UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica

L'astrazione in un dimostratore automatico interattivo: definizione e realizzazione.

Relatore: Prof. Enrico Pagello
Correlatore: Ing. Fausto Giunchiglia

Laureando:
Roberto Sebastiani

Anno Accademico 1990 – 1991

L'imperatore - così si dice - ha inviato a te, proprio a te individuo singolo, miserrimo tra i sudditi, a te che davanti al sole imperiale sei fuggito come futile ombra nella più remota lontananza, un messaggio dal suo letto di morte. Accanto al letto ha fatto inginocchiare un messo e gli ha bisbigliato all'orecchio il messaggio; e ci teneva tanto, che se l'è fatto ripetere a sua volta all'orecchio; poi, con un cenno del capo, ne ha confermato l'esattezza. E al cospetto di tutto il pubblico presente alla sua morte - i muri che fanno ostacolo sono demoliti, e sulle gradinate che si elevano alte e maestose fanno circolo i dignitari dell'impero - davanti a costoro ha licenziato il messo. Questi si è posto subito in viaggio: uomo gagliardo, instancabile, muovendo ora questo ora quel braccio, si fa strada tra la folla; se incontra resistenza, addita sul suo petto il segno del sole; l'avanzare gli è facile come a nessun altro. Ma la folla è smisurata, le sue dimore non hanno fine. Se egli trovasse campo libero, come volerebbe! e ben presto udresti alla tua porta, imperioso, il rimbombo dei suoi pugni. Invece si affatica invano; sta ancora aprendosi il cammino attraverso le stanze del palazzo più interno; non riuscirà mai a superarle, e se anche ci riuscisse, sarebbe al punto di prima: dovrebbe battersi da leone per discendere le scale; e se anche questo gli riuscisse, sarebbe ancora nulla: gli rimarrebbe da attraversare i cortili; e dopo i cortili la cerchia del secondo palazzo, e di nuovo scale e cortili; e poi un altro palazzo; e così via per millenni; e quando finalmente sbucasse dall'ultimissima porta - ma ciò non accadrà mai e poi mai -, si troverà davanti la città imperiale, il centro del mondo, colma fino all'orlo di tutta la sua feccia: nessuno può venirne a capo, anche se sia latore del messaggio di un morto.

Ma tu siedi alla finestra e immagini che giunga a te, quando scende la sera.

[F.Kafka - "*Un messaggio dell'imperatore*"]

Ringraziamenti

Al termine del lavoro di tesi, ed anche di tutti gli anni di università, voglio ringraziare:

- **Enrico Pagello**. Per avermi seguito e sostenuto nello svolgimento della tesi; ed anche per aver suscitato in me, ancora con il corso di A.L.F., l'interesse per la logica ed i vari aspetti dell'informatica teorica.
- L'**I.R.S.T.**, inteso come istituzione e come insieme di persone. Per avermi messo a disposizione una gran quantità di risorse, tecniche ed umane, e per avermi permesso in questo modo di avventurarmi nel mondo della ricerca.
- **Fausto Giunchiglia**. Per l'entusiasmo e la valanga di stimoli che mi ha saputo comunicare. Per avermi trasmesso una certa mentalità nell'affrontare i vari aspetti del lavoro. E per avere affrontato a sua volta certi aspetti del mio carattere.
- **Toby Walsh**. Per l'aiuto, le spiegazioni e le discussioni sull'astrazione, la logica matematica e lo sci. E per le implicite lezioni d'inglese in esse contenute.
- **Paolo Traverso**, **Alessandro Cimatti** e **Paolo Pecchiari**. Per le spiegazioni, le discussioni e l'aiuto durante tutto l'arco di questo periodo. E per aver sempre risposto con pazienza ad ogni mia frase del tipo: "*Scusa, ti posso rubare dieci minuti?*"...
- **Luciano Serafini**. Non mi posso limitare a ringraziarlo per le sue spiegazioni di logica matematica, nè dire soltanto che senza di lui non avrei mai potuto scrivere la tesi in LaTeX. Luciano è sempre stato un sostegno psicologico ed un punto di riferimento. La sua disponibilità e i suoi consigli sono sempre andati molto oltre agli aspetti puramente scientifici e tecnici di questo mio anno di lavoro all'I.R.S.T.
- **Carolyn Talcott**, per l'interesse dimostrato e per gli utilissimi commenti.
- **Carola Dori** ed **Ileana Collini**. Per la disponibilità, la pazienza e la capacità di sopportazione dimostrate nei confronti miei e delle mie richieste. E per tutto il tempo passato insieme.
- **Il gruppo m.r.g.** e tutti i suoi membri, vecchi, recenti e saltuari: Sandro, Alessandro & Alessandro, Enrico, Paolo, Lorenzo, David .... Per l'ambiente creato e per tutte le cose condivise in questo periodo, dai seminari sulle logiche non monotone ai caffè al bar.
- **I miei amici**, di Padova e di Trento. Anna, Aldo, Alessia, Giacomo, Roberta, Paola, Max, Laura.... Per tutto.
- **Mia madre**, "*sine qua non...*".

Sommario

Questa tesi è stata sviluppata all'interno del Gruppo di Ragionamento Meccanizzato dell'Istituto per la Ricerca Scientifica e Tecnologica (I.R.S.T.) di Trento. Essa si inserisce nel progetto `GETFOL`, volto a sviluppare un sistema interattivo di deduzione automatica, per la logica dei predicati del primo ordine, che possa essere utilizzato in varie aree di ricerca e di sviluppo, quali Intelligenza Artificiale, Theorem Proving, Logica Matematica e Teoria della Computabilità.

Scopo principale della tesi è quello di sviluppare una parte di un sottosistema di deduzione automatica basato su meccanismi di **astrazione** (in inglese "abstract theorem prover"), ispirandosi alla *Teoria Formale dell'Astrazione* elaborata da F.Giunchiglia e T.Walsh. In accordo con tale teoria, il sistema, una volta completato, dovrebbe consentire di ridurre significativamente la ridondanza all'interno di un processo di deduzione.

Come premessa a tale lavoro, si è resa necessaria l'analisi teorica di una serie di problematiche, legate all'uso dell'astrazione con sistemi formali "*multisorte*" e alla nozione di *funzione tra linguaggi del primo ordine*. La prima ha permesso di estendere la definizione formale di astrazione a sistemi "multisorte", creando i presupposti teorici per una futura espansione del lavoro. La seconda ha portato alla definizione di un particolare *linguaggio* per definire le astrazioni, e alla concezione di un possibile *interprete* per tale linguaggio.

Questi concetti sono poi stati applicati nella realizzazione di una serie di nuovi comandi di `GETFOL`, che permettono all'utente di *dichiarare* un'astrazione e di *applicarla* a vari "oggetti", tra cui semplici simboli logici, termini, formule, assiomi ed intere teorie.

Il codice scritto viene allegato come appendice alla tesi.

Organizzazione della tesi

La tesi è così strutturata:

- **Cap. 1: Introduzione**

  Vengono qui introdotte alcune tematiche utili alla comprensione globale della tesi. In particolare, vengono presentate una rapida panoramica sulla deduzione automatica, una breve descrizione del sistema `GETFOL` ed un'introduzione informale al concetto di astrazione. Vengono poi illustrati brevemente lo scopo del lavoro ed i risultati raggiunti.

- **Cap. 2: La teoria dell'astrazione**

  Questo capitolo presenta un riassunto della *Teoria Formale dell'Astrazione* elaborata da F. Giunchiglia e T. Walsh. Alcuni nuovi esempi sono stati introdotti per una migliore comprensione della teoria. Inoltre sono stati inserite alcune note e commenti personali che, senza avere la pretesa di aggiungere nulla di nuovo alla teoria stessa, cercano di chiarire il significato intuitivo di certi aspetti della trattazione.

- **Cap. 3: Estensione dell'astrazione a "sorted logics"**

  Scopo di questo capitolo è quello di estendere la *Teoria Formale dell' Astrazione* a sistemi logici "multisorte", introducendo una definizione di astrazione estesa a tali sistemi e mostrando come tale estensione mantenga inalterate certe fondamentali proprietà delle astrazioni "ordinarie".

- **Cap. 4: Definizione di una funzione tra linguaggi**

  In primo luogo, vengono qui descritte le proprietà richieste alla definizione di una generica funzione tra linguaggi, f_Λ. Successivamente, viene introdotto un formalismo per descrivere tali funzioni, il quale soddisfa le proprietà sopra menzionate. Queste nozioni costituiscono i fondamenti teorici per la realizzazione dell'astrazione in `GETFOL`.

- **Cap. 5: I comandi per l'astrazione in `GETFOL`**

  Con il lavoro di questa tesi sono stati definiti e realizzati i **comandi** per permettere all'utente di *definire* un'astrazione e di *applicarla* agli oggetti desiderati (espressioni logiche, assiomi, semplici simboli dell'alfabeto ...), generando i corrispondenti oggetti nello spazio astratto.

  In questo capitolo vengono illustrate le *modalità d'uso* di tali comandi, corredando la descrizione con alcuni esempi.

- **Cap. 6: La descrizione del codice**

  In questo capitolo viene brevemente descritto il codice che realizza i comandi per l'astrazione. In particolare, viene data una rapida descrizione della struttura dati "abstraction", e vengono presentate con sufficiente dettaglio le routine per l'astrazione delle espressioni logiche.

- **Cap. 7: Conclusioni**

  In primo luogo vengono riassunti i principali risultati del lavoro, presentando anche i limiti a cui essi sono soggetti. Viene poi tracciato un breve quadro del lavoro futuro necessario al completamento del modulo "*abstract theorem prover*". Infine, vengono illustrate una serie di prospettive di utilizzo per il sistema costruito.

Il codice scritto viene allegato come appendice della tesi.

Notazione e convenzioni

Per tutto l'arco della tesi, se non altrimenti specificato, useremo la seguente notazione:

Insiemi e funzioni	
$\{x \vert P(x)\}$	l'insieme degli oggetti che soddisfano la proprietà P
$\in$	appartenenza ad un insieme
$\cup, \cap$	unione ed intersezione tra insiemi
$\subseteq$	sottinsieme
$\times$	prodotto cartesiano, $A^2 = A \times A$
$f : A \longmapsto B$	una funzione f da A in B
$\circ$	composizione di funzioni (convenzione italiana)
$\langle A, ... \rangle$	una lista ordinata
Logica matematica	
$\varphi, \alpha, \beta, ...$	formule ben formate
$P(.), Q(.), ...$	predicati
$\top, \bot$	vero e falso
$f(.), g(.), ...$	funzioni
$a, b, ...$	costanti individuali
$x, y, ...$	variabili individuali
A_t^x	sostituzione della variabile x con il termine t nella formula A
$\underline{x}, ...$	n-uple di oggetti x_i
$\neg, \vee, \wedge, \supset, \leftrightarrow$	connettivi logici
$\forall, \exists$	quantificatori
$\Sigma, \Sigma_1, ...$	sistemi formali assiomatici, $\Sigma = \langle \Lambda, \Omega, \Delta \rangle$
$\Lambda, \Lambda_1, ...$	linguaggi logici del primo ordine
$\Omega, \Omega_1, ...$	insiemi di assiomi
$\Delta, \Delta_1, ...$	insiemi di regole di inferenza ("deductive machinery")
$\langle \mathcal{A}, \mathcal{T}, \mathcal{W} \rangle$	un linguaggio logico, formato da un alfabeto, dall'insieme dei termini ben formati e dall'insieme delle formule ben formate
$\mathrm{TH}(\Sigma)$	l'insieme delle formule dimostrabili in Σ (teoremi)
$\mathrm{NTH}(\Sigma)$	l'insieme delle negazioni di teoremi in Σ
$\vdash_\Sigma$	dimostrabile in Σ
$\Pi, \Pi_1, \Gamma, \Gamma_1, ...$	alberi di formule e dimostrazioni
$\subseteq$	relazione di sussunzione tra alberi
Astrazioni	
$f : \Sigma_1 \Rightarrow \Sigma_2$	un'astrazione
f_Λ	una funzione tra linguaggi del primo ordine
$\circ$	composizione di astrazioni
$\sqsubseteq, \sqsupseteq$	ordinamento parziale sulle astrazioni

"sorted logics"	
$\langle \mathcal{L}, \mathcal{S}, \mathcal{D} \rangle$	Sistema formale "Sorted Natural Deduction" (SND)
$\mathcal{L}$	linguaggio SND
$\mathcal{S}$	"`SORT INFORMATION`"
$\mathcal{D}$	insieme delle regole d'inferenza SND
$\mathcal{P}$	l'insieme dei simboli di sorte
$s, s_1, \ldots$	simboli di sorte
$sort_c \ldots sort_f$	funzioni $simbolo \longmapsto sorte$
$\succeq$	la relazione di "maggior generalità"
f_s	una funzione tra sorti
$\mathcal{P}\forall I \ldots \mathcal{P}\exists E$	regole d'inferenza SND dei quantificatori.

In generale, useremo i caratteri `ttype` per includere esempi di esecuzioni in `GETFOL` e parti di codice, il font *italic* per le definizioni, i teoremi, le formule matematiche e per evidenziare parole o frasi, il **neretto** per evidenziare con ulteriore forza.

Per alleggerire l'uso di certi vocaboli fortemente ricorrenti, sono stati utilizzati alcuni sinonimi, come "*contesto*" e "*spazio*" per "*sistema formale*", "*mappa*" per "*astrazione*". In particolare, si è usato scrivere `astrazione` (in caratteri `ttype`) per rappresentare il significato *intuitivo* della parola, in alternativa alla definizione formale.

Tutte queste convenzioni sono coerenti con quelle utilizzate negli articoli relativi alla *Teoria Formale dell'Astrazione* (ad es. [Giu]).

