza ordinale di T_E:

$$lfp(T_E) = T_E \uparrow \omega$$

dove le potenze ordinali sono definite su $\mathbb{B}$ nel solito modo:

$$T \uparrow 0 = \{true\}$$

$$T \uparrow n = \begin{cases} T(T \uparrow (n-1)) & \text{se } n \text{ è un ordinale successore} \\ \bigcup\{T \uparrow k \mid k < n\} & \text{se } n \text{ è un ordinale limite.} \end{cases}$$

Esempio 8.3.1 *Il minimo punto fisso del programma a riscrittura per la relazione ancestor è dato da:*

$T_E(\{true\}) = \{true, parent(jb, lc), parent(jb, gg), parent(gg, wm)\}$

$T_E^2(\{true\}) = T_E(\{true\}) \cup \{ancestor(jb, lc), ancestor(jb, gg), ancestor(gg, wm)\}$

$lfp(T_E) = T_E^3(\{true\}) = T_E^2(\{true\}) \cup \{ancestor(jb, wm)\}$.

8.4 Equivalenza di semantica operazionale, semantica logica e semantica a punto fisso

Mostriamo che $lfp(T_E)$ è una caratterizzazione a punto fisso della semantica operazionale e della semantica logica di un programma E. Si ottengono questi risultati attraverso alcuni passi. Il primo è il lemma seguente:

Lemma 8.4.1 *Per tutte le congiunzioni di atomi* $Q_1 \dots Q_m$, $Q_1 \dots Q_m \leftrightarrow^*_{E^*} true$ *se e solo se* $\forall j, 1 \leq j \leq m$, $Q_j \leftrightarrow^*_{E^*} true$.

Dimostrazione:

$\Rightarrow$) La prova è per induzione sulla lunghezza i della catena $Q_1 \dots Q_m \leftrightarrow^i_{E^*} true$.

Base: se $i = 1$ allora $m = 1$ e la tesi è banalmente vera.

Ipotesi di induzione: $\forall i, 1 < i \leq l$, $Q_1 \dots Q_m \leftrightarrow^i_{E^*} true$ implica $\forall j, 1 \leq j \leq m$ $Q_j \leftrightarrow^*_{E^*} true$.

Passo di induzione: se $i = l + 1$, allora abbiamo

o il caso 1: $Q_1 \dots Q_m \rightarrow_{E^*} C \leftrightarrow^l_{E^*} true$

o il caso 2: $Q_1 \dots Q_m \leftarrow_{E^*} C \leftrightarrow^l_{E^*} true$:

1. se si applica una regola fatto $A \rightarrow true$ tale che $Q_1 = A\sigma$, allora C è $Q_2 \dots Q_m$. L'atomo Q_1 viene riscritto a *true* dalla regola fatto. L'ipotesi di induzione si applica ai rimanenti atomi $Q_2 \dots Q_m$.
   Similmente, se si applica a Q_1 una regola se $AB_1 \dots B_n \rightarrow B_1 \dots B_n$, C è $Q_2 \dots Q_m$. L'ipotesi di induzione si applica agli atomi $Q_2 \dots Q_m$. Per Q_1 abbiamo $Q_1 \leftrightarrow^*_{E^*} Q_1 true \dots true \leftrightarrow^*_{E^*} Q_1 Q_2 \dots Q_m \leftrightarrow^*_{E^*} true$ così che Q_1 è E^*-equivalente a *true*.
   Se si applica a Q_1 una regola sse $A \rightarrow B_1 \dots B_n$, C è $(B_1 \dots B_n)\sigma Q_2 \dots Q_m$. L'ipotesi di induzione si applica a tutti gli atomi in C, così che $\forall k, 2 \leq k \leq m, Q_k \leftrightarrow^*_{E^*} true$ e $Q_1 \rightarrow_{E^*} (B_1 \dots B_n)\sigma \leftrightarrow^*_{E^*} true$.
2. Se C viene riscritto a $Q_1 \dots Q_m$ da una regola fatto o una regola se, allora per l'ipotesi di induzione, $\forall j, 1 \leq j \leq m$ $Q_j \leftrightarrow^*_{E^*} true$, poiché $\{Q_1 \dots Q_m\} \subset C$.
   Se si applica a C una regola sse $A \rightarrow B_1 \dots B_n$, allora se C è $D_1 \dots D_p$, $Q_1 \dots Q_m$ è $(B_1 \dots B_n)\sigma D_2 \dots D_p$. L'ipotesi di induzione si applica agli atomi $D_2 \dots D_p$. Restano gli atomi in $(B_1 \dots B_n)\sigma$. L'atomo D_1 ha un predicato definito in modo mutualmente esclusivo, siccome è riscritto da una regola sse. Segue che la stessa regola sse $A \rightarrow B_1 \dots B_n$ deve essere usata per ridurre D_1 nella catena di uguaglianze tra C e *true*. Il prodotto $(B_1 \dots B_n)\sigma$ viene ridotto a *true* mediante un cammino più corto di l passi, così che l'ipotesi di induzione si applica anche a questi atomi.

$\Leftarrow$) Banale. □

Il teorema seguente stabilisce la soundness di Linear Completion, ovvero che tutte le risposte date da Linear Completion sono corrette sostituzioni risposta:

Teorema 8.4.1 *Se* $E \vdash_{LC} Q_1 \dots Q_m\sigma$ *allora* $E^* \models Q_1 \dots Q_m\sigma \simeq true$.

Dimostrazione: la soundness di Linear Completion segue dalla soundness della sostituzione di uguali con uguali, dal momento che le regole di inferenza di Linear Completion sono applicazioni di sostituzione equazionale.

$E \vdash_{LC} Q_1 \dots Q_m\sigma$ sta per $E \cup \{Q_1 \dots Q_m \rightarrow answer(\bar{x})\} \vdash_{LC} answer(\bar{x})\sigma \rightarrow true$. LC deriva l'equazione $answer(\bar{x})\sigma \rightarrow true$ da E e $Q_1 \dots Q_m \rightarrow answer(\bar{x})$. Siccome il letterale *answer* è solo

un segnaposto per $Q_1 \dots Q_m$, questo significa in pratica che l'equazione $Q_1 \dots Q_m\sigma \to true$ è stata derivata da E per sostituzione equazionale, o $Q_1 \dots Q_m\sigma \leftrightarrow^*_{E^*} true$. Allora, $Q_1 \dots Q_m\sigma \leftrightarrow^*_{E^*} true$ implica $E^* \models Q_1 \dots Q_m\sigma \simeq true$ per la soundness della sostituzione di uguali con uguali. □

Restringiamo ora la nostra attenzione ai quesiti ground. Innanzitutto dimostriamo un lemma che ci permette di dividere il problema di provare una congiunzione ground nei sottoproblemi di provare i suoi singoli atomi. Intuitivamente questo rislutato vale perché siccome il quesito è ground, non c'è computazione di una sostituzione risposta e i processi di riduzione a *true* dei letterali nel quesito sono processi indipendenti.

Lemma 8.4.2 $\forall Q_1 \dots Q_m \in \mathcal{B}$, $E \vdash_{LC} Q_1 \dots Q_m$ *se e solo se* $\forall j$, $1 \le j \le m$, $E \vdash_{LC} Q_j$.

Dimostrazione:

$\Leftarrow$) La prova è per assurdo. Assumiamo che $E \vdash_{LC} Q_1 \dots Q_m$ non valga. Questo significa che tutti i cammini nell'albero generato da LC a partire dal quesito $Q_1 \dots Q_m$ terminano con una regola goal che non è una regola risposta e tale che nessun ulteriore passo può essere eseguito. Tale goal contiene almeno un letterale ground A che non è nè $answer(\ )$ nè $true$. Si noti che dal momento che il quesito è ground, tutti i letterali *answer* sono identici a $answer(\ )$ e ogni goal può contenere al più uno di essi a causa dell'applicazione di $x \cdot x \to x$.

Giacché tutte le regole goal sono ground, un passo di semplificazione di un goal con un goal non genera nuove istanze. Segue che questo letterale A è stato generato da una sequenza di passi con regole di programma. Allora, A viene generato anche nell'albero che ha per radice un quesito Q_j per qualche j, $1 \le j \le m$. Poiché $E \vdash_{LC} Q_j$, il programma risolve A contraddicendo la nostra assunzione che A non possa essere risolto

$\Rightarrow$) La prova è per induzione sulla lunghezza i della derivazione $E \vdash^i_{LC} Q_1 \dots Q_m$.

Base: se $i = 1$, allora $m = 1$ e il risultato segue.

Ipotesi di induzione: $\forall i$, $1 < i \le l$, $E \vdash^i_{LC} Q_1 \dots Q_m$ implica $\forall j$, $1 \le j \le m$, $E \vdash_{LC} Q_j$.

Passo di induzione: se $i = l + 1$, allora una regola di programma deve essere stata applicata:

1. se si applica una regola sse $A \to B_1 \dots B_n$ tale che $A\sigma = Q_1$, $E \vdash^l_{LC} (B_1 \dots B_n)\sigma Q_2 \dots Q_m$. Per l'ipotesi di induzione otteniamo $E \vdash_{LC} Q_j$ per $2 \le j \le m$ ed $E \vdash_{LC} B_k\sigma$ per $1 \le k \le n$. Siccome una regola sse si applica a Q_1, il suo predicato è definito in modo mutualmente esclusivo. Segue che se si dà un quesito costituito da Q_1 solamente, esso viene necessariamente ridotto a $(B_1 \dots B_n)\sigma$. Dal momento che tutti i $B_k\sigma$ vengono risolti, $E \vdash_{LC} (B_1 \dots B_n)\sigma$ ed $E \vdash_{LC} Q_1$ segue.
2. Se si applica una regola se $AB_1 \dots B_n \to B_1 \dots B_n$ tale che $A\sigma = Q_1$, significa che $Q_1 \dots Q_m$ è $(AB_1 \dots B_n)\sigma Q_{n+2} \dots Q_m$ ed $E \vdash^l_{LC} (B_1 \dots B_n)\sigma Q_{n+2} \dots Q_m$. Per l'ipotesi di induzione otteniamo $E \vdash_{LC} Q_j$ per $n + 2 \le j \le m$ ed $E \vdash_{LC} B_k\sigma$ per $1 \le k \le n$. Manca soltanto Q_1. Tuttavia, se si dà un quesito costituito da Q_1 solamente, esiste una computazione che comincia sovrapponendo Q_1 con $AB_1 \dots B_n \to B_1 \dots B_n$ riducendolo al quesito $(B_1 \dots B_n)\sigma$. Poiché tutti i $B_k\sigma$ sono risolti, $E \vdash_{LC} (B_1 \dots B_n)\sigma$ ed $E \vdash_{LC} Q_1$ segue. □

Possiamo ora applicare il Lemma 8.4.2 per mostrare che tutti quesiti ground provati da Linear

Completion da E sono equivalenti a *true*.

Teorema 8.4.2 *$\forall Q_1 \dots Q_m \in \mathcal{B}$, $E \vdash_{LC} Q_1 \dots Q_m$ se e solo se $Q_1 \dots Q_m \leftrightarrow^*_{E^*} true$.*

Dimostrazione:

$\Rightarrow$) Vedi il teorema di soundness (Teorema 8.4.1).

$\Leftarrow$) La prova è per induzione sulla lunghezza i della catena $Q_1 \dots Q_m \leftrightarrow^i_{E^*} true$.

Base: se $i = 1$, allora $m = 1$ e c'è una regola fatto $A \rightarrow true$ tale che $Q_1 = A\sigma$. Segue che $E \vdash_{LC} Q_1$ in un passo.

Ipotesi di induzione: $\forall i, 1 < i \leq l$, $Q_1 \dots Q_m \leftrightarrow^i_{E^*} true$ implica $E \vdash_{LC} Q_1 \dots Q_m$.

Passo di induzione: se $i = l+1$, allora $Q_1 \dots Q_m \leftrightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ e consideriamo due casi secondo la direzione del primo passo:

1. se $Q_1 \dots Q_m \rightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$, allora l'interprete LC può derivare il goal $\bar{C}$ dal quesito $Q_1 \dots Q_m$ applicando questo passo di riscrittura. Poiché $E \vdash_{LC} \bar{C}$ vale per ipotesi di induzione, segue che $E \vdash_{LC} Q_1 \dots Q_m$.
2. Se $Q_1 \dots Q_m \leftarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$, allora ci sono ancora due casi.
   Se una regola se, o una regola fatto, o $x \cdot x \rightarrow x$, o $x \cdot true \rightarrow x$ riduce $\bar{C}$ a $Q_1 \dots Q_m$, tutti gli atomi Q_j, $1 \leq j \leq m$, occorrono in $\bar{C}$. Per l'ipotesi di induzione abbiamo che $E \vdash_{LC} \bar{C}$. Segue per il Lemma 8.4.2 che $E \vdash_{LC} Q_j$, $1 \leq j \leq m$ ed $E \vdash_{LC} Q_1 \dots Q_m$.
   Se una regola sse $A \rightarrow B_1 \dots B_n$ riduce $\bar{C}$ a $Q_1 \dots Q_m$, $\bar{C}$ è $C_1 \dots C_p$, $C_1 = A\sigma$ e $Q_1 \dots Q_m$ è $(B_1 \dots B_n)\sigma C_2 \dots C_p$. Per l'ipotesi di induzione abbiamo che $E \vdash_{LC} \bar{C}$. Segue per il Lemma 8.4.2 che $E \vdash_{LC} C_q$, $1 \leq q \leq p$. Siccome $C_1 = A\sigma$ ed A è la testa di una regola sse, il predicato di C_1 è definito in modo mutualmente esclusivo. Segue che nella computazione $E \vdash_{LC} C_1$ questa regola deve essere applicata ed $E \vdash_{LC} B_1 \dots B_n\sigma$. Segue per il Lemma 8.4.2 che $E \vdash_{LC} B_k\sigma$, $1 \leq k \leq n$. Ora tutti gli atomi in $Q_1 \dots Q_m$ sono provati così che applicando ancora il Lemma 8.4.2 otteniamo $E \vdash_{LC} Q_1 \dots Q_m$. □

Questi risultati ci permettono di collegare la semantica a punto fisso di un programma a riscrittura E alla sua semantica operazionale:

Teorema 8.4.3 *$\forall Q_1 \dots Q_m \in \mathcal{B}$, $Q_1 \dots Q_m \leftrightarrow^*_{E^*} true$ se e solo se $\forall i$, $1 \leq i \leq m$, $Q_i \in lfp(T_E)$.*

Dimostrazione:

$\Leftarrow$) Proviamo che se $Q_i \in lfp(T_E)$, allora $Q_i \leftrightarrow^*_{E^*} true$. Quindi $Q_1 \dots Q_m \leftrightarrow^*_{E^*} true$ segue.

Se $Q_i \in lfp(T_E)$, esiste un $j \geq 1$ tale che $Q_i \in T^j_E(\{true\})$ e $Q_i \notin T^k_E(\{true\})$ per tutti i $k < j$, cioè j è la minima potenza tale che Q_i è incluso. La prova è per induzione sulla potenza j.

Base: se $j = 1$, esiste una regola fatto $A \rightarrow true$ tale che $Q_i = A\sigma$ per qualche sostituzione ground σ. Segue che $Q_i \leftrightarrow_{E^*} true$.

Ipotesi di induzione: $\forall j, 1 < j \leq l$, $Q_i \in T^j_E(\{true\})$ implica $Q_i \leftrightarrow^*_{E^*} true$.

Passo di induzione: se $j = l + 1$, esiste un'istanza ground $A_1\sigma \dots A_{k-1}\sigma Q_i A_{k+1}\sigma \dots A_n\sigma \leftrightarrow B_1\sigma \dots B_m\sigma$ di una regola in E tale che $\{A_1\sigma \dots A_{k-1}\sigma, A_{k+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq T^l_E(\{true\})$.

Per l'ipotesi di induzione tutti gli atomi in questo insieme sono E^*-equivalenti a $true$. Segue che anche $Q_i \leftrightarrow^*_{E^*} true$.

$\Rightarrow$) La prova è per induzione sulla lunghezza j di $Q_1 \ldots Q_m \leftrightarrow^j_{E^*} true$.

Base: se $j = 1$, allora $m = 1$ e $Q_1 = A\sigma$ per una regola fatto $A \rightarrow true$. Segue che $Q_1 \in T_E(\{true\})$ e $Q_1 \in lfp(T_E)$.

Ipotesi di induzione: $\forall j, 1 < j \leq l$, $Q_1 \ldots Q_m \leftrightarrow^j_{E^*} true$ implica $Q_i \in lfp(T_E)$ for $1 \leq i \leq m$.

Passo di induzione: se $j = l + 1$, allora $Q_1 \ldots Q_m \leftrightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ e abbiamo due casi come nella dimostrazione del Teorema 8.4.2.

1. Si consideri $Q_1 \ldots Q_m \rightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$. Se $x \cdot x \rightarrow x$ o $x \cdot true \rightarrow x$ riduce $Q_1 \ldots Q_m$ a $\bar{C}$, tutti gli atomi Q_i, $1 \leq i \leq m$ occorrono in $\bar{C}$ e così per l'ipotesi di induzione essi sono tutti in $lfp(T_E)$. Se $A \rightarrow true$ riduce $Q_1 \ldots Q_m$ a $\bar{C}$, l'atomo Q_k che è un'istanza di A è in $T_E(\{true\})$ e quindi in $lfp(T_E)$. Tutti gli atomi rimanenti $Q_2 \ldots Q_m$ sono in $\bar{C}$ e quindi in $lfp(T_E)$ per l'ipotesi di induzione. Se si applica $AB_1 \ldots B_n \rightarrow B_1 \ldots B_n$, allora $(AB_1 \ldots B_n)\sigma = Q_1 Q_2 \ldots Q_{n+1}$ con $n + 1 < m$ e C è $Q_2 \ldots Q_m$. Gli atomi $Q_2 \ldots Q_m$ sono in $lfp(T_E)$ per l'ipotesi di induzione. Dal momento che $(B_1 \ldots B_n)\sigma = Q_2 \ldots Q_{n+1}$, gli atomi $B_1\sigma \ldots B_n\sigma$ sono in $lfp(T_E)$. Sia p la minima potenza tale che tutti i $B_k\sigma$ sono in $T^p_E(\{true\})$. Segue che $Q_1 \in T^{p+1}_E(\{true\})$ e quindi in $lfp(T_E)$. Se una regola sse $A \rightarrow B_1 \ldots B_n$ riduce $Q_1 \ldots Q_m$ a $\bar{C}$, allora $Q_1 = A\sigma$ e $\bar{C}$ è $(B_1 \ldots B_n)\sigma Q_2 \ldots Q_m$. Per l'ipotesi di induzione tutti gli atomi in $\bar{C}$ sono in $lfp(T_E)$. Sia p la minima potenza tale che tutti i $B_k\sigma$ sono in $T^p_E(\{true\})$. Segue che $Q_1 \in T^{p+1}_E(\{true\})$ e quindi in $lfp(T_E)$.
2. Se $Q_1 \ldots Q_m \leftarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ e una regola se, o una regola fatto, o $x \cdot x \rightarrow x$, o $x \cdot true \rightarrow x$ riduce $\bar{C}$ a $Q_1 \ldots Q_m$, tutti gli atomi Q_i, $1 \leq i \leq m$, appartengono a $\bar{C}$. Per l'ipotesi di induzione essi sono tutti in $lfp(T_E)$. Se si applica una regola sse $A \rightarrow B_1 \ldots B_n$, allora $\bar{C}$ è $C_1 \ldots C_h$, $C_1 = A\sigma$ e $Q_1 \ldots Q_m$ è $(B_1 \ldots B_n)\sigma C_2 \ldots C_h$. Per l'ipotesi di induzione abbiamo che tutti gli atomi in $\bar{C}$ sono in $lfp(T_E)$. Mancano solo gli atomi in $(B_1 \ldots B_n)\sigma$. Tuttavia, poiché si applica una regola sse, significa che il predicato di C_1 è definito in modo mutualmente esclusivo. Segue che si deve applicare questa regola nella catena $\bar{C} \leftrightarrow^l_{E^*} true$, ovvero la prova ha la forma $\bar{C} \leftrightarrow^{j_1}_{E^*} \ldots (B_1 \ldots B_n)\sigma \ldots \leftrightarrow^{j_2}_{E^*} true$, dove $j_2 < l$. Applicando l'ipotesi di induzione all'ultima sottocatena otteniamo che tutti i $B_k\sigma$ sono anch'essi in $lfp(T_E)$. □