In alcune situazioni sono state usate parole ed espressioni della lingua inglese. Questo si è reso necessario, in alcuni casi, per evitare ambiguità introdotte dall'uso dei corrispondenti vocaboli italiani (es. "*map*" e "*rule*"), in altri (es. "*Sorted Natural Deduction*") per evitare di introdurre neologismi.

Indice

"Leibnitz, Boole and Gödel worked with logic.
I work with logic.
I am Leibnitz, Boole and Gödel."

Capitolo 1

Introduzione

Questa tesi è stata sviluppata all'interno del Gruppo di Ragionamento Meccanizzato (M.R.G.) dell'Istituto per la Ricerca Scientifica e Tecnologica (I.R.S.T.) di Trento, sotto la guida dell'ing. F.Giunchiglia. Essa si inserisce all'interno del progetto `GETFOL` [GT91, GAC+91], volto a sviluppare un sistema interattivo di deduzione automatica che possa costituire un utile strumento in varie aree di ricerca e di lavoro, quali Intelligenza Artificiale, Theorem Proving, Logica Matematica e Teoria della Computabilità.

In particolare, scopo della tesi è quello di creare (la prima parte di) un sottosistema di deduzione automatica basato su meccanismi di astrazione, *"abstract theorem prover"*. Tale sistema viene realizzato mediante una serie di nuovi comandi di `GETFOL`.

In questo primo capitolo verranno introdotte alcune tematiche utili alla comprensione globale della tesi. In particolare, verranno presentate una rapida panoramica sulla deduzione automatica, una breve descrizione del sistema `GETFOL` e infine un'introduzione informale al concetto di astrazione. Vengono poi illustrati brevemente lo scopo del lavoro ed i risultati raggiunti.

1.1 La deduzione automatica

Con il termine **deduzione** si intende il processo che permette di arrivare ad una certa conclusione, partendo da opportune ipotesi ed utilizzando determinate regole di inferenza. L'aggettivo "**automatica**" si riferisce alla possibilità di farlo in modo "meccanico", tipicamente tramite un elaboratore elettronico.

Nel campo della deduzione automatica, ipotesi, conclusioni e risultati intermedi vengono generalmente descritti tramite il linguaggio della logica dei predicati del primo ordine. Per esempio, il celebre sillogismo aristotelico:

"*Tutti gli uomini sono mortali, Socrate è un uomo, quindi Socrate è mortale*"
viene rappresentato dalla deduzione:

$$\frac{Uomo(Socrate) \quad \forall x.(Uomo(x) \supset Mortale(x))}{Mortale(Socrate)}.$$

L'idea di realizzare un programma in grado di dimostrare teoremi ha sempre incontrato grossi ostacoli, sia dal punto di vista teorico [1] che da quello computazionale. Il primo passo significativo si ebbe con l'invenzione di *Resolution* [Rob65], una potente procedura di dimostrazione fondata su un meccanismo di refutazione. Con l'avvento di tale sistema divenne possibile realizzare i primi meccanismi automatici di deduzione ("*theorem prover*"). Ad esempio il *Prolog* può essere considerato come un'evoluzione di questo metodo.

Una grossa limitazione dell'algoritmo di risoluzione sta nel fatto che le prove ottenute con questo metodo non hanno un'interpretazione intuitiva. All'interno di tale sistema, infatti, una formula viene convertita in un insieme di disgiunzioni di formule atomiche e/o di loro negazioni. In questo modo viene a perdersi l'informazione relativa alla costruzione della formula stessa, che potrebbe guidare il dimostratore verso la costruzione di una prova. Per esempio, se consideriamo le due formule equivalenti $A \supset B$ e $\neg A \vee B$, notiamo come la prima "suggerisca" di dimostrare A per poi ricavare B, mentre la seconda suggerisce piuttosto una dimostrazione per casi.

La grossa limitazione di tutti i theorem prover completamente automatici, e quindi di Resolution, sta nell'intrinseca *intrattabilità* computazionale insita nella nozione stessa di deduzione. Il problema è rappresentato dalla crescita esponenziale dell'albero di ricerca, che si manifesta, tra l'altro, in una gran quantità di passi inutili eseguiti. In questo senso, possiamo dire che la deduzione completamente automatica di un teorema è in generale un processo che causa una gran quantità di lavoro ridondante.

Un filone di ricerca diverso, storicamente successivo, portò alla realizzazione di logiche "naturali" [Gen69, Pra65], in cui il processo di deduzione viene ad essere più "simile" a quello usato da un matematico quando dimostra un teorema. L'avvento di queste logiche ha causato, assieme ad altri fattori, lo sviluppo di sistemi di deduzione "interattivi", in cui l'utente stabilisce la strategia di dimostrazione ed utilizza un dispositivo automatico per creare singoli "pezzi" di deduzione, più o meno elaborati a seconda del livello di sofisticazione del dispositivo. In questo modo, la guida "intelligente" dell'utente permette di evitare al sistema di ripetere più volte uno stesso passaggio e/o di perdere tempo nella dimostrazioni di formule inutili.

Il sistema `GETFOL` [GT91, GAC$^+$91] si inserisce in questo filone.

All'interno di quest'ultima ottica si inserisce anche l'idea di **astrazione**. Essa si propone come una tecnica per ridurre significativamente la ridondanza contenuta nel processo di deduzione. Una determinata formulazione di un teorema può infatti venire "astratta" in una rappresentazione più semplice, eliminando una serie di particolari ritenuti inutili al fine della costruzione della prova. La dimostrazione "più semplice" così ottenuta viene utilizzata in un secondo momento per costruire la prova vera e propria, reinserendo i "dettagli" che si erano eliminati nella prima fase.

[1] I risultati negativi di Church e Turing [Kle52] dimostrano l'impossibilità di costruire una macchina (od un programma) in grado di decidere *in un tempo finito* la dimostrabilità di una generica formula della logica del primo ordine. Tali risultati costituiscono un limite invalicabile alla costruzione di dimostratori automatici di teoremi.

Quest'idea si inserisce all'interno dell'approccio "interattivo" al theorem proving. Infatti, la *scelta* dell'astrazione da utilizzare costituisce una guida al sistema da parte dell'utente.

1.2 Il sistema GETFOL

`GETFOL` [GT91, GAC$^+$91] è un sistema interattivo di deduzione automatica sviluppato dal 1985 al 1991. Esso costituisce una re-implementazione del sistema `FOL` [Wey77, Wey80][2], sviluppato all'Università di Stanford dal 1973 al 1985 sotto la direzione di R.Weyhrauch.

Scopo dichiarato dall'autore per lo sviluppo di tale sistema è quello di fornire un utile strumento per ricercatori ed utenti di varie aree della matematica e dell'informatica, quali Intelligenza Artificiale, Theorem Proving, Logica Matematica e Teoria della Computabilità.

`GETFOL` utilizza un formalismo logico simile a "Natural Deduction" [Pra65]. Questo sistema di base è ulteriormente arricchito, fornendo all'utente una serie di potenti strumenti aggiuntivi, tra i quali ricordiamo alcuni decisori per certe sottoclassi della logica dei predicati ("TAUT" e "MONADIC"), la possibilità di associare dei modelli computazionali ad una teoria [Pec90], la possibilità di ragionare in termine di Metateorie.

Cerchiamo di analizzare un possibile utilizzo del sistema tramite un esempio.

Esempio 1
Supponiamo di voler dimostrare il seguente teorema:

"*Se tutti gli oggetti verificano la proprietà P, allora non esiste alcun oggetto che non la verifichi*".
Secondo la logica del primo ordine, esso viene descritto dalla formula:

$$\forall x.P(x) \supset \neg\exists x.\neg P(x).$$

Utilizzando *Natural Deduction* [Pra65], si può costruire la dimostrazione:

$$\dfrac{\dfrac{\dfrac{[\exists x.\neg P(x)] \qquad \dfrac{[\neg P(par)] \qquad \dfrac{[\forall x.P(x)]}{P(par)}\,(\forall E)}{\bot}\,(\bot\ I)}{\bot}\,(\exists E)}{\neg\exists x.\neg P(x)}\,(\bot\ E)}{\forall x.P(x) \supset \neg\exists x.\neg P(x)}\,(\supset I) \quad . \tag{1.1}$$

Utilizziamo ora il sistema `GETFOL` per costruire la stessa prova.

In primo luogo, dobbiamo generare il *linguaggio* con cui descrivere la nostra deduzione. Dobbiamo quindi *dichiarare* che il predicato unario P, la variabile individuale x e il parametro individuale *par* appartengono al linguaggio corrente:

[2] "`FOL`" è un acronimo per First Order Logic, "`GET`" per Genova, Edimburgh, Trento, le tre città in cui tale sistema è stato sviluppato.

```
GETFOL:: DECLARE PREDCONST P 1;
P has been declared to be a Predconst

GETFOL:: DECLARE INDVAR x;
x has been declared to be an Indvar

GETFOL:: DECLARE INDPAR par;
par has been declared to be an Indpar
```

Possiamo ora iniziare la dimostrazione vera e propria, seguendo lo schema della prova 1.1. Introduciamo l'assunzione[3]:

```
GETFOL:: ASSUME forall x.P(x);
1    forall x.P(x)    (1)
```

Istanziamo la variabile x al parametro par, applicando la regola di "forall elimination", $\forall E$:

```
GETFOL:: ALLE 1 par;
2    P(par)   (1)
```

Introduciamo un'ulteriore assunzione:

```
GETFOL:: ASSUME exists x.not P(x);
3    exists x. not P(x)   (3)
```

Eliminiamo ora la quantificazione esistenziale dalla formula n.3, utilizzando la regola di "exists elimination", $\exists E$ [4]:

```
GETFOL:: EXISTE 3;
4    not P(par)    (4)
```

Dalle relazioni 2 e 4 siamo in grado di ricavare il falso:

```
GETFOL:: FALSEI 2 4;
5    FALSE    (1 3)
```

Per assurdo, possiamo quindi dedurre la negazione dell'assunzione n.3, applicando la regola "not introduction", $\neg I$, alle formule n. 5 e 3:

```
GETFOL:: NOTI 5 3;
6    not exists x. not A(x)    (1)
```

[3]Secondo le convenzioni del sistema GETFOL, il numero sulla sinistra rappresenta il *nome* della formula ottenuta, mentre i numeri tra parentesi, sulla destra, rappresentano le assunzioni da cui la formula dipende. In questo caso, la formula n. 1 dipende solo da sè stessa, in essa quanto è un'assunzione.

[4]In GETFOL "$\exists E$" è realizzata in modo diverso rispetto alla corrispondenta regola in Natural Deduction. Si rimanda al manuale utente [GT91] per ulteriori spiegazioni.

A questo punto l'ultimo risultato dipende solo dall'assunzione n.1. Possiamo banalmente concludere, applicando una "imply introduction", $\supset I$:

```
GETFOL:: IMPI 1 6;
7    forall x.P(x) imp not exists not P(x).
```

Abbiamo quindi costruito una prova del teorema richiesto.

Abbiamo accennato come GETFOL disponga di strumenti molto più potenti della semplice applicazione delle regole d'inferenza. Infatti, il teorema si poteva ottenere direttamente tramite l'applicazione del decisore "monadic" [GG88]:

```
GETFOL:: monad forall x. P(x) imp (not exists x. (not P(x)));

8   forall x. P(x) imp (not exists x. (not P(x))).
```

□

Una caratteristica molto importante del sistema, che lo rende particolarmente adatto alla realizzazine di meccanismi di astrazione, è la sua **multicontestualità**. In GETFOL l'utente può creare numerosi "ambienti" distinti, i *contesti*, all'interno dei quali egli può sviluppare delle deduzioni in modo indipendente. Ciascun contesto è dotato di un proprio linguaggio e di propri assiomi, che risultano inaccessibili da un contesto esterno. Esiste inoltre la possibilità di definire un *contesto corrente*: se in un certo istante "*world*" è il contesto corrente, allora tutte le operazioni effettuate dall'utente andranno ad interessare solo *world*, e nessun oggetto appartenente ad altri contesti (simboli dell'alfabeto, assiomi, pezzi di dimostrazioni) potrà essere utilizzato o modificato. Sarà solo possibile "saltare" da un contesto all'altro tramite il comando "*switchcontext*".

Illustriamo tali aspetti in un esempio: [5]

Esempio 2

```
GETFOL:: setprompt to (curcname\-get);

NOTNAMED&:: namecontext AnticaGrecia;
You have named the current context: AnticaGrecia

AnticaGrecia:: declare indconst Socrate;
Socrate has been declared to be an Indconst

AnticaGrecia:: declare indvar x;
x has been declared to be an Indvar

AnticaGrecia:: declare predconst Uomo Mortale 1;
Uomo has been declared to be a Predconst
Mortale has been declared to be a Predconst
```

[5] In questo caso, per rendere l'esempio più comprensibile, il *prompt* di sistema viene definito con il nome del contesto corrente, tramite il comando "*setprompt*".

```
AnticaGrecia:: theory sillogismo :
Uomo(Socrate)
forall x.(Uomo(x) imp Mortale(x));

sillogismo
sillogismo1 : Uomo(Socrate)
sillogismo2 : forall x. (Uomo(x) imp Mortale(x))

AnticaGrecia:: makecontext Disneyland;
You have created the empty context: Disneyland

AnticaGrecia:: switchcontext Disneyland;
You are now using context: Disneyland

Disneyland:: declare indconst Pippo Pluto Paperino;
Pippo has been declared to be an Indconst
Pluto has been declared to be an Indconst
Paperino has been declared to be an Indconst

Disneyland:: term Socrate;
term Socrate;
     ^
The parser requires a <term> here

Disneyland:: andi sillogismo1 sillogismo2;
andi sillogismo1 sillogismo2;
     ^
The reference to a fact is required here.

Disneyland::
```

I comandi "*namecontext*", "*makecontext*" e "*switchcontext*" permettono, rispettivamente, di dare nomi ai contesti, di crearne di nuovi e di "saltare" da un contesto all'altro.