Dunque, la semantica a punto fisso $lfp(T_E)$ cattura sia la sematica operazionale che la semantica logica del programma a riscrittura E:

Teorema 8.4.4 *$\forall G \in \mathcal{B}$, $G \in lfp(T_E)$ se e solo se $E \vdash_{LC} G$.*

Dimostrazione: $G \in lfp(T_E)$ se e solo se $G \leftrightarrow^*_{E^*} true$ per il Teorema 8.4.3 se e solo se $E \vdash_{LC} G$ per il Teorema 8.4.2. □

Teorema 8.4.5 *$\forall G \in \mathcal{B}$, $G \in lfp(T_E)$ se e solo se $E^* \models G \simeq true$.*

Dimostrazione: $G \in lfp(T_E)$ se e solo se $G \leftrightarrow^*_{E^*} true$ per il Teorema 8.4.3 se e solo se $E^* \models G \simeq true$ per la completezza della sostituzione di uguali con uguali (teorema di Birkhoff). □

8.5 Semantica denotazionale dei programmi a riscrittura

La caratterizzazione a punto fisso dei programmi a riscrittura è sostanzialmente equivalente alla caratterizzazione a punto fisso dei programmi Prolog. La semantica a punto fisso di un programma Prolog P è $lfp(T_P) = T_P \uparrow \omega$, dove T_P è la funzione $T_P : \mathcal{P}(\mathcal{B}) \to \mathcal{P}(\mathcal{B})$ tale che $A \in T_P(I)$ se e solo se esiste in P una clausola $A' :- B_1 \dots B_m$ $(m \geq 0)$ tale che $A = A'\sigma$ e $\{B_1\sigma \dots B_m\sigma\} \subseteq I$ per qualche sostituzione ground σ [64, 6]. Il teorema seguente mostra che infatti le due semantiche concordano. Scriviamo $E \equiv P$ e diciamo che il programma a riscrittura E corrisponde al programma Prolog P se essi definiscono gli stessi predicati.

Teorema 8.5.1 *Se $E \equiv P$, allora $lfp(T_E) = lfp(T_P)$.*

Dimostrazione:

1. $lfp(T_E) \subseteq lfp(T_P)$.
   Se $G \in lfp(T_E)$, allora esiste un $i \geq 1$ tale che $G \in T_E^i(\{true\})$ e $\forall j < i, G \notin T_E^j(\{true\})$. La prova è per induzione sulla potenza i.
   Base: se $i = 1$, allora esiste una regola fatto $A \to true$ in E tale che $G = A\sigma$ per qualche sostituzione ground σ. Poiché $E \equiv P$, il fatto $A.$ appartiene al programma Prolog P e $G \in lfp(T_P)$.
   Ipotesi di induzione: $\forall i, 1 < i \leq l$, $G \in T_E^i(\{true\})$ implica $G \in lfp(T_P)$.
   Passo di induzione: se $i = l + 1$, esiste un'istanza ground $C_1\sigma \dots C_{j-1}\sigma G C_{j+1}\sigma \dots C_p\sigma \leftrightarrow D_1\sigma \dots D_q\sigma$ di una regola in E i cui atomi eccetto G sono in $T_E^l(\{true\})$. Per l'ipotesi di induzione tutti gli atomi in questo insieme appartengono a $lfp(T_P)$. Segue che c'è qualche $k \geq 1$ tale che $\{C_1\sigma \dots C_{j-1}\sigma, C_{j+1}\sigma \dots C_p\sigma, D_1\sigma \dots D_q\sigma\} \subseteq T_P^k(\emptyset)$. La suddetta regola ground è o un'istanza di una regola se $AB_1 \dots B_n \to B_1 \dots B_n$ o un'istanza di una regola sse $A \to B_1 \dots B_n$. Siccome $P \equiv E$, in entrambi i casi il programma Prolog P contiene la clausola corrispondente $A :- B_1 \dots B_n$. Nel caso della regola se, se $G = B_h\sigma$ per qualche $h, 1 \leq h \leq n$, allora è in $lfp(T_P)$ per l'ipotesi di induzione, perché ogni B_h occorre due volte nella regola. Se $G = A\sigma$, allora $G \in T_P^{k+1}(\emptyset)$ e $G \in lfp(T_P)$. Nel caso della regola sse, se $G = A\sigma$, allora $G \in T_P^{k+1}(\emptyset)$ e $G \in lfp(T_P)$. Se $G = B_h\sigma$ per qualche $h, 1 \leq h \leq n$, allora $A\sigma$ è già in $T_P^k(\emptyset)$. Dal momento che il predicato di A è definito in modo mutualmente esclusivo, $A\sigma$ deve essere stato incluso in $T_P^k(\emptyset)$ a causa della clausola $A :- B_1 \dots B_n$ e $G = B_h\sigma$ deve essere in $T_P^z(\emptyset)$ per qualche $z < k$, così che $G \in lfp(T_P)$.
2. $lfp(T_P) \subseteq lfp(T_E)$.
   La prova è analoga alla precedente, applicando lo stesso argomento per induzione alla potenza di T_P e sfruttando la corrispondenza tra i due programmi. □

Questo teorema stabilisce che un programma Prolog e un programma a riscrittura che definiscono gli stessi predicati hanno lo stesso modello. Per un atomo ground G, $E \vdash_{LC} G$, $G \leftrightarrow_{E^*}^* true$, $E^* \models G \simeq true$, $G \in lfp(T_E)$, $G \in lfp(T_P)$, $P \models G$ e $P \vdash_{Prolog} G$ sono tutti equivalenti. Però, il comportamento dei due programmi P and E può essere differente. Il programma a riscrittura E può generare meno risposte del corrispondente programma Prolog P. Tuttavia, per tutte le

risposte date da P c'è una risposta data da E che è E-equivalente. Al fine di ottenere questo risultato, dimostriamo innanzitutto che tutte le risposte date da Linear Completion vengono date anche da Prolog. Questo risultato segue dalla completezza della SLD-risoluzione.

Teorema 8.5.2 *Se $E \equiv P$ e $E \vdash_{LC} Q_1 \dots Q_m\sigma$, esiste una risposta θ, $P \vdash_{Prolog} Q_1 \dots Q_m\theta$, tale che $\sigma = \theta\rho$ per qualche sostituzione ρ.*

Dimostrazione: se $E \vdash_{LC} Q_1 \dots Q_m\sigma$, allora σ è una corretta sostituzione risposta per la soundness di Linear Completion. Quindi per la completezza della SLD-risoluzione esistono una sostituzione risposta computata da Prolog θ, $P \vdash_{Prolog} Q_1 \dots Q_m\theta$ e una sostituzione ρ, tali che $\sigma = \theta\rho$. □

Dimostriamo ora che Linear Completion è anch'essa completa, provando che tutte le risposte date da Prolog sono rappresentate da qualche risposta data da Linear Completion.

Nel seguito assumiamo che tutti i quesiti siano quesiti di un singolo letterale. Non si perde in generalità perché si può scrivere un quesito $Q_1 \dots Q_m \to answer(\bar{x})$ come un quesito di un singolo letterale introducendo un nuovo simbolo di predicato N, una nuova regola di programma $N \to Q_1 \dots Q_m$ e il quesito $N \to answer(\bar{x})$.

Innanzitutto proviamo che i programmi a riscrittura e i programmi Prolog danno le stesse risposte se Linear Completion è ristretta alla sola sovrapposizione, ovvero non si esegue nessuna semplificazione. Questo è immediato, dal momento che i passi di sovrapposizione e i passi di risoluzione corrispondono:

Teorema 8.5.3 *Sia LC' un sottoinsieme dell'interprete di Linear Completion che esegua soltanto passi di sovrapposizione. Se $E^* \models G\theta \simeq true$, cioè θ è una risposta corretta per G, esiste una risposta σ, computata da LC', $E \vdash_{LC'} G\sigma$, tale che $\theta = \sigma\rho$ per qualche sostituzione ρ.*

Dimostrazione: proviamo che se $E \vdash_{LC'} G\sigma$ allora $P \vdash_{Prolog} G\sigma$ e vice versa. La completezza di LC' segue quindi dalla completezza della SLD-risoluzione. I passi di sovrapposizione corrispondono ai passi di risoluzione e vice versa. A ogni passo si generano gli stessi unificatori. Siccome una risposta è data dalla composizione degli unificatori generati dai passi di sovrapposizione/risoluzione, i due programmi generano le stesse sostituzioni risposta. Questo vale indipendentemente dal fatto che un passo Prolog corrisponda a un passo di sovrapposizione con una regola se, o a un passo di sovrapposizione con una regola sse. Se si sovrappone al goal corrente una regola sse, il corpo della regola viene aggiunto soltanto al lato sinistro del nuovo goal, mentre un passo di sovrapposizione con una regola se aggiunge il corpo a entrambi i lati del nuovo goal. Comunque, l'occorrenza di un atomo su entrambi i lati di un goal è irrilevante per quanto concerne passi di sovrapposizione e generazione di unficatori. Quando l'occorrenza sinistra di un atomo viene eliminata con un passo di sovrapposizione, la sua occorrenza destra viene istanziata dall'mgu del passo di sovrapposizione e sarà prima o poi eliminata da un passo di sovrapposizione il cui mgu non istanzia le variabili del goal. Si noti che non si verificano sovrapposizioni con regole risposta, dal momento che atomi *answer* possono presentarsi nel lato sinistro di un goal soltanto come effetto di precedenti semplificazioni con goals, che non sono eseguite da LC'. □

Al fine di provare un risultato analogo per Linear Completion con la semplificazione ci occorre provare prima due lemmi.

Lemma 8.5.1 *Se Linear Completion genera un cammino* $(E; G \rightarrow answer(\bar{x}); \emptyset) \vdash^*_{LC} (E; \bar{W} \rightarrow \bar{V}; \_) \vdash^*_{LC} (E; H; \_) \vdash_{LC} (E; \bar{R} \leftrightarrow \bar{R}; \_)$*, dove il goal* H *è ridotto a un'identità dal suo antenato* $\bar{W} \rightarrow \bar{V}$*, allora* H *è* $\bar{Z}(\bar{W}\lambda) \rightarrow \bar{Z}(\bar{V}\lambda)$ *per qualche sostituzione* λ *e letterali* $\bar{Z}$*. (Il tratto* $\_$ *in* $(E; H; \_)$ *significa che la terza componente non è rilevante.)*

Dimostrazione: sia $H\ \bar{S}\bar{X} \rightarrow \bar{S}\bar{Y}$, dove $\bar{S}$ sono i letterali che occorrono su entrambi i lati, i letterali in $\bar{X}$ occorrono sul lato sinistro soltanto e i letterali in $\bar{Y}$ occorrono sul lato destro soltanto. Se $\bar{W} \rightarrow \bar{V}$ si applica a entrambi i lati di H, allora un sottoinsieme di $\bar{S}$ è un'istanza di $\bar{W}$, cioè $\bar{S} = (\bar{W}\lambda)\bar{Z}$ per qualche sostituzione λ e $\bar{W} \rightarrow \bar{V}$ riscrive H in $(\bar{V}\lambda)\bar{Z}\bar{X} \rightarrow (\bar{V}\lambda)\bar{Z}\bar{Y}$ contro l'ipotesi che $\bar{W} \rightarrow \bar{V}$ riscriva H a un'identità. Segue che $\bar{W} \rightarrow \bar{V}$ si applica a un lato di H solamente.

Assumiamo che $\bar{W} \rightarrow \bar{V}$ si applichi al lato sinistro di H. Il caso in cui si applica al lato destro è simmetrico. Segue che H viene ridotto a $\bar{S}\bar{Y} \rightarrow \bar{S}\bar{Y}$. Se un sottoinsieme di $\bar{X}$ è un'istanza di $\bar{W}$, allora il sottoinsieme di $\bar{X}$ non coinvolto apparirebbe ancora sul lato sinistro e non sul lato destro dopo il passo di semplificazione, contraddicendo l'ipotesi che $\bar{W} \rightarrow \bar{V}$ riscriva H a un'identità. Segue che l'istanza di $\bar{W}$ deve coprire l'intero $\bar{X}$ e possibilmente anche un sottoinsieme di $\bar{S}$: abbiamo $\bar{W} = \bar{W}_1\bar{W}_2$, $\bar{S} = (\bar{W}_1\lambda)\bar{Z}$ e $\bar{X} = (\bar{W}_2\lambda)$. Il lato sisnistro di H è $(\bar{W}_1\lambda)\bar{Z}(\bar{W}_2\lambda)$ ed è riscritto a $\bar{Z}(\bar{V}\lambda)$. Inoltre, $\bar{Z}(\bar{V}\lambda) = \bar{S}\bar{Y}$, dal momento che il lato sisnistro di H deve essere riscritto al suo lato destro. Siccome $\bar{S} = (\bar{W}_1\lambda)\bar{Z}$, segue che $(\bar{V}\lambda) = (\bar{W}_1\lambda)\bar{Y}$, cioè $\bar{V} = \bar{W}_1\bar{W}_3$, dove $\bar{W}_3\lambda = \bar{Y}$. Segue che $\bar{W} \rightarrow \bar{V}$ è $\bar{W}_1\bar{W}_2 \rightarrow \bar{W}_1\bar{W}_3$ e H è $\bar{Z}(\bar{W}_1\bar{W}_2\lambda) \rightarrow \bar{Z}(\bar{W}_1\bar{W}_3\lambda)$. □

Il lemma seguente mostra che Linear Completion con la semplificazione è refutazionalmente completa. Innanzitutto, introduciamo la terminologia usata nella prova: un cammino che parte dalla radice, cioè il quesito, e termina con una risposta è un *cammino di successo*. Un cammino che parte dalla radice e termina con un goal tale che nessuna regola di inferenza è applicabile è un *cammino fallimentare*. Ci sono tre tipi di cammino fallimentare in un LC-albero:

- cammini fallimentari dove il goal è ridotto a un'identità, eliminata poi da un passo di *Eliminazione*,
- cammini fallimentari dove il goal è ridotto a un'equazione contenente soltanto letterali *answer* ai quali nessuna regola di inferenza è applicabile e
- cammini fallimentari dove l'ultimo goal contiene ancora letterali il cui predicato non è *answer*, ma nessuna regola del programma è applicabile.

I primi due tipi di cammino fallimentare sono determinati dalla semplificazione e quindi non compaiono in alberi prodotti da LC'. Il terzo tipo corrisponde alla nozione di *fallimento finito* in Prolog ed è il solo tipo di cammino fallimentare che possa apparire in un albero prodotto da LC'. Usiamo Δ per indicare o un'identità o un'equazione contenente soltanto letterali *answer* ai quali nessuna regola di inferenza è applicabile.

Lemma 8.5.2 *Dato un quesito* G*, se esiste una risposta* θ *generata da* LC'*, ovvero* $E \vdash_{LC'} G\theta$*, allora esiste una risposta* σ *generata da* LC*, cioè* $E \vdash_{LC} G\sigma$*.*

Dimostrazione: la prova è per assurdo. Assumendo che LC non generi nessuna risposta, consideriamo l'albero T_{LC} generato da LC per il quesito G. Siccome LC non genera nessuna risposta,

tutti i cammini in T_{LC} sono fallimentari. LC differisce da LC' per la semplificazione dei goals da parte delle regole di programma, degli antenati e delle risposte precedentemente generate. La semplificazione mediante regole nel programma non è rilevante poiché un passo di semplificazione con una regola di programma è un passo di sovrapposizione dove l'mgu è una sostituzione matching. Semplificazione e sovrapposizione con risposte precedentemente generate non si verificano perché assumiamo che LC non generi nessuna risposta per il quesito G. Dunque, dobbiamo considerare soltanto il caso dove tutti i cammini di successo generati da LC' sono interrotti dalla semplificazione con goals antenati in LC. Più precisamente, tutti i cammini di successo generati da LC' sono rimpiazzati per effetto della semplificazione con antenati da cammini fallimentari che terminano con un goal Δ:

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC} (E; H'; \_) \vdash^*_{LC} (E; H; \_) \vdash_{LC} (E; \Delta; \_)$,

dove H' semplifica H in Δ.

Per ogni tale cammino α, denotiamo con $I(\alpha)$ l'insieme di tutti i cammini di successo nell'albero generato da LC' che hanno il prefisso $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_)$, cioè tutti i cammini di forma:

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$.

Dato un insieme di cammini A, denotiamo con $min(A)$ il cammino più corto in A. Se non è unico, selezioniamo semplicemente uno tra i più corti. Quindi $min(I(\alpha))$ è il cammino più corto nell'insieme $I(\alpha)$. Consideriamo il cammino α^*, definito nel modo seguente:

$\alpha^* = min(\{min(I(\alpha)) | \alpha \in T_{LC}\})$.

α^* ha la forma

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$.

Per le nostre assunzioni, α^* è interrotto da semplificazione del goal in T_{LC}. Mostriamo che se questo è il caso, allora LC' genera un cammino di successo più corto di α^*, che è una contraddizione. Ci sono due casi:

caso 1: il goal H è ridotto dal suo antenato H' a un'identità,

caso 2: il goal H è ridotto dal suo antenato H' a un'equazione contenente soltanto letterali *answer* ai quali nessuna regola di inferenza è applicabile.

1. Per il Lemma 8.5.1, segue che se H' è $\bar{W} \to \bar{V}$, allora H è $\bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda)$ per qualche sostituzione λ e letterali $\bar{Z}$. Sappiamo che $(E; \bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. I letterali $\bar{Z}$ devono essere eliminati per ottenere una soluzione. Siccome l'ordine di selezione dei letterali per sovrapposizione/risoluzione non altera le risposte generate da LC' o da Prolog, possiamo modificare l'ordine dei passi nel cammino α^* in modo tale che i letterali in $\bar{Z}$ siano eliminati per primi. Questo non modifica la lunghezza del cammino. Segue che α^* è $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to \bar{V}; \_) \vdash^*_{LC'} (E; \bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda); \_) \vdash^*_{LC'} (E; \bar{W}\lambda \to \bar{V}\lambda; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. Un cammino siffatto non può essere il cammino più corto alla soluzione. Se un letterale $P(\bar{t})\lambda$ in $\bar{W}\lambda$ unifica con la testa di una regola $P(\bar{s})$, anche $P(\bar{t})$ in $\bar{W}$ e $P(\bar{s})$ unificano. Segue che c'è un altro cammino di successo $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to \bar{V}; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta' \to true; \_)$ ottenuto appli-

cando a $\bar{W} \to \bar{V}$ le stesse clausole applicate a $\bar{W}\lambda \to \bar{V}\lambda$ per ottenere la soluzione. Questo cammino è più corto di α^*.