All'interno dell'ambiente "Disneyland", il sistema non riconosce nè il simbolo "Socrate", nè gli assiomi "sillogismo1" e "sillogismo2", in quanto appartengono tutti al contesto "AnticaGrecia". Per questo motivo falliscono sia la chiamata "*term Socrate*" ('Socrate' è un termine?), sia il tentativo di applicare la regola d'inferenza "*and introduction*" ai due assiomi: "*andi sillogismo1 sillogismo2*". □

1.3 L'idea intuitiva di astrazione

A livello intuitivo, per "**astrazione**" possiamo intendere un processo che trasforma una rappresentazione base ("*ground*") di un problema in una nuova rappresentazione "*astratta*", che:

(i) sia ottenuta dalla rappresentazione "ground" eliminando "dettagli", cioè buttando via dell'informazione ridondante ai fini della soluzione del problema;

(ii) sia più semplice da maneggiare;

(iii) aiuti a risolvere il problema a livello "ground", preservando certe desiderabili proprietà.

In tutta la trattazione per "*risolvere un problema*" intenderemo "trovare una prova di un teorema", cioè dedurre una formula da un insieme di assiomi applicando determinate regole d'inferenza.

Esempio 3
Consideriamo il caso di un uomo che debba prendere un treno. Mentalmente, egli può pianificare le azioni necessarie a tale scopo secondo uno schema

$$precondizioni \longmapsto conseguenza,$$

del tipo: "*Consulto l'orario, compro il biglietto, discendo e risalgo il sottopassaggio. A questo punto posso salire sul treno*".

Egli può semplificare tale piano eliminando quelle precondizioni che ritiene banalmente verificabili, e quindi inessenziali nella pianificazione, come "discendere e risalire il sottopassaggio". In questo modo egli esegue un'implicita astrazione del problema, dal mondo delle azioni "*effettive*" a quello delle azioni "*significative*" per la pianificazione.

Seguendo un formalismo logico del primo ordine [Sac74], tale problema può essere rapresentato mediante una formula, scritta nella forma di implicazione tra una congiunzione di premesse (formule atomiche o loro negazioni) ed una conseguenza:

$$(\alpha_1 \wedge \ldots \wedge \alpha_n) \supset \beta.$$

Tali formule vengono astratte eliminando (cioè mappando in "VERO") le premesse α_i che si ritengono essere più facili da verificare, dando origine alla nuova formula:

$$(\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_k}) \supset \beta, \quad i_j \in \{1 \ldots n\},$$

che possiede un numero minore di precondizioni. □

Esempio 4
Supponiamo ora di voler verificare la correttezza del progetto di un certo circuito elettronico digitale, analizzando l'andamento dei segnali logici nei vari punti di interconnessione interna. Se il circuito è molto complicato, un analisi fatta punto per punto può risultare estremamente onerosa.

All'interno del progetto, però, possono esservi numerose occorrenze di uno stesso sotto-circuito (per esempio un "*half adder*"). Potremmo verificare *una volta sola* il funzionamento di tale dispositivo, e poi generare una nuova rappresentazione del circuito globale, in cui ogni occorrenza del dispositivo venga *astratta* in una "scatola bianca", di cui si conoscano solo le relazioni ingresso-uscita. Iterando il procedimento per successivi livelli di astrazione, si ottiene progressivamente una rappresentazione semplificata del problema.

Il procedimento è formalizzabile nell'ambito della logica dei predicati [Mel87]. Possiamo descrivere il "sotto-dispositivo" in questione tramite un predicato

$$Device(in_1, \ldots, in_n, out_1, \ldots, out_k, int_1, \ldots, int_m), \quad in_i, out_j, int_r \in \{0, 1\},$$

dove in_i e out_j rapresentino i segnali logici di ingresso ed uscita ed int_r i segnali nei punti di interconnessione interni. Il funzionamento del dispositivo può quindi venir descritto mediante un assioma del tipo:

$$Device(\underline{in}, \underline{out}, \underline{int}) \longleftrightarrow \varphi(\underline{in}, \underline{out}, \underline{int}),$$

che tiene conto anche del comportamento dei valori interni, $\underline{int}$. Tale assioma può essere piuttosto complicato.

A questo punto possiamo "astrarre" la rappresentazione del circuito globale, sostituendo ogni occorrenza di $Device(\ldots)$ con un predicato di arità minore:

$$WhiteBox(in_1, \ldots, in_n, out_1, \ldots, out_k).$$

Quest'ultimo non contiene riferimenti al comportamento dei valori logici interni $\underline{int}$, e quindi può essere descritto da un assioma assai più semplice:

$$WhiteBox(\underline{in}, \underline{out}) \longleftrightarrow \varphi'(\underline{in}, \underline{out}).$$

□

Nel Capitolo 2 introdurremo una definizione formale di astrazione. Per evitare ambiguità, useremo la parola "`astrazione` " (scritta in caratteri `ttype`) per riferirci al significato intuitivo della parola.

1.4 Esempi di astrazione in letteratura

Accenniamo ora, brevemente, ad alcuni esempi "storici" dell'utilizzo di meccanismi di `astrazione` , attinti dalla letteratura riguardante l'Intelligenza Artificiale, il theorem proving e la logica matematica applicata.

- **Abstrips** [Sac74] [Planning]. Secondo questo formalismo un *piano* viene rappresentato come una formula logica, nella forma:

  $$(\alpha_1(s) \wedge \ldots \wedge \alpha_n(s)) \supset \beta(f(s)).$$

  In questo caso, s rappresenta uno *stato del mondo*, f un'*azione* (una funzione tra stati del mondo), $\alpha_i(s)$ una generica *precondizione*, ed infine $\beta(f(s))$ una *conseguenza*.

Abstrips permette di astrarre questi piani, mappando in "Vero" le precondizioni più facili da verificare. Nell'esempio di p.9, la precondizione "*ho sceso e risalito il sottopassaggio*" viene eliminata sulla base di tale criterio.

- **Resolution plan** [Cha79] [theorem proving].

  Per sistemi di deduzione basati su *Resolution* [Rob65] è stato proposto un metodo per costruire strategie efficienti di risoluzione, basato sulla nozione di "*resolution plan*". Tutte le clausole nella forma "$P_i(t)$" vengono astratte in opportune costanti proposizionali, A_i, una per ogni predicato P_i. Nella teoria proposizionale così generata si creano in seguito delle prove, i resolution plan, che vengono utilizzati come riferimento per cercare una prova del goal originale.

- **Granularità** [Hob85]. [common sense reasoning]. Hobbs propone una *teoria della granularità*, secondo cui una teoria complessa può essere notevolmente semplificata collassando interi gruppi di costanti, tra loro "*indistinguibili*", in un'unico simbolo.

  La nozione di *indistinguibilità* è concepita come una relazione di equivalenza: a è indistinguibile da b se e solo se, per ogni predicato P che riteniamo *rilevante* nella teoria, vale la relazione $P(a) \leftrightarrow P(b)$. Il questo senso, due costanti individuali vengono collassate nello stesso simbolo se la loro distinzione è ritenuta irrilevante.

- **Hardware verification** [Mel87] Come abbiamo già visto nell'esempio di p.9, un meccanismo di astrazione può essere utilizzato per semplificare l'operazione di verifica nel progetto di un circuito elettronico digitale.

  A tale fine, si può rappresentare un generico sotto-circuito con un predicato $Device(\underline{in}, \underline{out}, \underline{int})$, e generare poi una rappresentazione "astratta" del sistema, in cui ogni occorrenza di $Device(\ldots)$ venga sostituita con $WhiteBox(\underline{in}, \underline{out})$, il quale rappresenta solo la relazione ingresso-uscita del sotto-circuito in questione.

- ...

Oltre ai singoli esempi illustrati, e a molti altri non esplicitamente citati, vi sono stati alcuni significativi lavori volti a catturare la nozione di `astrazione` .

- La prima definizione sufficientemente generale venne introdotta da D.A.Plaisted [Pla80, Pla81]. Egli definì il concetto di astrazione, tra sistemi utilizzanti Resolution, come una generica funzione tra formule che verifichi determinate proprietà. Tale definizione garantiva la "maggior semplicità" di una prova a livello astratto.

- J.D.Tenenberg [Ten87, Ten88] approfondì l'analisi delle astrazioni che "collassano" predicati. In particolare, si dedicò al problema della generazione di spazi inconsistenti (cfr. p. 45). Arrivò infine ad introdurre alcune nozioni "ristrette" di astrazione, che evitano di generare inconsistenza.

Con il lavoro di F.Giunchiglia e T.Walsh (1987-1991) si è giunti alla formulazione di una *Teoria Generale dell'Astrazione* [Giu], introducendo una definizione generale di **astrazione** tra sistemi formali del primo ordine. In primo uogo vengono analizzate le proprietà formali di tale definizione, classificando i vari tipi di astrazioni in base a tali proprietà. Viene inoltre effettuato uno studio sul possibile uso dell'astrazione nella dimostrazione automatica/interattiva di teoremi ("*abstract theorem proving*").

In [Giu] è stato dimostrato come la definizione introdotta catturi la maggior parte delle astrazioni reperibili in letteratura e sia *più generale* di ogni precedente definizione. In particolare, gran parte dei risultati proposti in [Pla80, Pla81, Ten87] vengono catturati da risultati più generali della teoria.

Il Capitolo 2 presenta un riassunto di tale teoria. Essa costituisce il principale riferimento teorico per il lavoro presentato in questa tesi.

1.5 Scopo del lavoro

Il lavoro di questa tesi si inserisce in un progetto più ampio, volto a realizzare un sistema di deduzione automatica basato sull'astrazione ("*abstract theorem prover*"). Tale sistema viene realizzato come un *modulo* del sistema `GETFOL`.

Secondo la *Teoria Generale dell'Astrazione*, il processo di "*abstract theorem proving*" si sviluppa secondo una sequenza di fasi distinte (cfr Capitolo 2):

(i) **Scelta** e **definizione** di un'astrazione adatta allo scopo.

(ii) **Astrazione** vera e propria. Una formula φ da dimostrare viene mappata in una nuova formula appartenente ad un diverso contesto, $f_\Lambda(\varphi)$, utilizzando l'astrazione definita nella fase (i). In questo senso si preferisce parlare di *applicazione* di un'astrazione a φ, piuttosto che di *astrazione* della formula stessa.

(iii) **Dimostrazione**: si costruisce una prova di $f_\Lambda(\varphi)$ nello spazio astratto. Questa fase non coinvolge direttamente il meccanismo di astrazione.

(iv) **Mapping back**: si costruisce una dimostrazione di φ nello spazio ground, prendendo come riferimento la prova di $f_\Lambda(\varphi)$.

Quest'ultima viene "*mappata all'indietro*", generando un albero di formule ground, eventualmente parametrico. Tale albero, che costituisce la "struttura" della dimostrazione cercata, viene successivamente *rifinito*, aggiungendo i particolari mancanti alla stesura definitiva della prova.

Scopo primario del lavoro di questa tesi consiste nella realizzazione di una serie di *comandi* in `GETFOL` che permettano all'utente di *definire* un'astrazione e di *applicarla* a vari *oggetti* di un contesto (simboli logici, termini, formule, assiomi, teorie ...), generando i corrispondenti oggetti nel contesto astratto. In questo modo vengono realizzate le fasi (i) e (ii) dello schema precedente.

Si noti, per inciso, come questo lavoro abbia una sua utilità autonoma dalla realizzazione della fase (iv). In primo luogo, la nozione di astrazione che verrà introdotta può descrivere anche processi che non sono esualmente pensati come astrazioni (si veda l'esempio di p.81). Inoltre, la costruzione di una prova astratta può essere, anche da sola, un validissimo aiuto alla costruzione "manuale" della dimostrazione ground. Infine, esistono delle astrazioni per le quali una prova di una formula livello astratto lo è anche per la stessa formula livello ground (cfr. Esempio 4). In tal caso la fase di mapping back è inutile.

Prima di realizzare i comandi per l'astrazione, si è reso necessario affrontare una serie di problematiche teoriche.

(*a*) La nozione di astrazione riportata nella teoria riguarda sistemi formali "classici". Il sistema `GETFOL` "estende" tale nozione, utilizzando una logica "multisorte" [6], detta "*Sorted Natural Deduction*" [CG90].

Per sfruttare pienamente le potenzialità di tale definizione, si è dovuto quindi studiare la possibilità di estendere la definizione formale di astrazione a logiche di tipo "sorted". I primi risultati di questo lavoro teorico sono riportati nel Capitolo 3.

(*b*) Nella teoria un'astrazione viene definita mediante una generica *funzione* f_Λ, computabile, suriettiva e totale, tra i linguaggi di due sistemi formali del primo ordine ("contesti"). Tale nozione, però, *non dà alcun suggerimento su come questa funzione possa venire definita operativamente.*

Per questo motivo si è reso necessario introdurre, a livello teorico, una definizione effettiva di "funzione tra linguaggi del primo ordine", f_Λ, che verificasse alcuni requisiti tali da renderla utilizzabile in una definizione "operativa" di astrazione.

(*c*) Nella teoria, inoltre, si richiede implicitamente che il contesto astratto *esista già* all'istante in cui viene definita un'astrazione. Operativamente, prima di definire un'astrazione, l'utente dovrebbe generare l'alfabeto dello spazio astratto ed i relativi assiomi, operazione lunga e noiosa.

Si vorrebbe, quindi, poter generare automaticamente l'alfabeto e/o gli assiomi a livello astratto, come immagine di quelli ground rispetto ad f_Λ. A questo scopo è necessario, come vedremo nel Capitolo. 4, estendere la nozione di mappa tra due linguaggi anche a singoli simboli dell'alfabeto [7].