2. Assumiamo ora che H' riduca H a un'equazione contenente soltanto letterali *answer*. Segue che H' ha la forma $\bar{W} \to answer(\_)$ e H è $\bar{W}\lambda \to answer(\_)$ o $\bar{W}\lambda answer(\_) \to \bar{W}\lambda$. Ci può essere più di un letterale *answer* in H' e H, ma questo è irrilevante. Segue che α^* è $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to answer(\_); \_) \vdash^*_{LC'} (E; \bar{W}\lambda \to answer(\_); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. Per lo stesso argomento applicato nel caso precedente, c'è un cammino di successo più corto $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to answer(\_); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta' \to true; \_)$. □

Possiamo finalmente formulare il nostro risultato di completezza:

Teorema 8.5.4 *Sia E un programma sse. Se $E^* \models G\theta \simeq true$, cioè θ è una risposta corretta per G, esiste una risposta σ, generata da LC, $E \vdash_{LC} G\sigma$, tale che $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$ per qualche sostituzione ρ.*

Dimostrazione: se $E^* \models G\theta \simeq true$, allora per completezza di LC' esiste una risposta τ, computata da LC', $E \vdash_{LC'} G\tau$, tale che $G\theta = G\tau\rho$. Per il Lemma 8.5.2, esiste una risposta per lo stesso quesito computata da LC: $E \vdash_{LC} G\sigma$. $E \vdash_{LC'} G\tau$ implica che $G\tau \leftrightarrow^*_{E^*} true$. Similmente, $E \vdash_{LC} G\sigma$ implica $G\sigma \leftrightarrow^*_{E^*} true$. Segue che $G\tau \leftrightarrow^*_{E^*} G\sigma$. Allora $G\tau\rho \leftrightarrow^*_{E^*} G\sigma\rho$ o $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$. □

Teorema 8.5.5 *Se $E \equiv P$, se $P \vdash_{Prolog} G\theta$, allora esiste una risposta σ data da Linear Completion, $E \vdash_{LC} G\sigma$, tale che $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$ per qualche sostituzione ρ.*

Dimostrazione: se $P \vdash_{Prolog} G\theta$, allora θ è una sostituzione risposta corretta per la soundness della SLD-risoluzione. Allora per il Teorema 8.5.4 esiste una sostituzione risposta σ, computata da LC, $E \vdash_{LC} G\sigma$, e una sostituzione ρ, tale che $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$. □

8.6 Confronto con alcuni meccanismi di controllo dei cicli in Prolog

I sistemi di riscrittura sono stati applicati in programmazione sia logica che funzionale e c'è una vasta letteratura su questo soggetto. In questo capitolo abbiamo ristretto la nostra attenzione a una classe speciale di sistemi di riscrittura interpretati da Linear Completion. Le regole di riscrittura sono *equivalenze* tra congiunzioni di atomi. Quindi, il linguaggio è relazionale come il Prolog puro. Il sistema di inferenza di Linear Completion include la semplificazione e la sovrapposizione. Un passo di sovrapposizione intuitivamente corrisponde a un passo di risoluzione e perciò abbiamo accuratamente evitato di chiamarlo *narrowing*. Il nome narrowing venne introdotto per la prima volta in [75] e si riferisce a un passo di paramodulazione mediante un'equazione orientata i cui lati sono termini. IL nostro metodo non è in rapporto con quelli che usano narrowing dal momento che non usiamo equazioni o regole di riscrittura tra termini del prim'ordine. E' più vicino invece a quelli dati in [29, 30, 31, 73]. L'approccio proposto nel primo di questi tre articoli non permette la semplificazione, quindi non può utilizzare pienamente le possibilità della riscrittura. Il nostro

approccio è simile a quello in [73], ma in [73] non è studiato il caso in cui un predicato è definito da implicazioni e non da equivalenze logiche, nè è data la semantica del metodo.

Uno degli effetti positivi della semplificazione nell'esecuzione di programmi logici è la capacità di evitare certi cicli. Tecniche per riconoscere ed evitare cicli nell'esecuzione di programmi Prolog hanno ricevuto considerevole attenzione. Qui ci limitiamo a considerare i risultati in [7, 17] poiché i loro meccanismi di controllo dei cicli sono basati sulla *sussunzione di goals dai loro antenati* e quindi possono dare effetti di limitazione dello spazio di ricerca simili a quelli della semplificazione.

I meccanismi di controllo dei cicli sono descritti in [7, 17] secondo la seguente terminologia: un meccanismo di controllo dei cicli L si dice

- *completo*, se elimina tutte le derivazioni infinite,
- *sound*, se nessuna sostituzione risposta viene persa, ovvero, per tutte le risposte σ a un quesito G generate da Prolog, c'è una risposta σ' generata da Prolog con L tale che $G\sigma = G\sigma'\rho$ per qualche sostituzione ρ,
- *debolmente sound*, se tutte le volte che ci sono risposte generate da Prolog per un quesito G, c'è almeno una risposta per G generata da Prolog con L.

La completezza è un requisito molto forte e infatti è stato provato in [7] che non esistono meccanismi di controllo dei cicli per programmi Prolog che siano completi e debolmente sound, anche se si impone la restrizione che i programmi non contengano simboli di funzione.

La soundness debole non sembra essere una proprietà sufficientemente significativa: garantisce che un controllo dei cicli che sia debolmente sound conservi almeno una risposta, ma non dà nessuna informazione su una possibile relazione tra le risposta generata e quelle che non vengono date. Il nostro teorema 8.5.5 invece, stabilisce qualcosa di più della soundness debole, poiché dice che per tutte le risposte Prolog σ a un quesito G, esiste una risposta θ di Linear Completion tale che $G\sigma \leftrightarrow^*_{E^*} G\theta\rho$ per qualche sostituzione ρ.

In [7, 17] si introducono tre tipi di meccanismi di controllo dei cicli. Il primo è chiamato controllo *Contains a Variant/Instance of Atom (CVA/CIA)*. L'idea è di interrompere una derivazione se il goal corrente contiene un atomo che è sussunto da un atomo occorrente in un goal antenato. Come osservato in [7, 17], il controllo CVA/CIA non è nemmeno debolmente sound. Si considerino gli esempi di *append* che abbiamo presentato nella Sezione 8.1: Prolog con CVA/CIA interromperebbe la derivazione infinita del primo esempio di *append*, ma, diversamente da Linear Completion, arresterebbe senza risposta anche la computazione per il quesito

$$?- append(X, [b|Y], [a, b, c|Z]), size(X) > 3$$

del terzo esempio. Questo comportamento non è corretto e mostra che un meccanismo di controllo dei cicli che non sia almeno debolmente sound è inaccettabile. Intuitivamente, i controlli CVA/CIA non sono corretti perché controllano singoli atomi fuori dai loro contesti: la condizione per arrestare la derivazione si applica al letterale *append* senza prendere il letterale *size* in considerazione. D'altra parte, Linear Completion risponde al suddetto quesito perchè la semplificazione tiene in considerazione il contesto, dal momento che richiede che l'intero lato sinistro dell'antenato compaia istanziato nel goal corrente.

La seconda famiglia di controlli dei cicli introdotti in [7, 17] sostituisce "atomo" con "goal": una derivazione viene interrotta se il goal corrente è sussunto da uno dei suoi antenati. Questi controlli sono chiamati *Equal Variant of Goal (EVG)*, *Equal Instance of Goal (EIG)*, *Subsumed as Variant of Goal (SVG)* e *Subsumed as Instance of Goal (SIG)*, dove "equal" significa che il goal corrente è esattamente un'istanza di uno dei suoi antenati. Questi quattro controlli dei cicli sono debolmente sound [7, 17] e completi per una classe molto ristretta di programmi definita in [7, 17]. La soundness debole garantisce semplicemente che almeno una risposta venga generata. L'esempio seguente da [7] mostra che la soundness debole non è soddisfacente: dato il programma

$p(a)$.

$p(Y) :- p(Z)$.

e il quesito

$?- p(X)$.,

Prolog genera prima la risposta $\{X \mapsto a\}$, poi la risposta $\{X \mapsto Y\}$ e poi cade in un ciclo infinito. Prolog con uno qualsiasi dei summenzionati controlli debolmente sound è in grado di trovare soltanto la risposta a come si vede in Fig. 8.4, dove l'albero sotto $?- p(Z)$ è tagliato perché $p(Z)$ è sussunto da $p(X)$.

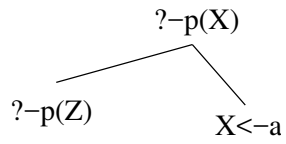


Figura 8.4: Prolog trova soltanto la risposta a

Se il goal corrente è sussunto da un antenato, la semplificazione è applicabile. Quindi ci si potrebbe aspettare che anche Linear Completion perda tutte le risposte eccetto $\{X \mapsto a\}$. Il risultato è invece molto diverso: il programma a riscrittura

$p(a) \to true$.

$p(Y)p(Z) \to p(Z)$.

con il quesito

$p(X) \to answer(X)$.,

genera le risposte $\{X \mapsto a\}$ e $\{X \mapsto X\}$ e poi si arresta come mostrato in Fig. 8.5.

Linear completion inferisce correttamente che $p(a)$ è vero e che $p(X)$ è vero per tutti gli X, dal momento che la seconda clausola nel programma dice che esiste un elemento Z tale che $p(Z)$ è vero, $p(Y)$ è vero per tutti gli Y. Operazionalmente, la ragione di questo diverso comportamento è che la semplificazione non è un mero meccansimo di riduzione dello spazio di ricerca, ma una regola di inferenza: se un goal viene semplificato, la computazione non si arresta, ma continua con il goal ridotto.