L'analisi approfondita di tali problematiche, e la proposta di soluzioni appropriate, costituiscono la componente teorica di questa tesi. Essa verrà interamente esposta nei Capitoli 3 e 4.

[6] Per la definizione di sorte e sistemi multisorte cfr. Capitolo 3.

[7] Nella *Teoria dell'Astrazione* per "linguaggio"si intende implicitamente l'insieme delle formule logiche costruibili dai simboli di un'alfabeto. In questo modo, i simboli stessi non appartengono al linguaggio. Si noti infatti, ad esempio, che il simbolo di funzione "f" non è un'espressione logica.

1.6 Obiettivi raggiunti

Con il lavoro svolto in questa tesi, sono stati raggiunti alcuni importanti obiettivi.

Nel Capitolo 3 è stata introdotta una nuova definizione di astrazione, estesa a sistemi formali *multisorte*. Si è poi dimostrato (cfr. Teoremi 4, 5 e 6) come tale definizione "estesa" conservi, nel caso di sistemi "*Sorted Natural Deduction*", certe fondamentali proprietà delle astrazioni tra sistemi "ordinari".

Nel Capitolo 4 è stata introdotto un formalismo per descrivere funzioni tra linguaggi logici del primo ordine, che estende la definizione di f_Λ anche ai simboli dell'alfabeto. La nozione di astrazione risultante viene ad essere sufficientemente generale, tale cioè da catturare la definizione formale introdotta nella teoria, pur essendo relativamente semplice ed intuitiva.

Sulla base delle nozioni illustrate nel Capitolo 4, sono stati realizzati i comandi per *dichiarare* ed *applicare* le astrazioni, oltre ad alcuni semplici comandi di utilità.

In particolare, l'introduzione dei *prototipi di astrazione* permette all'utente di dichiarare un'astrazione in modo indiretto, istanziando un prototipo standard al linguaggio corrente. In questo modo, creare nuove astrazioni diviene un'operazione sufficientemente semplice.

Per quanto riguarda la fase di astrazione vera e propria, la nozione "operativa" di astrazione introdotta permette all'utente (ma non lo obbliga!) di generare automaticamente l'alfabeto e/o gli assiomi del contesto astratto.

Riassumendo, è stato definito un **linguaggio** per definire astrazioni, relativamente semplice ed espressivo, ed è stato realizzato un **interprete** per tale linguaggio.

Capitolo 2

La teoria dell'astrazione

2.1 Introduzione

Questo capitolo rappresenta un riassunto della **Teoria Formale dell'Astrazione** elaborata da Fausto Giunchiglia e Toby Walsh dal 1987 al 1991 [GW89a, GW89b, GW89c, GW90a, GW90b, Giu, GW90c, GW91, F.]. Attualmente (ottobre 1991) la trattazione completa più esauriente di tale teoria può essere considerata [Wal90].

Alcuni nuovi esempi sono stati introdotti per una migliore comprensione della teoria. Inoltre sono stati inserite alcune note e commenti personali che, senza avere la pretesa di aggiungere nulla di nuovo alla teoria stessa, cercano di chiarire il significato intuitivo di certi aspetti della trattazione, in particolar modo per quanto riguarda il concetto di "maggior semplicità" di una dimostrazione nello spazio astratto e quello di "composizione" tra astrazioni.

2.2 Definizione formale di astrazione

2.2.1 Sistemi formali ed astrazioni

Diamo innanzitutto due fondamentali definizioni:

Definizione 1 (Sistema Formale) *: Un sistema formale* Σ *è una tripla* $\langle \Lambda, \Omega, \Delta \rangle$*, dove* Λ *è il* **linguaggio**, Ω *è* **l'insieme di assiomi** *e* Δ *è* **l'insieme delle regole d'inferenza** *di* Σ*.*

Un *linguaggio* è definito tramite un *alfabeto* e l'insieme dei *termini* e delle *formule* costruibili dai simboli dell'alfabeto tramite le usuali regole di ben formatezza della logica del primo ordine. In tutta la trattazione useremo le espressioni "*contesto*" e "*spazio*" quali sinonimi di "*sistema formale*". Inoltre, se non ulteriormente specificato, ci riferiremo sempre a sistemi formali nella logica dei predicati del primo ordine.

Definizione 2 (Astrazione) *: Un' astrazione* $f : \Sigma_1 \Rightarrow \Sigma_2$ *è una tripla* $\langle \Sigma_1, \Sigma_2, f_\Lambda \rangle$*, dove*

$$\Sigma_1 = \langle \Lambda_1, \Omega_1, \Delta_1 \rangle$$
$$\Sigma_2 = \langle \Lambda_2, \Omega_2, \Delta_2 \rangle$$

sono due sistemi formali, detti rispettivamente "ground" ed "astratto", e

$$f_\Lambda : \Lambda_1 \rightarrow \Lambda_2$$

è una funzione **computabile, suriettiva** *e* **totale** *tra i due linguaggi.*

Useremo spesso la parola *mappa* come sinonimo di *astrazione*.

La definizione di f_Λ richiede l'ipotesi di *computabilità* in quanto tutta la teoria è finalizzata ad un uso *effettivo* (ad es. in un sistema di deduzione automatica) dell'astrazione. Tale funzione dev'essere inoltre *totale* in quanto si vuole poter sempre mappare un'espressione nello spazio astratto. La *suriettività* è richiesta in quanto si vuole poter sempre mappare "all'indietro" ogni espressione del contesto astratto.

In [Giu] è stato dimostrato come tale definizione catturi la maggior parte delle astrazioni definite in letteratura e sia più generale di ogni precedente definizione (es. [Pla80, Pla81, Ten87]). A causa della sua generalità, essa cattura inoltre alcuni tipi di mappe che non sono usualmente pensate come astrazioni.

Nella nozione di astrazione precedentemente introdotta si può intravvedere una limitazione: lo spazio astratto Σ_2 (e quindi Λ_2 e Ω_2) deve esistere *a priori* rispetto a f_Λ. Questo significa che Σ_2 deve essere esplicitamente dichiarato dall'utente *prima* di definire ogni astrazione di cui esso costituisca lo spazio astratto. Tale operazione può essere operativamente molto onerosa.

Sarebbe quindi desiderabile, in alternativa, poter **generare automaticamente** Σ_2 tramite f_Λ, ($\Sigma_2 := f(\Sigma_1)$. Per quanto riguarda l'alfabeto del linguaggio astratto, $\mathcal{A}_2$, la suriettività di f_Λ garantisce tale possibilità ($\mathcal{A}_2 := f_\Lambda(\mathcal{A}_1)$. Sarebbe inoltre desiderabile poter generare anche alcuni (o tutti) gli assiomi di Ω_2 come immagine di alcuni (o tutti) gli assiomi di Ω_1 ($\Omega_2 := f_\Lambda(\Omega_1)$). Vedremo nei capitoli successivi come la meccanizzazione dell'astrazione presentata in questo lavoro permetta queste opportunità.

2.2.2 Operazioni sulle astrazioni

Alcune operazioni sulle astrazioni sono state definite nella teoria (ad es. in [Giu]). La definizione di molte di esse è immediatamente intuibile, come quella di **uguaglianza** tra astrazioni (quando sia la mappa del linguaggio che i due sistemi formali sono uguali), di esistenza dell'**astrazione identità** (quando $\Sigma_1 = \Sigma_2$ e f_Λ è l'identità), e di **astrazione inversa** (quando f_Λ è invertibile, $f^{-1} : \Sigma_2 \Rightarrow \Sigma_1$ è tale che $(f^{-1})_\Lambda = (f_\Lambda)^{-1}$).

Vale invece la pena di soffermarsi un momento sul concetto di **composizione** tra due astrazioni, e sul suo significato pratico:

Definizione 3 (Composizione tra astrazioni) *: Se* $f : \Sigma_1 \Rightarrow \Sigma_2$ *e* $g : \Sigma_2 \Rightarrow \Sigma_3$ *sono due astrazioni allora* $h : \Sigma_1 \Rightarrow \Sigma_3$ *è detta la loro* **astrazione composta** *se* $h_\Lambda = g_\Lambda \circ f_\Lambda$ [1].

Tale definizione potrebbe sembrare ovvia, ma il suo significato pratico non lo è. Una volta dichiarata, h diviene un'astrazione a sè, indipendente da f, da g e soprattutto da Σ_2. Se a viene mappato in b da g_Λ e b in c da f_Λ, h_Λ mappa a in c *direttamente*, senza passare per b. Se si vuole, si può fare un'analogia con gli operatori matriciali dell'algebra lineare: la composizione (prodotto) di due matrici è ancora una matrice ed è indipendente dallo spazio lineare "intermedio". Si possono pensare quindi 2 modi operativi di realizzare composizione $[g \circ f]$ tra due astrazioni:

- **iterazione** ($g_\Lambda(f_\Lambda(x))$): si applica f all'oggetto desiderato e poi si applica ulteriormente g al risultato ottenuto . Per tale applicazione è necessario passare per lo spazio intermedio Σ_2.
- **composizione** ($[g_\Lambda \circ f_\Lambda](x)$): la composizione avviene *in fase di dichiarazione.* $h : \Sigma_1 \Rightarrow \Sigma_3$ viene dichiarata per composizione di f e g e da questo momento agisce tra Σ_1 e Σ_3 in modo assolutamente autonomo.

Si è introdotta precededentemente l'idea della possibile funzionalità "generativa" di un'astrazione. In questo caso l'uso di un'astrazione "composta" risulta essere determinante, in quanto permette di generare lo spazio astratto Σ_3 senza generare il contesto intermedio Σ_2.

2.3 Una prima classificazione delle astrazioni

Abbiamo visto come un sistema formale sia composto da un linguaggio, un insieme di assiomi ed uno di regole di inferenza. Introduciamo un primo tipo di classificazione, a cui faremo riferimento in seguito, la quale si fonda sul modo in cui un'astrazione mappa tali componenti [Giu].

Definizione 4 (Astrazione Λ/Ω-invariante) *: Siano* $\Sigma_1 = \langle \Lambda_1, \Omega_1, \Delta_1 \rangle$ *e* $\Sigma_2 = \langle \Lambda_2, \Omega_2, \Delta_2 \rangle$ *due sistemi formali. Un'astrazione* $f : \Sigma_1 \Rightarrow \Sigma_2$ *è detta* **Λ/Ω-invariante** *sse* $\Omega_2 = f_\Lambda(\Omega_1)$.

Tali astrazioni sono utili quando non è richiesta alcuna distinzione tra assiomi e formule. Questo è spesso vero nei casi in cui vogliamo *generare* lo spazio astratto tramite la mappa f_Λ, oppure quando non vi siano particolari vincoli relativi all'assiomatizzazione dello spazio astratto stesso.

Definizione 5 (Astrazione Δ-invariante) *: Siano* $\Sigma_1 = \langle \Lambda_1, \Omega_1, \Delta_1 \rangle$ *e* $\Sigma_2 = \langle \Lambda_2, \Omega_2, \Delta_2 \rangle$ *due sistemi formali. Un'astrazione* $f : \Sigma_1 \Rightarrow \Sigma_2$ *è detta* **Δ-invariante** *sse* $\Delta_2 \subseteq \Delta_1$.

[1] si sottintende l'uso della notazione italiana per "$\circ$": $[g_\Lambda \circ f_\Lambda](x) = g_\Lambda(f_\Lambda(x))$.

Gran parte delle astrazioni sono Δ-invarianti, ed utilizzano lo stesso motore inferenziale in entrambi i contesti.

Definizione 6 (Astrazione Λ/Δ-invariante) *: Siano $\Sigma_1 = \langle\Lambda_1, \Omega_1, \Delta_1\rangle$ e $\Sigma_2 = \langle\Lambda_2, \Omega_2, \Delta_2\rangle$ due sistemi formali. Un'astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta* **Λ/Δ-invariante** *sse*

$$\Delta_2 = \{\frac{f_\Lambda(\alpha_1), ..., f_\Lambda(\alpha_n)}{f_\Lambda(\alpha_{n+1})} \quad | \quad \frac{\alpha_1, ..., \alpha_n}{\alpha_{n+1}} \in \Delta_1\}$$

La Λ/Δ-invarianza richiede che, per ogni inferenza, l'immagine della conseguenza sia direttamente ricavabile dalle immagini delle ipotesi. Si noti che non c'è alcuna relazione di interdipendenza con la definizione di Δ-invarianza. Assieme alla Λ/Ω-invarianza, essa costituisce una proprietà molto forte, in quanto permette di astrarre dimostrazioni in dimostrazioni: infatti se $f : \Sigma_1 \Rightarrow \Sigma_2$ è un'astrazione contemporaneamente Λ/Ω- e Λ/Δ-invariariante, e se Π è una prova di φ, allora $f_\Lambda(\Pi)$ è una prova di $f_\Lambda(\varphi)$. Tali astrazioni sono dette**Σ-invarianti**.

Introduciamo ora due definizioni estremamente importanti:

Definizione 7 (Astrazione teoria-invariante) *: Un'astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ è chiamata* **teoria-invariante** *sse per ogni formula α e β si ha che:*

$$f(\alpha \otimes \beta) = f(\alpha) \otimes f(\beta)$$

per ogni connettivo logico $\otimes$ (con una condizione analoga per i connettivi unari e n-ari), e

$$f(\diamond x.\alpha) = \diamond x.f(\alpha)$$

per ogni quantificatore $\diamond$.