Il terzo tipo di meccansimo di controllo dei cicli in [7, 17] si basa sui *risultanti*: se G_0 è il

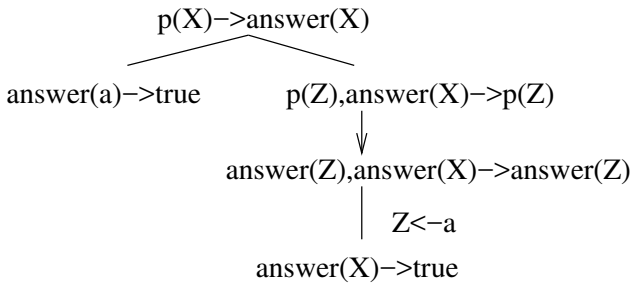


Figura 8.5: Linear completion genera $\{X \mapsto a\}$, $\{X \mapsto X\}$, e si arresta

quesito, G_i è il goal corrente e σ_i è la sostituzione risposta parziale computata lungo il cammino da G_0 a G_i, il risultante associato a G_i è $G_0\sigma_i$. I controlli dei cicli basati sui risultanti interrompono una derivazione se il goal corrente e il suo associato risultante avengono sussunti rispettivamente da un goal antenato e dal suo risultante. Questi controlli dei cicli sono chiamati *Equal Variant of Resultant (EVR)*, *Equal Instance of Resultant (EIR)*, *Subsumed as Variant of Resultant (SVR)* e *Subsumed as Instance of Resultant (SIR)*. I controlli dei cicli basati sui risultanti sono sound e completi per una classe di programmi molto ristretta [7, 17].

Controlli dei cicli basati su varianti piuttosto che su istanze possono essere troppo deboli: per esempio i controlli EVR e SVR non interromperebbero la derivazione infinita del nostro primo esempio su *append*, dal momento che i goals sono varianti, ma i risultanti non lo sono. I controlli EIR e SIR la interromperebbero come Linear Completion.

Il controllo SIR sembra essere quello i cui effetti sono più vicini a quelli della semplificazione in Linear Completion. Intuitivamente, questa similarità si spiega osservando che Linear Completion incorpora un qualche controllo sui risultanti perchè memorizza le sostituzioni risposta parziali nei letterali *answer*. Se il quesito ha la forma $G_0(\bar{x}) \rightarrow answer(\bar{x})$, il letterale $answer(\bar{x}\sigma_i)$ nel goal G_i è esattamente il risultante $G_0(\bar{x}\sigma_i)$. Questo non è il caso se il quesito ha la forma $G_0(c[\bar{x}]) \rightarrow answer(\bar{x})$ per qualche contesto c, ma un quesito siffatto può essere riformulato come $G_0(c[\bar{x}]) \rightarrow answer(c[\bar{x}])$, se dovesse risultare conveniente mantenere più informazione nel letterale *answer*. Se un letterale *answer* appare nel lato sinistro dell'antenato usato come semplificatore, la semplificazione controlla automaticamente anche i risultanti. Questo spiega intuitivamente perché nell'esempio seguente, preso da [17], il controllo dei cicli il cui comportamento è il più vicino a quello di Linear Completion è SIR. Dato il programma

$a(0).$

$a(Y) :- a(0), c(Y).$

$b(1).$

$c(Z) :- b(Z), a(W).$

e il quesito $?- a(X).$,

l'albero di esecuzione Prolog viene tagliato dai controlli dei cicli in [7, 17] come si vede in Fig. 8.6, dove ogni cornice delimita la porzione dell'albero che viene generata da Prolog aumentato con il

controllo indicato.

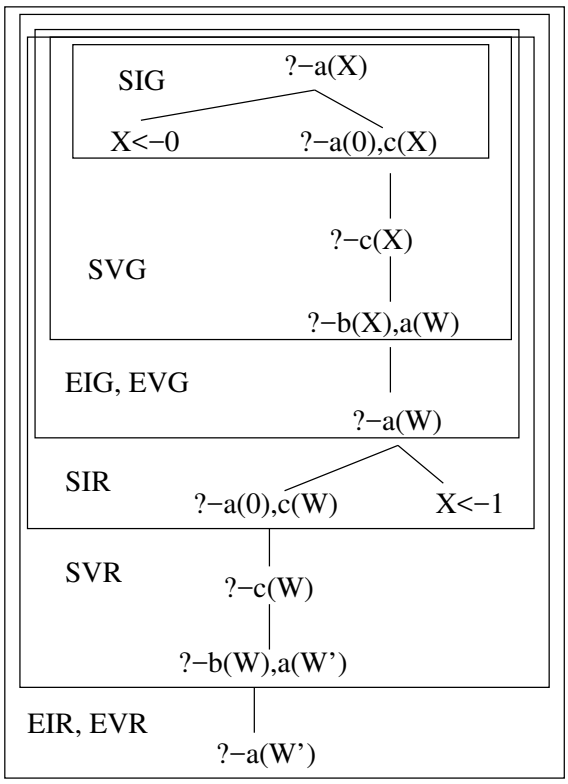


Figura 8.6: I controlli dei cicli tagliano l'esecuzione Prolog

In particolare, il controllo SIR arresta la derivazione dopo che le risposte $\{X \mapsto 0\}$ e $\{X \mapsto 1\}$ sono state ottenute perché il goal $?- a(0), c(W)$ e il suo risultante $a(1)$ sono rispettivamente una variante e un'istanza dell'antenato $?- a(0), c(X)$ e del suo risultante $a(X)$. Linear Completion genera le due risposte $\{X \mapsto 0\}$ e $\{X \mapsto 1\}$ e poi si ferma come illustrato in Fig. 8.7.

I controlli SIG/SVG e EIG/EVG arrestano la computazione troppo presto ed inibiscono la generazione della risposta $\{X \mapsto 1\}$. I controlli SVR, EIR ed EVR danno entrambe le soluzioni come Linear Completion e SIR, ma sono meno efficienti, poiché interrompono la computazione molto più tardi. E' notevole il fatto che Linear Completion ritaglia dall'albero Prolog esattamente quei cammini che conducono a una soluzione.

Riassumendo, ogni confronto tra Linear Completion e Prolog aumentato con questi controlli dei cicli è limitato dall'osservazione che i due approcci sono di natura molto diversa. L'approccio in [7, 17] consiste nel limitare il sistema inferenziale di Prolog aggiungendo controlli dei cicli. Il concetto di controllo dei cicli è puramente procedurale: il sistema di inferenza non è modificato, ma soltanto ristretto dall'aggiunta di un controllo esterno. Questo approccio sembra quindi un po' artificiale.

Nel nostro approccio la semplificazione è una conseguenza naturale del fatto che le unità di programma sono equazioni. La semplificazione non è un meccanismo per tagliare cammini nello spazio di ricerca aggiunto alle regole di inferenza, ma una regola di inferenza in sè, con la capacità

$a(0) \rightarrow true.$
$a(Y), a(0), c(Y) \rightarrow a(0), c(Y).$
$b(1) \rightarrow true.$
$c(Z) \rightarrow b(Z), a(W).$
$a(X) \rightarrow answer(X).$

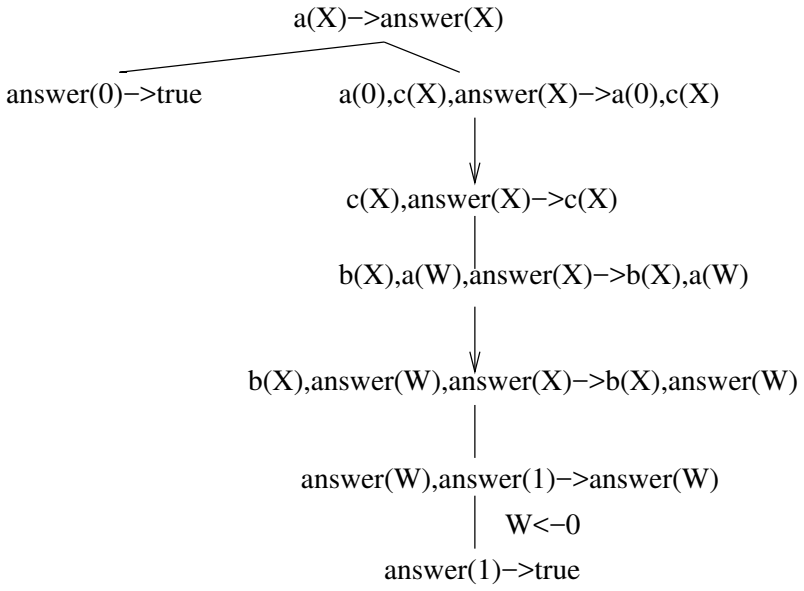


Figura 8.7: Linear completion genera $\{X \mapsto 0\}$, $\{X \mapsto 1\}$, e si ferma

di ridurre lo spazio di ricerca. Quindi, se si applica la semplificazione, la dervazione non viene interrotta, ma continua con il goal ridotto. Nel nostro primo esempio di *append*, Prolog con EIR o SIR non va in ciclo, grazie al meccanismo esterno di controllo, mentre Linear Completion non va in ciclo perché *append* è definito da equivalenze piuttosto che da implicazioni. Linear Completion si comporta come è desiderabile in tutti gli esempi proposti in [7, 17].

Inoltre, la semplificazione in Linear Completion usa uniformemente regole di programma, regole risposta e goals antenati come semplificatori, permettendo di ottimizzare le esecuzioni anche in assenza di derivazioni infinite, come mostrato dall'esempio *ancestor* nella Sezione 8.1.

Capitolo 9

Sommario e direzioni per future ricerche

Abbiamo sviluppato un nuovo approccio allo studio delle procedure di completamento di tipo Knuth-Bendix, dove tutti i concetti fondamentali sono definiti in modo uniforme in termini di *riduzione di prove* rispetto a un ordinamento ben fondato. Al fine di dare questo quadro di definizioni, abbiamo proposto una nozione di ordinamento su prove più generale di quelle note finora, in quanto permette di confrontare anche prove di diversi teoremi.