Definizione 8 (Astrazione atomica) *Un'astrazione teoria-invariante $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta* **atomica** *sse $\Omega_2 = f_\Lambda(\Omega_1)$ e $\Delta_1 = \Delta_2$.*

L'idea fondamentale espressa da tali definizioni è che un'astrazione teoria-invariante *mappa la teoria e mantiene inalterata la logica*, nel senso che preserva inalterate le regole di ben formatezza delle formule, limitandosi astrarre le formule atomiche.Si noti come essa possa anche eliminare variabili, in quanto se $f_\Lambda(\varphi)$ non contiene occorrenze di x, allora $\diamond x.f(\varphi)$ è logicamente equivalente a $f(\varphi)$.

Le nozioni di teoria-invarianza ed atomicità possono sembrare restrittive. In realtà la maggior parte delle astrazioni di uso pratico verificano almeno la prima definizione, molto spesso anche la seconda. La spiegazione è semplice: le regole di ben formatezza della logica sono bilanciate molto finemente, al punto che ogni minima alterazione (con qualche eccezione) può causare seri problemi.

2.4 Proprietà preservate da un'astrazione:

Abbiamo informalmente introdotto l'idea di `astrazione` come una mappa tra diverse rappresentazioni di un problema che preserva certe desiderabili proprietà delle formule e delle relative dimostrazioni. La scelta di tali proprietà dipende strettamente dal tipo di sistema di deduzione, e dall'idea di "*maggior semplicità*" che vogliamo catturare.

2.4.1 Dimostrabilità

Nei sistemi di *dimostrazione* siamo interessati a dimostrare la *verità* di una formula, cioè a dedurla [2] come vera da un certo numero di assiomi utilizzando determinate regole di inferenza. In tutta la trattazione, chiameremo $TH(\Sigma)$ l'insieme delle formule del linguagio Λ dimostrabili in Σ. Siamo quindi interessati a sapere quali relazioni possano intercorrere tra la dimostrabilità di una formula ground e quella della sua immagine.
Introduciamo alcune definizioni:

Definizione 9 (astrazioni T*) *: Un' astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta* [3]

1. **TC**, *sse per ogni formula α vale: $\alpha \in TH(\Sigma_2) \Leftrightarrow f_\Lambda(\alpha) \in TH(\Sigma_2)$;*
2. **TD**, *sse per ogni formula α vale: $f_\Lambda(\alpha) \in TH(\Sigma_2) \Rightarrow \alpha \in TH(\Sigma_1)$;*
3. **TI**, *sse per ogni formula α vale: $\alpha \in TH(\Sigma_1) \Rightarrow f_\Lambda(\alpha) \in TH(\Sigma_2)$.*
   Le astrazioni TI sono anche chiamate **truthful**;

Chiameremo un'astrazione "**strettamente**" TD [TI] sse essa è TD [TI] ma non è TC.

Le astrazioni **TI** vengono usate in modo **addutivo**: se φ è un teorema, anche $f_\Lambda(\varphi)$ lo è. Come vedremo più avanti, la dimostrazione di $f_\Lambda(\varphi)$ "suggerisce" quella di φ, anche se non ne garantisce l'esistenza. Questo limitazione potrebbe far sembrare tali astrazioni poco utili. Al contrario, è opinione degli autori che le astrazioni di questo tipo siano le più adatte a rappresentare l'idea di `astrazione`:

"...riteniamo che (certe sottoclassi del)le astrazioni TI costituiscano il formalismo più appropriato per `astrazione` *.."* ([Giu], p.6).

Questa apparente contraddizione è facilmente spiegabile:
noi richiediamo a $f : \Sigma_1 \Rightarrow \Sigma_2$ di "buttare via dettagli" che riteniamo inutili ai fini della nostra dimostrazione, per rendere la deduzione "più semplice" nello spazio astratto. Tale azione in generale può fare sì che qualche altro teorema a livello "astratto" non sia più immagine di un teorema "ground". È compito dell'utente la scelta di un'astrazione "*ad hoc*", che elimini particolari realmente ridondanti conservando però tutta l' informazione utile alla dimostrazione del teorema richiesto. Non è un caso che gran parte degli esempi di astrazione reperibili in letteratura si dimostrino essere di tipo TI (cfr. [Giu], pp.13-26).

[2]Per semplicità di esposizione supponiamo di lavorare in sistemi formali in cui valga il teorema di deduzione e dotati di MODUS PONENS, ad esempio Natural Deduction [Pra65]. In questo caso possiamo permetterci (cfr. [Giu] p.5) di "confondere", a livello intuitivo, il concetto di "deduzione" con quello di "dimostrazione".

[3]"T" sta per "theorem", "C" per "constant", "D" per "decreasing" e "I" per "increasing".

Le astrazioni **TD** vengono usate in modo **deduttivo**: se $f_\Lambda(\varphi)$ è un teorema, anche φ lo è. Spesso la dimostrazione di φ risulta immediatamente deducibile dalla dimostrazione di $f_\Lambda(\varphi)$.

Il primo inconveniente presentato da questo gruppo di astrazioni è la loro *non completezza*: non siamo sicuri che $f_\Lambda(\varphi)$ sia un teorema anche se φ lo è. Potremmo perdere inutilmente molto tempo a tentar di provare una formula $f_\Lambda(\varphi)$ non dimostrabile . Anche qui una scelta "ad hoc" dell'astrazione scavalca questa limitazione.

Il maggiore inconveniente delle astrazioni TD è che esse talvolta non danno origine ad una rappresentazione "più semplice". Se nel sistema formale astratto è dimostrabile una frazione "minore" di formule, si può *presupporre* che, in qualche modo, trovare una prova a livello astratto *possa* (ma non debba!) essere "più difficile" che nel sistema ground. In generale, infatti, il numero di esempi di astrazioni TD reperibili in letteratura è inferiore a quello delle TI (cfr. [Giu], p.26).

Questo non significa che le astrazioni TD siano poco utili (cfr. ad es. [GG88]). Esse corrispondono solo ad un concetto diverso e meno comune di "maggior semplicità" di una deduzione.

Le astrazioni **TC** sono contemporaneamente TD e TI. A parte qualche significativa eccezione, esse sono poco usate, poiché le ipotesi su cui si basa la loro definizione sono piuttosto restrittive e quindi catturano pochi casi utili. In generale esse si limitano ad effettuare un cambio di rappresentazione, senza eliminare informazione.

2.4.2 Inconsistenza

Nei sistemi di *refutazione*, ad esempio Resolution [Rob65], una formula α viene dimostrata se, aggiungendo la sua negazione agli assiomi di un sistema formale, si genera inconsistenza. In formule:

$$\Omega_\Sigma \vdash_\Sigma \alpha \qquad \Longleftrightarrow \qquad \Omega_\Sigma \bigcup \{\neg\alpha\} \vdash_\Sigma \bot.$$

In tutta la trattazione, chiameremo $NTH(\Sigma)$ l'insieme delle formule che, aggiunte agli assiomi di Σ, generano inconsistenza.

Per i sistemi a refutazione esiste una teoria dell'astrazione assolutamente duale rispetto a quella dei sistemi di deduzione.

Si introducono le definizioni:

Definizione 10 (astrazioni NT*) *: Un' astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta*

1. **NTC**, *sse per ogni formula α vale: $\alpha \in NTH(\Sigma_2) \Leftrightarrow f_\Lambda(\alpha) \in NTH(\Sigma_2)$;*

2. **NTD**, *sse per ogni formula α vale: $f_\Lambda(\alpha) \in NTH(\Sigma_2) \Rightarrow \alpha \in NTH(\Sigma_1)$;*

3. **NTI**, *sse per ogni formula α vale: $\alpha \in NTH(\Sigma_1) \Rightarrow f_\Lambda(\alpha) \in NTH(\Sigma_2)$.*

Le proprietà delle astrazioni NT* sono assolutamente duali rispetto a quelle delle T* e non verranno quindi descritte in questa trattazione. Sono stati dimostrati [Giu] un certo numero di risultati riguardanti le interrelazioni tra queste categorie. È stato inoltre osservato come, con rare eccezioni, quasi tutti gli esempi di astrazione reperibili in letteratura siano classificabili come T* e/o NT*. Tali risultati non vengono qui descritti per brevità di trattazione.

La proprietà di essere T* o NT* viene in molti casi conservata dalla composizione tra astrazioni. La tabella seguente [Giu] riassume tale comportamento. Il simbolo "?" significa che nessuna di tali proprietà è predicibile.

$\circ$	TD	TC	TI	NTD	NTC	NTI
TD	TD	TD	?	?	?	?
TC	TD	TC	TI	?	?	?
TI	?	TI	TI	?	?	?
NTD	?	?	?	NTD	NTD	?
NTC	?	?	?	NTD	NTC	NTI
NTI	?	?	?	?	NTI	NTI

2.4.3 Dimostrabilità e struttura di una dimostrazione

Una proprietà desiderabile di un astrazione è quella di stabilire relazioni non solo tra la dimostrabilità di una formula e quella della sua immagine, ma anche tra la struttura delle rispettive dimostrazioni.

Per chiarire il significato di tale affermazione, è necessario introdurre alcuni nuovi concetti.

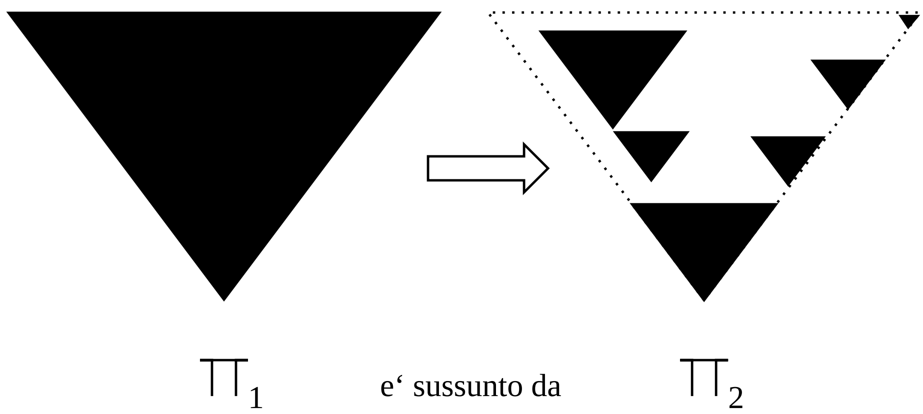
Definizione 11 (Sussunzione tra alberi) *:*
L'albero Π_1 **sussume** *l'albero* Π_2 *(*$\Pi_1 \subseteq \Pi_2$*) sse*

- $\Pi_1 = \varphi$ *e* φ *occorre in* Π_2*;*
-
$$\Pi_1 = \frac{\Gamma_1 ... \Gamma_n}{\varphi}$$
e φ *è la radice di un sottoalbero* Γ *di* Π_2*, e per ogni* i *tale che* $1 \leq i \leq n$*, esiste un distinto sottoalbero stretto* Γ'_i *di* Γ*, tale che* $\Gamma_i \subseteq \Gamma'_i$*.*

Intuitivamente, "$\Pi_1 \subseteq \Pi_2$" significa che un albero Π_1 può essere ottenuto da un'albero Π_2 "tagliandone" alcuni sottoalberi. È importante notare come tale definizione sia **topologica** e quindi non riguardi solo alberi di formule, come nel nostro caso.

Molte interessanti proprietà della relazione di sussunzione sono documentate (e dimostrate) in [GW90a].

Una rappresentazione grafica intuitiva della relazione di sussunzione tra alberi è data in figura:



Esempio 5

Consideriamo i seguenti alberi di formule:

$$\Pi_1 = \cfrac{\cfrac{A \quad A}{B} \quad C}{D} \qquad \Pi_2 = \cfrac{A \quad A}{B} \qquad \Pi_3 = \cfrac{A \quad C}{D} \qquad \Pi_4 = \cfrac{\cfrac{A}{A}}{B}$$

Si osserva che:

- Π_2 e Π_3 sussumono entrambi Π_1.
- Π_2 è un sottoalbero di Π_1 ma Π_3 non lo è. La relazione di sussunzione infatti è più debole di quella di sottoalbero.
- Π_2 non sussume Π_4.

□

Consideriamo ora solo alberi di formule. La relazione di sussunzione ci suggerisce intuitivamente che, se $(\Pi_1 \subseteq \Pi_2)$, allora Π_2 è ottenibile da Π_1 aggiungendo delle inferenze "ponte" nella sua parte interna. Π_1 può quindi essere uno "*scheletro*" di una dimostrazione Π_2. Possiamo inoltre facilmente osservare che, per ogni astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$, si ha $f(\Pi_1) \subseteq f(\Pi_2)$.

Questo ci porta ad un'importante definizione:

Definizione 12 (astrazione PI) [4]*:*
Un' astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta **PI** *sse, per ogni dimostrazione Π_1 di φ in Σ_1, ne esiste una di $f(\varphi)$ in Σ_2, Π_2, tale che $\Pi_2 \subseteq f(\Pi_1)$.*

[4] "P" sta per "proof" ed "I" per "increasing".

Notiamo come $f(\Pi_1)$ sia un generico albero di formule e non necessariamente una dimostrazione. È immediato inoltre osservare come una mappa PI sia anche TI, mentre il viceversa non è vero in generale.

Un'astrazione PI non si limita ad affermare la dimostrabilità dell'immagine di un teorema "ground", ma introduce anche una relazione tra le rispettive prove: la dimostrazione astratta è in qualche modo "contenuta" nell'immagine di quella "ground". Come banale conseguenza, si ha che esiste nello spazio ground (almeno) un albero di formule Π'_1 "contenuto" nella dimostrazione ground e la cui immagine è proprio la prova astratta: tale albero verrà in seguito chiamato *proof plan*.

La nozione di astrazione PI può sembrare restrittiva. In realtà è stato dimostrato come gran parte delle astrazioni interessanti venga catturata da tale definizione. In particolare l'importante (e molto generale) categoria delle astrazioni **atomiche** la verifica sotto condizioni sufficientemente deboli:

Teorema 1 *([GW90a] p.10)*
Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione atomica tra due sistemi formali del primo ordine. Se f preserva i nomi delle variabili libere e vincolate (o le elimina), e se preserva ogni istanza di sostituzione (cioè $f(p[a]) = f(p[x])\{f(a)/x\}$), allora è un'astrazione PI.

Vedremo in seguito la grande importanza delle astrazioni PI.

2.5 Uso dell'astrazione in theorem proving

La definizione delle astrazioni PI permette di definire in modo molto più esplicito alcuni concetti precedentemente accennati, in particolare chiarisce il significato della frase: "*...si usa la dimostrazione astratta come riferimento per costruire la dimostrazione ground ...*".

Per questo scopo dobbiamo introdurre un nuovo concetto:

Definizione 13 (proof plan) *: Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione e siano Π_1 e Π_2 le prove, rispettivamente, delle formule φ e $f_\Lambda(\varphi)$. Un albero Π'_1 e' detto* **proof plan** [5] *sse $f(\Pi'_1) = \Pi_2$ e $\Pi'_1 \subseteq \Pi_1$.*

E' immediato osservare che, se $f : \Sigma_1 \Rightarrow \Sigma_2$ e' PI, allora esiste sempre (almeno) un proof plan.

Un proof plan Π'_1 e' un generico albero di formule e *non una dimostrazione*: è infatti costituito da singoli "pezzetti" di prova tra loro logicamente scollegati, detti *isole* (si veda la figura a p. 31 e l'esempio di p. 31). Esso può quindi essere visto come lo "scheletro" di una prova di φ, ed è "*suggerito*" dalla dimostrazione astratta di $f(\varphi)$. La prova richiesta può essere successivamente ottenuta "*rifinendo*" il proof plan, cioè aggiungendo archi e sottoalberi "ponte" tra le singole isole interne ("*jumping betwen islands*"). In questo senso la prova Π_2 serve come **riferimento** alla costruzione di Π_1.

[5] Un proof plan Π'_1 viene talvolta indicato convenzionalmente come $f^{-1}(\Pi_2)$. Poiché f_Λ non è in generale iniettiva, tale notazione non ha nulla a che vedere con l'usuale inversa di una funzione.

La deduzione di un teorema tramite astrazione è quindi un processo che si articola in quattro fasi:

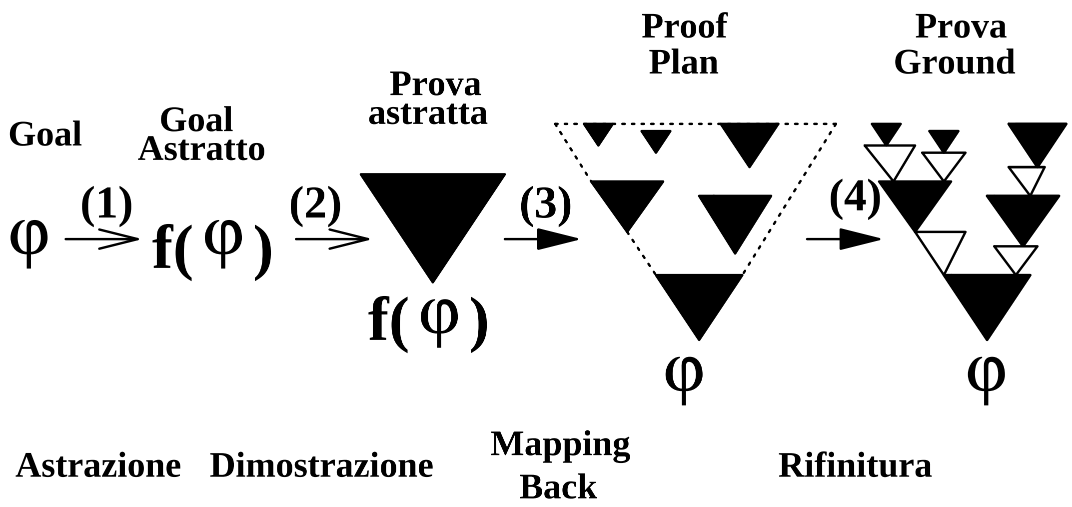
1. **Astrazione**: la formula φ da dimostrare viene mappata nello spazio astratto, creando la nuova formula $f_\Lambda(\varphi)$. In realtà preferiamo parlare di *applicazione* di un'astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ a φ piuttosto che di *astrazione di* φ: questo aspetto verrà approfondito in seguito.

2. **Dimostrazione**: si trova una dimostrazione Π_2 di $f_\Lambda(\varphi)$ nello spazio astratto.

3. **Mapping back**: si costruisce un proof plan (o uno schema di proof plans), Π'_1, secondo la definizione precedente. poiché f_Λ non è in generale iniettiva, si dovrà introdurre alcune meta-variabili "sorted" al posto di ogni simbolo la cui controimmagine non sia univoca (cioè tale che esistano più simboli ground mappati da f_Λ nel simbolo stesso). Il range di tali variabili sarà proprio l'insieme delle controimmagini relative.

4. **Rifinitura**: si ottiene la prova richiesta Π_1,instanzioando appropriatamente le meta-variabili ed introducendo alcune sottoprove "ponte" per collegare logicamente i singoli pezzi di deduzione.

È importante osservare come questo processo sia implicitamente preceduto da una fase preliminare non meno importante: *la scelta e la conseguente definizione di un'astrazione adatta allo scopo.*



Si noti come solo nelle fasi 2 e 4 venga effettuata un'azione di theorem proving. Nele fasi 1 e 3 è il gestore delle astrazioni ad effettuare le operazioni di mapping in avanti ed indietro; nella fase 4 quest'ultimo interagisce con il theorem prover.

Esempio 6
Consideriamo la seguente astrazione, che rappresenta due diversi modi con cui un animale domestico può chiamare il proprio padrone:

Spazio Ground (Σ_1):

- Linguaggio (Λ_1): {*Cat()* ,*Dog()* ,*Meows()*, *Barks()*, *Pussy*, *Tom*}.
- Assiomi (Ω_1):
$$Cat(Pussy)$$
$$Dog(Tom)$$
$$\forall x.(Cat(x) \supset Meows(x))$$
$$\forall x.(Dog(x) \supset Barks(x))$$
- Regole di inferenza (Δ_1): Natural Deduction.

Mappa del linguaggio (f_Λ):

$$\begin{array}{lcl} Pussy & \longmapsto & Poppy \\ Tom & & \\ Cat() & \longmapsto & Pet() \\ Dog() & & \\ Meows() & \longmapsto & Calls() \\ Barks() & & \end{array}$$

e conserva la struttura logica di un'espressione.

Spazio Astratto (Σ_2):

- Linguaggio (Λ_1): {*Pet()* , *Calls()*, *Poppy*}.
- Assiomi (Ω_2):
$$Pet(Poppy)$$
$$\forall x.(Pet(x) \supset Calls(x))$$
- Regole di inferenza (Δ_2): Natural Deduction più la seguente regola derivata (MP):

$$\frac{A^x_t \quad \forall x.(A \supset B)}{B^x_t}\,(MD) \qquad := \qquad \dfrac{A^x_t \quad \dfrac{\forall x.(A \supset B)}{A^x_t \supset B^x_t}\,(\forall E)}{B^x_t}\,(\supset E).$$

Vogliamo dimostrare il seguente teorema "ground" φ (Pussy miagola e Tom abbaia):

$$Meows(Pussy) \ \wedge \ Barks(Tom).$$

1. **Astrazione**: Si applica f_Λ a φ, ottenendo:

$$f_\Lambda(\varphi) \quad = \quad Calls(Poppy) \wedge Calls(Poppy).$$

2. **Dimostrazione**: Si trova una dimostrazione di $f_\Lambda(\varphi)$ nello spazio astratto:

$$\Pi_2 \quad = \quad \cfrac{\cfrac{Pet(Poppy) \quad \forall x.(Pet(x) \supset Calls(x))}{Calls(Poppy)}\,(MP)}{Calls(Poppy) \wedge Calls(Poppy)}\,(\wedge I)$$

3. **Mapping Back**: Si genera "all'indietro" un *proof plan* (o meglio uno *schema di proof plan*) nello spazio ground:

$$\Pi_1' \quad = \quad \cfrac{\cfrac{P_1(u_1) \quad \forall x.(P_1(x) \supset P_2(x))}{P_2(u_1)}\,(?)}{P_2(u_1) \wedge P_2(u_1)}\,(\wedge I)$$

dove le meta-variabili "P_1, P_2, u_1" sono tali che:

$$P_2() \in \{Meows(), Barks()\},$$

$$P_1() \in \{Cat(), Dog()\}$$
$$u_1 \in \{Pussy, Tom\}.$$

Si noti che Π_1' non è una dimostrazione in quanto "(?)" non è una regola d'inferenza nel linguaggio ground.

4. **Rifinitura**: Si prende Π_1' come riferimento per costruire una dimostrazione "ground" di φ:

$$\cfrac{\cfrac{Cat(Pussy) \quad \cfrac{\forall x.(Cat(x) \supset Meows(x))}{Cat(Pussy) \supset Meows(Pussy)}\,(\forall E)}{Meows(Pussy)}\,(\supset E) \qquad \cfrac{Dog(Tom) \quad \cfrac{\forall x.(Dog(x) \supset Barks(x))}{Dog(Tom) \supset Barks(Tom)}\,(\forall E)}{Barks(Tom)}\,(\supset E)}{Meows(Pussy) \wedge Barks(Tom)}\,(\wedge I) \quad .$$

□

Si noti che nell'atto di rifinitura il dimostratore (theorem prover) ha compiuto 2 azioni:

- Poiché "MP" non è una regola d'inferenza valida nello spazio ground, ha introdotto un'applicazione "ponte" di $\forall E$ tra $\forall x.(P_1(x) \supset P_2(x))$ e $P_2(u_1)$. Ha così creato un sotto-albero Π_1'' che è una dimostrazione parametrica valida:

$$\Pi_1'' \quad = \quad \cfrac{P_1(u_1) \quad \cfrac{\forall x.(P_1(x) \supset P_2(x))}{P_1(u_1) \supset P_2(u_1)}\,(\forall E)}{P_2(u_1)}\,(\supset E).$$

- Ha utilizzato 2 volte il sottoalbero Π_1'', istanziando la terna $\langle P_1, P_2, u_1\rangle$ rispettivamente a $\langle Cat, Meows, Pussy\rangle$ e $\langle Dog, Barks, Tom\rangle$, per ottenere la dimostrazione finale.

L'esempio è banale, ma permette di evidenziare alcuni aspetti tipici dell'uso delle astrazioni PI.

La scelta della mappa e' dovuta all' osservazione che la distinzione tra le terne $\langle Cat, Meows, Pussy\rangle$ e $\langle Dog, Barks, Tom\rangle$ è ridondante ai fini della dimostrazione: scambiandone i contenuti negli assiomi e nel teorema da dimostrare, la rappresentazione del problema non cambia. Si sono quindi **collassati** tali simboli nella terna $\langle Pet, Calls, Poppy\rangle$. Il collasso di simboli, la cui distinzione è (o si ritiene) ridondante ai fini della dimostrazione, è una pratica comunissima nell'uso dell'astrazione [Hob85, Imi87, Ten87]. Essa comporta spesso il ri-utilizzo di sottoalberi comuni (nel nostro esempio Π_1''), che possono essere anche di grosse dimensioni, con evidente risparmio di lavoro. Un'altra pratica molto comune e' quella di introdurre nuove regole d'inferenza (o nuovi assiomi) nello spazio astratto quando si ritiene che esse possano essere usate frequentemente nella prova. Tali regole [assiomi], che possono essere anche molto complesse, vengono espanse [dimostrati] *una volta sola* a livello ground, con evidente risparmio di lavoro ridondante.

Nel nostro esempio l'applicazione dell'astrazione ha semplificato il lavoro del theorem prover. Infatti egli (o esso) ha dovuto effettuare complessivamente tre inferenze (due nello spazio astratto ed una a livello ground) contro le cinque della deduzione “diretta”. Nel nostro caso l'introduzione della regola derivata MP (il primo sillogismo aristotelico, o “*Modus Ponens*”), non ha ridotto il numero complessivo di inferenze, ma ha semplificato la dimostrazione astratta, permettendo così al dimostratore di concentrarsi su una visione più globale della prova. Inoltre, cosa non evidente immediatamente, la deduzione nello spazio astratto è avvenuta in uno **spazio di ricerca più ridotto** (a parte l'introduzione della nuova regola d'inferenza) : due predicati, una costante e due assiomi, contro quattro predicati, due costanti e quattro assiomi. Quest'aspetto è meno evidente, ma estremamente importante: possono esistere delle astrazioni tali che la prova astratta non sia di per sè più “semplice” di quella ground, bensì *più semplice da trovare*.

2.6 La nozione di “maggior semplicità”

L'atto di “astrarre” una rappresentazione di un problema è finalizzato alla costruzione di una diversa rappresentazione, la cui soluzione è ritenuta “più semplice”. Una completa ed esauriente trattazione formale del concetto di “maggior semplicità” non è stata mai data. Tentiamo di approfondire il problema presentando alcuni esempi.

2.6.1 Scelta di un'astrazione

Il primo aspetto da osservare è che *non esiste una nozione universale di “semplicità”* di una rappresentazione di un problema: essa è strettamente dipendente dal problema stesso. Conseguentemente, non esiste un criterio universale di scelta delle astrazioni da usare: si può solo tentare di creare una casistica dei problemi più comuni e di individuare i possibili approcci alla scelta, creando in tale modo una tipologia di astrazioni. Osserviamo che la scelta "*ad hoc*" dell'astrazione da usare costituisce un aspetto cruciale: una scelta non adatta può non limitarsi a "non semplificare" la soluzione, bensì può ulteriormente complicarla. Queste considerazioni sono riassunte nel seguente principio:

La nozione di "maggior semplicità" della rappresentazione di un problema e la conseguente scelta di un'astrazione adatta a tale semplificazione sono strettamente legate al problema stesso.