Una procedura di completamento si compone di un insieme di *regole di inferenza* e di un *piano di ricerca*. Abbiamo messo in evidenza la distinzione tra queste due componenti attraverso tutto il nostro lavoro. Questa distinzione è spesso dimenticata o sottostimata nella letteratura, dove la maggior parte delle strategie per la dimostrazione di teoremi viene presentata dando soltanto un insieme di regole di inferenza, quando invece è il piano di ricerca quello che volge un insieme di regole di inferenza in una procedura.

Le regole di inferenza sono o regole di *espansione* o regole di *contrazione*. Ogni passo di inferenza riduce alcune prove o elimina formule *ridondanti*. Abbiamo confrontato il nostro schema di espansione/contrazione con lo schema di deduzione/eliminazione in [16] e la nostra nozione di ridondanza con quelle in [74, 16]. Il nostro schema consente una trattazione più soddisfacente delle regole di contrazione e la nostra nozione di ridondanza sembra essere più vicina al significato intuitivo della ridondanza di formule in una derivazione.

Se le regole di inferenza sono *refutazionalmente complete* e il piano di ricerca è *fair*, una procedura di completamento è una procedura di semidecisione per dimostrazione di teoremi. La parte chiave di questo risultato è la nozione di *fairness*. La nostra definizione di fairness è la prima definizione di fairness per procedure di completamento che affronti il problema dal punto di vista della dimostrazione di teoremi. E' nuova sotto tre aspetti: è *orientata all'obbiettivo*, ovvero tiene il teorema da dimostrare in considerazione, è espressa esplicitamente come una proprietà del piano di ricerca ed è definita in termini di riduzione di prove, in modo tale che che regole di espansione e di contrazione sono trattate uniformemente.

Se il piano di ricerca di una procedura di completamento è *uniformemente fair*, il limite di una derivazione prodotta dalla procedura è *saturato*. Sotto condizioni aggiuntive, una presentazione

saturata è una *procedura di decisione* e quindi la procedura di completamento agisce come un *generatore di procedure di decisione*. Saturazione e semidecisione sono collegate nel nostro *teorema di Knuth-Bendix-Huet generalizzato*, la prima generalizzazione del ben noto risultato per logica equazionale. I precedenti lavori su presentazioni saturate in logica di Horn con uguaglianza non rappresentano generalizzazioni del teorema di Knuth-Bendix-Huet, perché non mettono in relazione il processo di saturazione con quello di semidecisione e considerano la saturazione come un processo di compilazione.

Abbiamo presentato alcune procedure di completamento in logica equazionale, basate sulla procedura Unfailing Knuth-Bendix, incluse la procedura AC-UKB con Leggi di Cancellazione, la S-strategy e la Inequality Ordered Saturation strategy. Queste estensioni di UKB non erano state presentate prima d'ora nel quadro di un approccio unitario al completamento.

Due ulteriori applicazioni del completamento sono la confutazione di congetture induttive e la programmazione logica. Per la prima, abbiamo mostrato che il cosiddetto metodo di *inductionless induction* è un processo di semidecisione. Per il secondo, abbiamo dato una semantica operazionale e denotazionale di programmi a riscrittura interpretati da *Linear Completion*. I programmi a riscrittura sembrano catturare il significato inteso dei predicati definiti in modo mutualmente esclusivo più accuratamente dei programmi Prolog. Inoltre, in presenza di tali predicati, la semplificazione permette di prevenire certe esecuzioni infinite che sono altrimenti inevitabili nel Prolog puro.

La ricerca riportata in questa tesi può essere continuata in parecchie direzioni. Abbiamo sviluppato questo approccio con l'intento di applicarlo alla progettazione di nuove procedure di completamento più efficienti. Questo goal conduce allo studio di nuove regole di contrazione più potenti e di piani di ricerca fair ed efficienti.

Siamo interessati ad efficienti procedure di completamento per dimostrazione di teoremi sia nel caso speciale della logica equazionale sia nel caso generale della logica del prim'ordine con uguaglianza. Quindi contiamo di ottenere una completa estensione del nostro approccio a procedure di completamento in logica di Horn e del prim'ordine con uguaglianza. Tale obbiettivo include lo studio di specifici ordinamenti su prove per la logica del prim'ordine con uguaglianza e ulteriore lavoro sulle nozioni di insieme saturato, derivazione lineare input riducente e fallimento in logica del prim'ordine con uguaglianza.

Infine, la nostra idea del completamento come trasformazione di problemi potrebbe essere estesa a una nozione di *completamento modulo un algoritmo di decisione*. Se un algoritmo di decisione è disponibile per una certa teoria, non è necessario spingere il completamento di un problema di dimostrazione $(S;\varphi)$ fino a uno stadio $(S_k;\varphi_k)$ dove $S_k \models \varphi_k$ è banale, come nel caso in cui φ_k è un'equazione $s \simeq s$. Il completamento termina a uno stadio k se $S_k \models \varphi_k$ è decidibile dall'algoritmo dato. Questo approccio apre la via allo studio del completamento in connessione con procedure di decisione per teorie speciali.

Bibliografia

[1] S.Anantharaman e J.Mzali, Unfailing Completion modulo a set of equations, Technical Report, LRI, Université de Paris Sud, 1989.

[2] S.Anantharaman, J.Hsiang e J.Mzali, SbReve2: A Term Rewriting Laboratory with (AC)-Unfailing Completion, in N.Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, NC, USA, Aprile 1989, Springer Verlag, Lecture Notes in Computer Science 355, 533–537, 1989.

[3] S.Anantharaman e J.Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *Journal of Automated Reasoning*, Vol. 6, No. 1, 76–109, 1990.

[4] S.Anantharaman e N.Andrianarivelo, Heuristical Criteria in Refutational Theorem Proving, in A.Miola (ed.), *Proceedings of the Symposium on the Design and Implementation of Systems for Symbolic Computation*, 184–193, Capri, Italy, Aprile 1990.

[5] S.Anantharaman, N.Andrianarivelo, M.P.Bonacina e J.Hsiang, SBR3: A Refutational Prover for Equational Theorems, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, Giugno 1990.

[6] K.R.Apt e M.H.Van Emden, Contributions to the Theory of Logic Programming, *Journal of the ACM*, Vol. 29, No. 3, 841–862, Luglio 1982.

[7] K.R.Apt, R.N.Bol e J.W.Klop, On the Safe Termination of PROLOG programs, in G.Levi, M.Martelli (eds.), *Proceedings of the Sixth International Conference on Logic Programming*, 353–368, MIT Press, Cambridge MA, 1989.

[8] L.Bachmair, N.Dershowitz e J.Hsiang, Orderings for Equational Proofs, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 346–357, Cambridge, MA, Giugno 1986.

[9] L.Bachmair e N.Dershowitz, Completion for rewriting modulo a congruence, in P.Lescanne (ed.), *Proceedings of the Second International Conference on Rewriting Techniques and Applications*, Bordeaux, France, Maggio 1987, Springer Verlag, Lecture Notes in Computer Science 256, 192–203, 1987.

[10] L.Bachmair e N.Dershowitz, Inference Rules for Rewrite-Based First-Order Theorem Proving, in *Proceedings of the Second Annual Symposium on Logic in Computer Science*, Ithaca, New York, Giugno 1987.

[11] L.Bachmair, Proofs Methods for Equational Theories, Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, IL.,1987.

[12] L.Bachmair e N.Dershowitz, Critical Pair Criteria for Completion, *Journal of Symbolic Computation*, Vol. 6, 1–18, 1988.

[13] L.Bachmair, Proof by consistency in equational theories, in *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, 228–233, Edinburgh, Scotland, Luglio 1988.

[14] L.Bachmair, N.Dershowitz e D.A.Plaisted, Completion without failure, in H.Ait-Kaci, M.Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. II: Rewriting Techniques, 1–30, Academic Press, New York, 1989.

[15] L.Bachmair e H.Ganzinger, On Restrictions of Ordered Paramodulation with Simplification, in *Proceedings of the Tenth International Conference on Automated Deduction*, Kaiserslautern, Germany, Luglio 1990.

[16] L.Bachmair e H.Ganzinger, Completion of First-Order Clauses with Equality by Strict Superposition, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, Giugno 1990.

[17] R.N.Bol, K.R.Apt e J.W.Klop, On the Power of Subsumption and Context Checks, in A.Miola (ed.), *Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*, 131–140, Capri, Italy, Aprile 1990.

[18] M.P.Bonacina e G.Sanna, KBlab: An Equational Theorem Prover for the Macintosh, in N.Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, NC, USA, Aprile 1989, Springer Verlag, Lecture Notes in Computer Science 355, 548–550, 1989.

[19] M.P.Bonacina e J.Hsiang, Completion procedures as Semidecision procedures, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, Giugno 1990.

[20] M.P.Bonacina e J.Hsiang, Operational and Denotational Semantics of Rewrite Programs, in S.Debray, M.Hermenegildo (eds.), *Proceedings of the North American Conference on Logic Programming*, Austin, TX, Octobre 1990, MIT Press, 449–464, 1990.

[21] M.P.Bonacina e J.Hsiang, On fairness of completion-based theorem proving strategies, to appear in R.V.Book (ed.), *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, Como, Italy, Aprile 1991.

[22] M.P.Bonacina e J.Hsiang, The Knuth-Bendix-Huet theorem and its extensions, in preparation.

[23] R.M.Burstall, D.B.MacQueen e D.T.Sannella, HOPE: an experimental applicative language, in *Conference Record of the 1980 LISP Conference*, 136–143, Stanford, California, 1980.

[24] C.L.Chang e R.C.Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[25] S.K.Debray e D.S.Warren, Functional Computations in Logic Programs, *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 3, 451–481, Luglio 1989.