Un esempio può essere utile a chiarire tale concetto.

Esempio 7
Consideriamo due astrazioni diametralmente "opposte", mostrando come ciascuna possa semplificare la rappresentazione di un particolare tipo di problema.

- Supponiamo di ritenere che una certa regola di inferenza derivata IR (anche molto complessa) possa occorrere ripetutamente nella prova che cerchiamo. Possiamo pensare quindi di aggiungerla come nuova regola di inferenza nell'insieme Δ. Questo è ottenibile tramite un'astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ tale che Σ_2 sia generato da Σ_1 aggiungendo la regola richiesta ($\Delta_2 := \Delta_1 \cup \mathrm{IR}$), mantenendo inalterato il linguaggio ($f_\Lambda :=$ Identità). Il risultato è che il dimostratore avrà bisogno di "espandere" tale regola *una volta sola* nello spazio ground, ricavandone un sottoalbero di dimostrazione parametrico che andrà ad instanziare ripetuamente nella prova ground, risparmiando una grossa mole di lavoro. È immediato notare che tale astrazione è TI (nella fattispecie anche PI). Il costo di tale azione sta nel fatto che lo spazio di ricerca è stato ampliato (è stata aggiunta una regola d'inferenza). Se tale regola occorre frequentemente, il vantaggio ottenuto è notevole. Infatti, in generale vale [GW91]:

$$t = O(branch^{length}) \tag{2.1}$$

  dove t è il tempo impiegato a produrre una dimostrazione, *length* è la lunghezza di tale prova e *branch* è il *branching rate*, cioè la grandezza dello spazio di ricerca. Si vede quindi come un aumento del branching rate sia facilmente compensato da una diminunzione della lunghezza della prova.

- Supponiamo ora di ritenere inutile una regola di inferenza IR' (o più di una) ai fini della nostra deduzione. Per esempio, se riteniamo che la nostra prova sia ottenibile esclusivamente mediante inferenze proposizionali, possiamo eliminare le regole d'inferenza dei quantificatori [GG88]. Questa comporta una riduzione del branching rate, e di conseguenza del lavoro di ricerca. L'astrazione corrispondente è simile a quella dell'esempio precedente, con l'unica variante: $\Delta_2 := \Delta_1$ - IR'. Tale astrazione è TD, ma non è più TI.

□

2.6.2 L'idea di "Maggior semplicità" nelle astrazioni PI

Poiché il trattamento delle astrazioni PI costituisce la spina dorsale della teoria del "mapping back", la nozione di semplicità che esse descrivono merita una menzione specifica. L'idea nascosta dietro l'uso di tali mappe è quella secondo cui esse generano dimostrazioni *più corte* nello spazio astratto. Infatti la prova a livello astratto sussume (cioè vi è in qualche modo "contenuta") l'immagine della prova ground: $\Pi_2 \subseteq f_\Lambda(\Pi_1)$. Dalla formula 2.1 deduciamo che questa prerogativa può essere estremamente interessante dal punto di vista computazionale, in quanto la riduzione del valore dell'esponente può compensare sia un eventuale aumento del branching rate, sia il sovraccarico di lavoro dovuto all'atto di astrazione e, soprattutto, di mapping back e rifinitura. Non è un caso che le astrazioni di questa categoria siano state tra le più studiate in passato e che quindi incorrano assai frequentemente in letteratura [Giu].

Nonostante l'apparente ristrettezza della condizione di "PI-ness", sono molte le astrazioni che incorrono in questa categoria. Per questo motivo le forme in cui viene effettuata la "semplificazione" della dimostrazione astratta sono molteplici. Cerchiamo qui di catalogare le principali:

- **Collassamento di simboli**: gruppi di due o più simboli del linguaggio ground (predicati, funzioni o costanti individuali) posson venire "collassati" in uno unico se la distinzione tra di essi è ridondante ai fini della dimostrazione, cioè se sono interscambiabili (singolarmente o a gruppi) all'interno degli assiomi e della formula da dimostrare. Questo causa, in generale, una riduzione dello spazio di ricerca (e quindi del branching rate) e la possibilità di utilizzare più volte lo stesso sottoalbero, istanziandolo a gruppi di simboli diversi (si veda il semplice esempio di pag 36).

- **Aggiunta di una regola di inferenza o di un assioma**: l'aggiunta di una nuova regola d'inferenza IR (che si sa o si suppone essere derivabile dalle altre) può essere un ottimo metodo per accorciare notevolmente una dimostrazione, specialmente se si ritiene che la deduzione che essa rappresenta occorra ripetutamente all'interno della prova cercata. Un tipico esempio si ha quando IR rappresenta un teorema proposizionale, più o meno complesso. Un discorso analogo vale per l'aggiunta di ulteriori assiomi che si sanno (o si suppongono) essere derivabili dagli altri. È da notare come l'uso reiterato di tali astrazioni porti a costruire uno "schema a blocchi" della prova cercata, i cui singoli "blocchi" vengano dimostrati in un secondo momento, ed *una volta sola*. La formula di p. 36 mostra chiaramente come l'aumento del branching rate sia ampliamente compensato dalla riduzione della lunghezza della deduzione.

- **riduzione degli argomenti**: alcuni predicati $P(x_1, \ldots, x_n)$ di arità n posson venire mappati in predicati di arità minore k: $P(x_{i_1}, \ldots, x_{i_k})$, dove $x_{i_j} \in \{x_1, \ldots, x_n\}$. In questo modo la dipendenza dela formula atomica da un certo sottinsieme dei suoi argomenti è ritenuta ridondante e viene conseguentemente eliminata. La prova astratta risulterà quindi più semplice in quanto la presenza di un cerrto numero di termini verra' eliminata. Un discorso analogo vale per i simboli di funzione.

Per esempio [Mel87], supponiamo che un certo dispositivo elettronico combinatorio sia realizzato tramite una certa interconnessione di porte logiche, descritta da un predicato $Device\_impl(in_1, \ldots, in_n, out_1, \ldots, out_k, int_1, \ldots, int_m)$, $in_i, out_j, int_r \in \{0, 1\}$, dove in_i e out_j rapresentino i segnali logici di input ed output e int_r i segnali nei punti di interconnessione interni. Se astraiamo tale predicato in $Device\_spec(in_1, \ldots, in_n, out_1, \ldots, out_k)$, creiamo una rappresentazione del dispositivo *indipendente dalla sua realizzazione*: ogni sua occorrenza in un circuito verrà rappresentata come una scatola opaca della quale siano note solo le relazioni input-output. Si noti che questo processo di astrazione è effettivamente usato nella progettazione e verifica progettuale di circuiti digitali.

- **Mappe proposizionali**: un caso limite della situazione precedente si ha quando tutti gli argomenti di un predicato vengono eliminati. In questo caso una generica formula atomica viene mappata in una costante proposizionale. Applicando tale procedimento (cfr [Bor84], pp.??) a tutti i predicati, si viene a generare uno spazio proposizionale. La semplificazione introdotta in questo caso è molto forte: si è passati da una rappresentazione di un problema in uno contesto **semidecidibile** (una generica teoria del primo ordine) ad uno **decidibile** (una teoria proposizionale).

- **ABSTRIPS** [Sac74] merita un discorso a sè. Senza entrare nei dettagli formali, diciamo che in tale formalismo vengono rappresentate formule nella forma di implicazione tra una congiunzione di premesse (formule atomiche o loro negazioni) dette *precondizioni* ed una conseguenza. Un esempio può essere: "ho consultato l'orario, ho comprato il biglietto, ho disceso e risalito il sottopassaggio, quindi prendo il treno". Tali formule vengono astratte eliminando (cioè mappando in "True") le premesse che si ritengono essere più facili da verificare. Nell'esempio del treno possiamo eliminare la precondizione "ho disceso e risalito il sottopassaggio" in quanto riteniamo di riuscire facilmente a verificarla, e quindi possiamo preoccupercene in un secondo momento. La rappresentazione del problema viene quindi semplificata. Si noti come questa astrazione, e molte altre simili,siano effettivamente applicate (a livello inconscio) da un uomo che voglia prendere un treno.

2.7 Un modello computazionale semplificato per l'astrazione

Lo schema "a quattro fasi" di p. 30 mostra come l'idea di utilizzare forme di astrazione in theorem proving sia suggerita da una strategia del tipo "*divide et impera*": se la dimostrazione di un teorema risulta troppo onerosa per un theorem prover, è meglio suddividere il lavoro in due momenti: la dimostrazione di una versione "semplificata" del teorema (fase 2) e la successiva dimostrazione di una serie di brevi deduzioni "ponte" nella fase di rifinitura (fase 4). Si noti che il theorem prover agisce solo in questi due momenti, negli altri è il dispositivo di astrazione, mapping back ed unificazione (abstract proof checker) ad operare per semplificargli il lavoro.

Un modello computazionale esaustivo di tale procedimento è uno degli obbiettivi di futura estensione della Teoria dell'Astrazione, anche se una sua definizione presenterà sicuramente dei problemi non indifferenti di rappresentazione; non è infatti banale modellare una stima quantitativa del lavoro di unificazione (del secondo ordine) effettuato dal abstract proof checker in fase di rifinitura. Si noti che l'atto di astrazione può essere iterato successivamente, cioè la fase di ricerca di una prova nello spazio astratto può a sua volta avvenire per successive astrazioni. In [GW91] è stato presentato un modello estremamente semplificato per valutare la riduzione di lavoro a cui è soggetto il theorem prover. Tale modello, che non tiene conto del lavoro di unificazione sopra menzionato, considera quattro fattori:

1. il tempo impiegato per astrarre il problema;
2. il tempo impiegato a trovare una prova nello spazio astratto di livello maggiore;
3. il tempo impiegato a costruire i proof plan;
4. il tempo speso a dimostrare i teoremini "ponte" nell'atto di rifinitura del proof plan.

Per calcolare il costo computazionale complessivo di theorem proving (punti 2 e 4), considereremo solo tre parametri: il *branching rate* b, la lunghezza l della prova ed il tempo t per effettuare una singola inferenza. Assumeremo anche qualche strategia completa di ricerca, ad esempio "*breadth first*". Poiché dovremo esplorare $b + b^2 + ... + b^{l-1} = \sum_{i=1}^{l-1} b^i$ nodi, definiamo una **funzione costo** per il tempo necessario a trovare una prova:

$$c(b, l, t) = \sum_{i=1}^{l-1} b^i t = \frac{b}{b-1}(b^{l-1} - 1)t.$$

Risultati simili si ottengono con *branching rate* non costanti. Per considerare il costo computazionale nei diversi spazi di ricerca, indicizzeremo b, l e t con dei numeri di livello; per esempio t_0 rappresenterà il tempo per una singola inferenza a livello ground, b_1 il branching rate nel primo livello di astrazione e così via. Useremo τ_n per per rappresentare il tempo necessario a trovare la prova di un teorema usando n livelli di astrazione, e $\tau(m, n)$ per il teorema al livello m-esimo, usando i livelli da m ad n. Ovviamente $\tau_n = \tau(0, n)$. Definiamo quindi $\tau(m, n)$ tramite una ricorsione "all'indietro" (da $m = n$ a $m = 0$) che considera come caso base $\tau(n, n)$.

- Il caso base è semplicemente costituito dal tempo necessario a provare il teorema nello spazio più astratto, cioè $c(b_n, l_n, t_n)$.
- Il passo induttivo stabilisce il tempo necessario a provare il teorema a livello m-th dati il tempo necessario ad astrarre la formula ($\tau(m + 1, n)$), il tempo per trovare una prova a livello $m + 1$-th ($\tau(m + 1, n)$), il tempo per costruire il proof plan, cioè il tempo per "astrarre all'indietro" le l_{m+1} formule ($l_{m+1}u_m$), ed infine il tempo per rifinire tale proof plan.

Il tempo impiegato nella fase di rifinitura (considerando le approssimazioni sopra menzionate) dipende dalla lunghezza di ciascuno degli $(l_{m+1}-1)$ "salti" presenti nel proof plan. E' stato calcolato che $\tau(m,n)$ risulta minimo se il lavoro di rifinitura necessario è distribuito in modo uniforme nel proof plan. Per questo motivo assumeremo che la lunghezza di ciascuno degli$(l_{m+1}-1)$ "salti" sia una costante, g_m, dove $(g_m-1)(l_{m+1}-1)=l_m-1$. Il tempo per rifinire il proof plan risulta quindi essere uguale ad $(l_{m+1}-1)c(b_m,g_m,t_m)$. Assumeremo un tempo costante per mappare in avanti ed all' indietro le formule (a_m e u_m rispettivamente). Ignoreremo volontariamente i casi di **fallimento**, in cui le dimostrazioni astratte non sono mappabili all'indietro (ricordiamo che usiamo prevalentemente astrazioni TI). In questo senso il nostro modello è sicuramente *ottimista*.

A questo punto possiamo definire $\tau(m,n)$ come segue:

$$\begin{aligned}\tau(n,n) &= c(b_n,l_n,t_n)\\ \tau(m,n) &= a_m+\tau(m+1,n)+l_{m+1}u_m+(l_{m+1}-1)c(b_m,g_m,t_m).\end{aligned}$$

Possiamo confrontare ora il tempo τ_0 necessario a provare un teorema "direttamente", con quello impiegato utilizzando un livello di astrazione, τ_1:

- $\tau_0 \;=\; c(b_0,l_0,t_0)$
- $\tau_1 \;=\; a_0+c(b_1,l_1,t_1)+l_1u_1+(l_1-1)c(b_0,g_0,t_0)$.