[26] N.Dershowitz e Z.Manna, Proving termination with multisets orderings, *Communications of the ACM*, Vol. 22, No. 8, 465–476, Agosto 1979.

[27] N.Dershowitz, Orderings for term-rewriting systems, *Theoretical Computer Science*, Vol. 17, No. 3, 279–301, 1982.

[28] N.Dershowitz, J.Hsiang, N.A.Josephson e D.A.Plaisted, Associative-commutative rewriting, in A.Bundy ed., *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 940–944, Karlsrue, Germany, 1983.

[29] N.Dershowitz e N.A.Josephson, Logic Programming by Completion, in *Proceedings of the Second International Conference on Logic Programming*, 313–320, Uppsala, Sweden, 1984.

[30] N.Dershowitz, Computing with Rewrite Systems, *Information and Control*, Vol. 65, 122–157, 1985.

[31] N.Dershowitz e D.A.Plaisted, Logic Programming Cum Applicative Programming, in *Proceedings of the IEEE Symposium on Logic Programming*, 54–66, Boston, MA, 1985.

[32] N.Dershowitz, Termination of Rewriting, *Journal of Symbolic Computation*, Vol. 3, No. 1 & 2, 69–116, Febbraio/Aprile 1987.

[33] N.Dershowitz, Completion and its Applications, in *Proceedings of Conference on Resolution of Equations in Algebraic Structures*, Lakeway, Texas, Maggio 1987.

[34] N.Dershowitz e J.-P.Jouannaud, Rewrite Systems, Chapter 15 of Volume B of *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[35] N.Dershowitz e J.-P.Jouannaud, Notations for Rewriting, Rapport de Recherche 478, LRI, Université de Paris Sud, Gennaio 1990.

[36] N.Dershowitz, A Maximal-Literal Unit Strategy for Horn Clauses, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, Giugno 1990.

[37] F.Fages, Associative-commutative unification, in R.Shostak (ed.), *Proceedings of the Seventh International Conference on Automated Deduction*, Napa Valley, CA, USA, 1984, Springer Verlag, Lecture Notes in Computer Science 170, 1984.

[38] L.Fribourg, A Strong Restriction to the Inductive Completion Procedure, in *Proceedins of the Thirteenth International Conference on Automata Languages and Programming*, Rennes, France, Luglio 1986, Springer Verlag, Lecture Notes in Computer Science 226, 1986.

[39] K.Futatsugi, J.A.Goguen, J.P.Jouannaud e J.Meseguer, Principles of OBJ2, in *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages*, New Orleans, LA, 1985.

[40] J.A.Goguen, How to prove algebraic inductive hypotheses without induction, in W.Bibel and R.Kowalski (eds.), *Proceedings of the Fifth International Conference on Automated Deduction*, 356–373, Les Arcs, France, 1980, Springer Verlag, Lecture Notes in Computer Science 87, 1980.

[41] J.A.Goguen e J.Meseguer, Equality, types, modules and (why not?) generics for logic programming, *Journal of Logic Programming*, Vol. 1, No. 2, 179–210, 1984.

[42] C.C.Green, The Application of Theorem-proving to Question-answering, Ph.D. Thesis, Dept. of Computer Science, Stanford University, Stanford, California, 1969.

[43] C.M.Hoffmann e M.J.O'Donnell, Programming with Equations, *ACM Transactions on Programming Languages and Systems*, Vol. 4. No. 1, 83–112, Gennaio 1982.

[44] J.Hsiang e N.Dershowitz, Rewrite Methods for Clausal and Nonclausal Theorem Proving, in *Proceedings of the Tenth International Conference on Automata, Languages and Programming*, Barcelona, Spain, Luglio 1983, Springer Verlag, Lecture Notes in Computer Science 154, 1983.

[45] J.Hsiang, Refutational Theorem Proving Using Term Rewriting Systems, *Artificial Intelligence*, Vol. 25, 255–300, 1985.

[46] J.Hsiang e M.Rusinowitch, A New Method for Establishing Refutational Completeness in Theorem Proving, in J.Siekmann (ed.), *Proceedings of the Eighth Conference on Automated Deduction*, Oxford, England, Luglio 1986, Springer Verlag, Lecture Notes in Computer Science 230, 141–152, 1986.

[47] J.Hsiang, Rewrite Method for Theorem Proving in First Order Theories with Equality, *Journal of Symbolic Computation*, Vol. 3, 133–151, 1987.

[48] J.Hsiang e M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, Karlsruhe, Germany, Luglio 1987, Springer Verlag, Lecture Notes in Computer Science 267, 54–71, 1987.

[49] J.Hsiang, M.Rusinowitch e K.Sakai, Complete Inference Rules for the Cancellation Laws, in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milano, Italy, Agosto 1987, 990–992, 1987.

[50] J.Hsiang e M.Rusinowitch, Proving Refutational Completeness of Theorem Proving Strategies: the Transfinite Semantic Tree Method, to appear in *Journal of the ACM*, 1990.

[51] G.Huet, Confluent reductions: abstract properties and applications to term rewriting systems, *Journal of the ACM*, Vol. 27, 797–821, 1980.

[52] G.Huet, A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm, *Journal of Computer and System Sciences*, Vol. 23, 11–21, 1981.

[53] G.Huet e J.M.Hullot, Proofs by Induction in Equational Theories with Constructors, Journal of Computer and System Sciences, Vol. 25, 239–266, 1982.

[54] J.-P.Jouannaud e E.Kounalis, Proofs by induction in equational theories without constructors, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 358–366, Cambridge, MA, Giugno 1986.

[55] J.-P.Jouannaud e C.Kirchner, Completion of a set of rules modulo a set of equations, *SIAM Journal of Computing*, Vol. 15, 1155–1194, Novembre 1986.

[56] J.-P.Jouannaud e E.Kounalis, Automatic proofs by induction in equational theories without constructors, *Information and Computation*, 1989.

[57] J.-P.Jouannaud e C.Kirchner, Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification, Rapport de Recherche, LRI, Université de Paris Sud, Novembre 1989.

[58] S.Kamin e J.-J.Lévy, Two generalizations of the recursive path ordering, Unpublished note, Department of Computer Science, University of Illinois, Urbana, Illinois, Febbraio 1980.

[59] D.Kapur e P.Narendran, An equational approach to theorem proving in first order predicate calculus, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1146–1153, Los Angeles, CA, Agosto 1985.

[60] D.Kapur e D.R.Musser, Proof by consistency, *Artificial Intelligence*, Vol. 31, No. 2, 125–157, Febbraio 1987.

[61] D.Kapur, P.Narendran e H.Zhang, Proof by induction using test sets, in J.Siekmann (ed.), *Proceedings of the Eighth Conference on Automated Deduction*, Oxford, England, Luglio 1986, Springer Verlag, Lecture Notes in Computer Science 230, 99–117, 1986.

[62] D.E.Knuth e P.B.Bendix, Simple Word Problems in Universal Algebras, in J.Leech (ed.), *Proceedings of the Conference on Computational Problems in Abstract Algebras*, Oxford, England, 1967, Pergamon Press, Oxford, 263–298, 1970.

[63] E.Kounalis e M.Rusinowitch, On Word Problems in Horn Theories, in E.Lusk, R.Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction*, 527–537, Argonne, Illinois, Maggio 1988, Springer Verlag, Lecture Notes in Computer Science 310, 1988.

[64] R.A.Kowalski e M.H.Van Emden, Predicate Logic as a Programming Language, *Journal of the ACM*, Vol. 4, No. 23, 1976.

[65] D.S.Lankford, Canonical inference, Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX, Maggio 1975.

[66] D.S.Lankford, A simple explanation of inductionless induction, Technical report MTP-14, Mathematics Department, Louisiana Technical University, Ruston, LA, 1981.

[67] J.W.Lloyd, *Foundations of Logic Programming*, second edition, Springer Verlag, Berlin, 1987.

[68] D.Musser, On proving inductive properties of abstract data types, in *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, 154–162, Las Vegas, Nevada, 1980.

[69] A.Ohsuga e K.Sakai, Refutational Theorem Proving for Equational Systems Based on Term Rewriting, Technical Report COMP86–40, ICOT, 1986.

[70] G.E.Peterson e M.E.Stickel, Complete sets of reductions for some equational theories, *Journal of the ACM*, Vol. 28, No. 2, 233–264, 1981.

[71] G.E.Peterson, A Technique for Establishing Completeness Results in Theorem proving with Equality, *SIAM Journal of Computing*, Vol. 12, No. 1, 82–100, 1983.

[72] D.A.Plaisted, Semantic confluence tests and completion methods, *Information and Control*, Vol. 65, 182–215, 1985.

[73] P.Rety, C.Kirchner, H.Kirchner e P.Lescanne, NARROWER: a new algorithm for unification and its application to logic programming, in J.P.Jouannaud ed., *Proceedings of the First International Conference on Rewrite Techniques and Applications*, Dijon, France, Maggio 1985.

[74] M.Rusinowitch, Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure as a complete set of inference rules, Thèse d'Etat, Université de Nancy, 1987.

[75] J.R.Slagle, Automated theorem proving with simplifiers, commutativity and associativity, *Journal of the ACM*, Vol. 21, 622–642, 1974.

[76] M.E.Stickel, Unification Algorithms for Artificial Intelligence Languages, Ph.D. thesis, Carnegie Mellon University, 1976.

[77] M.E.Stickel, A unification algorithm for associative-commutative functions, *Journal of the ACM*, Vol. 28, No. 3, 423–434, 1981.

[78] H.Zhang e D.Kapur, First Order Theorem Proving Using Conditional Rewrite RUles, in E.Lusk, R.Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction*, 1–20, Argonne, Illinois, Maggio 1988, Springer Verlag, Lecture Notes in Computer Science 310, 1988.