τ_1 è costituito essenzialmente da due opposte funzioni costo: quella relativa alla dimostrazione del teorema nello spazio astratto e quella rappresentante il costo computazionale dell'operazione di rifinitura. Se consideriamo come parametro il rapporto $\frac{l_1}{l_0}$ (in un certo senso l'inverso della "forza" dell'astrazione) notiamo che, con la discesa di tale parametro da 1 a 0, $c(b_1,l_1,t_1)$ decresce ed invece cresce il costo della rifinitura: $(l_1-1)c(b_0,g_0,t_0)$, in quanto i teoremini "ponte" risultano più onerosi da provare. Come casi estremi, per $\frac{l_1}{l_0}=1$ (astrazione troppo "debole") τ_1 risulta maggiore τ_0, così come per $\frac{l_1}{l_0}=0$ (astrazione troppo "forte"). *In medio stat virtus*: con una scelta intermedia della "forza" dell'astrazione possiamo raggiungere un buon risparmio di tempo, come è evidenziato in figura 2.1[GW91] .

Alcune interessanti osservazioni sui benefici dell'astrazione posson esser aggiunte osservando il comportamento di alcune derivate parziali di τ_n:

- Considerando $\frac{\partial\tau_n}{\partial l_i}$, possiamo osservare come il massimo beneficio si ha con una ripartizione "uniforme" dei livelli di astrazione.
- Analizzando il comportamento di $\frac{\partial\tau_n}{\partial l_n}$, osserviamo come il comportamento ottimale si ottenga per un valore intermedio della "forza" dell'ultimo livello di astrazione, nè troppo debole (quando trovare una prova astratta risulti ancora troppo oneroso), nè troppo forte (quando l'azione di rifinitura risulti troppo pesante).
- L'andamento di $\frac{\partial\tau_n}{\partial n}$ ci suggerisce inoltre che, quando la ripartizione della forza tra i vari livelli di astrazione è uniforme, esiste un valore ottimo del numero dei livelli.

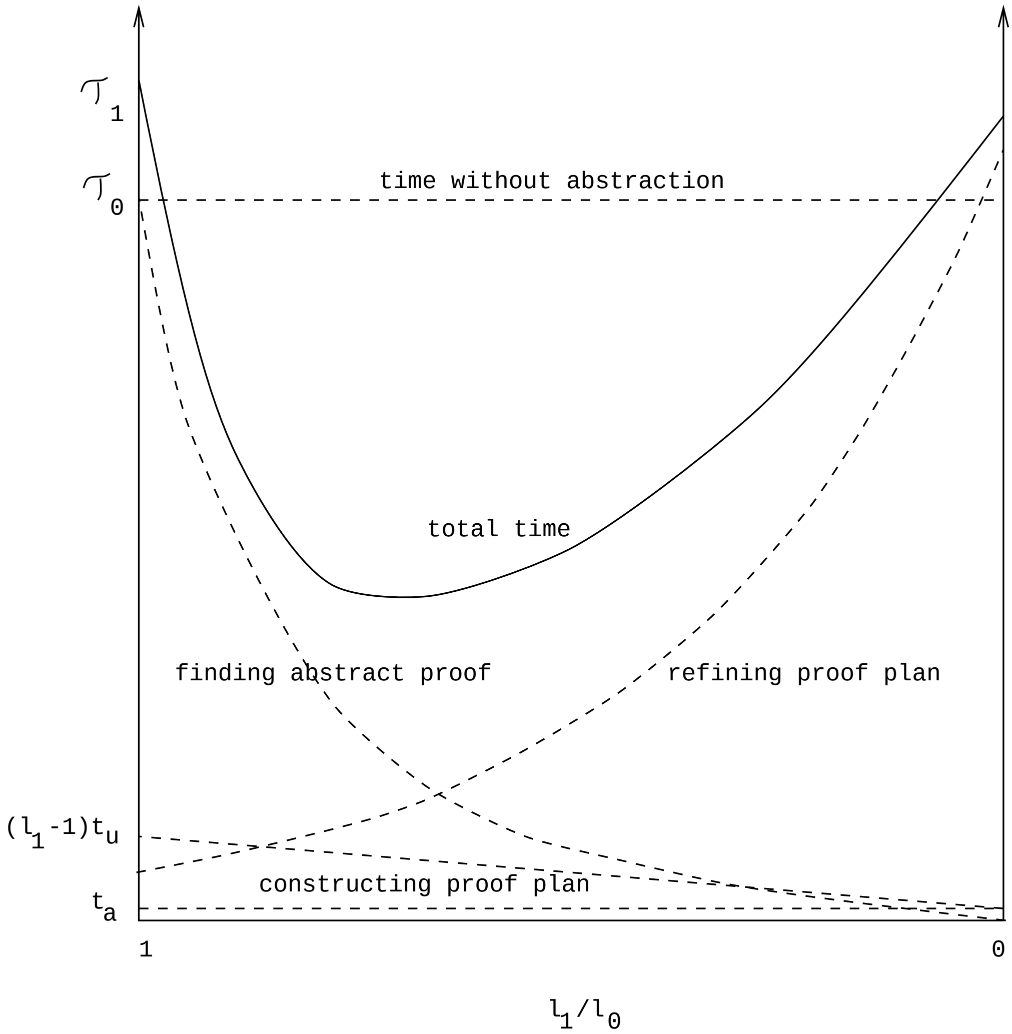


Figura 2.1: Confronto tra il tempo di theorem proving impiegando un livello di astrazione (τ_1) e quello relativo ad una ricerca diretta della prova(τ_0).

Questo modello è da considerarsi provvisorio, in quanto presenta due grossi limiti: non tiene conto nè degli eventuali fallimenti del mapping back nè dei tempi di unificazione (al secondo ordine) richiesti all' "abstract proof checker" nell'azione di rifinitura del proof plan. Pur con questi evidenti limiti, tale modello ha avuto alcune significative verifiche sperimentali [F.], che confermano la validità delle considerazioni precedenti.

2.8 Il problema dell'inconsistenza

Abbiamo osservato in precedenza come le astrazioni TI siano considerate le più importanti in theorem proving, in quanto garantiscono la dimostrabilità dell'immagine di ogni teorema "ground". Abbiamo inoltre introdotto l'idea della funzione "generativa" di $f : \Sigma_1 \Rightarrow \Sigma_2$, in cui $\Sigma_2 := f(\Sigma_1)$.

È stato ripetutamente osservato in passato [Nil80, Pla80, Ten87] come alcune astrazioni possano mappare sistemi formali consistenti in sistemi inconsistenti [6]. Un semplice esempio può chiarire il senso del problema:

Esempio 8
Supponiamo di avere un sistema formale consistente di tipo ABSTRIPS, che rappresenti una classificazione zoologica, in cui occorrano i seguenti assiomi:

$$\vdots$$
$$Vertebrato(Uomo)$$
$$\forall x.((Vertebrato(x) \land HaLaPelliccia(x)) \supset mammifero(x))$$
$$\forall x.((Vertebrato(x) \land FaLeUova(x)) \supset \neg mammifero(x))$$
$$\vdots$$

Supponiamo di applicare un'astrazione che si limiti ad eliminare le precondizioni $HaLaPelliccia()$ e $FaLeUova()$. È immediato notare come essa generi uno spazio inconsistente. □

Molti altri esempi concernenti questo fenomeno sono stati prodotti in letteratura. È utile ricordare, tra l'altro, che l' inconsistenza di una generica teoria del primo ordine è una proprietà **semidecidibile**.

Vi sono stati diversi tentativi di soluzione di questo problema, che però si sono sempre ricondotti a costruire categorie ristrette di astrazioni. In particolare Tenenberg [Ten87, Ten88] ha definito una categoria di astrazioni che evitano di generare inconsistenza. Tali astrazione non sono però TI. Una formulazione completa del problema è stata effettuata in [GW89b], dove si è mostrato come la generazione di uno spazio astratto inconsistente sia indissolubilmente legata all'uso delle astrazioni (strettamente) TI:

[6] Nei sistemi a refutazione (ad esempio Resolution) per "spazio inconsistente" si intende quello generato dagli assiomi *escludendo* la negazione della formula che si vuole dimostrare.

Teorema 2 *([Wal90] p.81). Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione TD (o NTD) tale che Σ_1 sia consistente e Σ_2 sia uno spazio con negazione* [7]*. Allora Σ_2 è consistente.*

Questo risultato mostra come la categoria delle astrazioni TD non presenti alcun problema di generazione di inconsistenza.

Teorema 3 *[GW89b] Sia $f^{(\Omega)} : \Sigma_1^{(\Omega)} \Rightarrow \Sigma_2^{(\Omega)}$ una famiglia di astrazioni TI parametrica in Ω, tale che:*

$$\Sigma_1^{(\Omega)} := \langle \Lambda, \Omega, \Delta \rangle$$
$$\Sigma_2^{(\Omega)} := \langle f_\Lambda(\Lambda), f_\Lambda(\Omega), \Delta \rangle,$$

e tale che f_Λ preservi la negazione, cioè $f_\Lambda(\neg\alpha) = \neg f_\Lambda(\alpha)$. Se esiste almeno un insieme consistente di assiomi Ω' tale che $f^{(\Omega')}$ sia **strettamente TI***, allora esiste un altro insieme consistente Ω'' tale che $f_\Lambda(\Omega'')$ sia inconsistente.*

Altre formulazioni simili di tale teorema si possono trovare in [GW93], in particolare esiste una formulazione duale per astrazioni NTI. Tale risultato è facilmente riassumibile: per ogni mappa TI f_Λ, esiste una scelta consistente di assiomi che rende inconsistente lo spazio astratto. Tale fenomeno ha una possibile interpretazione intuitiva : nell'atto di "buttare via dettagli" l'utente può sempre (inavvertitamente) eliminare quell'informazione che garantisce la consistenza di un sistema (ad esempio la distinzione tra due predicati che *sembrano* simili, ma che occorrono in modo leggermente diverso negli assiomi). Due esempi significativi si possono trovare in ([Wal90], p.88).

La generazione di uno spazio inconsistente non è necessariamente un problema: la struttura di una prova astratta potrebbe essere lo stesso utilizzabile per costruire una dimostrazione a livello ground se molti suoi rami non utilizzano l'inconsistenza. Il problema è intrinsecamente riconducibile a quello comune a tutte le astrazioni (strettamente) TI: data una prova di $f_\Lambda(\varphi)$, non è detto che ne esista una di φ. Questo vale per tutti i sottoalberi della prova astratta che abbiamo ottenuto.

In generale, la creazione di uno spazio astratto inconsistente è comunque indesiderabile. Oltre a varie considerazioni teoretiche, essa è sintomo di una scelta poco felice dell'astrazione usata. Riferendoci all'esempio 8, la precondizione $FaLeUova()$, sebbene sia indipendente da $Vertebrato()$, è determinante per affermare la conclusione $\neg Mammifero()$, e non può essere eliminata senza stravolgere il significato dell'assiomatizzazione.

Si noti che in questo caso, ed in molti altri, la semantica può essere molto di aiuto alla definizione della mappa: l'utente può facilmente accorgersi dell'incongruenza

[7] Σ è detto un *sistema con negazione* sse, per ogni formula α, vale che:

$$\neg\alpha \in \Lambda_1 \iff \alpha \in \Lambda_1,$$
$$\alpha \in NTH(\Sigma) \iff \neg\alpha \in TH(\Sigma)$$
$$\alpha \in TH(\Sigma) \iff \neg\alpha \in NTH(\Sigma).$$

osservando che esistono mammiferi vertebrati.

Da queste considerazioni emerge come, anche per quanto riguarda l'aspetto della possibile generazione di inconsistenza, valga un principio già espresso precedentemente: la scelta dell'astrazione è fondamentale per un suo uso effettivo ed efficace in theorem proving.

In [GW89b] è stata proposta una (parziale) soluzione al problema dell'inconsistenza dello spazio astratto, definendo una possibile metodologia nella scelta delle astrazioni da usare. Brevemente, si è definito un ordinamento (parziale) "$\sqsubseteq$" tra astrazioni aventi lo stesso spazio ground, tale che, se f genera uno spazio consistente e $g \sqsubseteq f$, allora anche g non genera inconsistenza. Se si costruisce quindi una catena ordinata $f_1 \sqsubseteq \ldots \sqsubseteq f_n$ di astrazioni tale che lo spazio astratto di f_n sia decidibile e si riveli consistente, allora anche tutte le altre astrazioni non generano inconsistenza. Tale metodologia è stata sperimentata tramite un simulatore prolog che è stato provato con successo su vari esempi.

Una trattazione più dettagliata dell'argomento esula dagli scopi di questa breve introduzione alla Teoria dell'Astrazione.

Capitolo 3

Estensione dell'astrazione a "sorted logics"

3.1 Introduzione

Descrivendo un generico problema tramite un sistema formale del primo ordine è spesso molto utile poter restringere il campo d'azione di una variabile ad un sottinsieme stretto del dominio. Per esempio, vorremmo poter dichiarare la variabile x come "Numero Naturale", in modo che l'istanziazione di x ad un numero non naturale

$$\frac{\forall x.P(x)}{P(1.5)}\ (\forall E)$$

non sia permessa. A questo fine è stato introdotto in passato, in varie forme diverse, il concetto di **sorte**. Intuitivamente, una sorte è un qualche particolare costrutto a cui è associabile una parte del dominio d'interpretazione (ad esempio: i numeri naturali).

Il sistema `GETFOL` utilizza una logica multisorte, detta "Sorted Natural Deduction" (SND) [CG90], che descriveremo brevemente nelle sezioni 3.2 e 3.3. In SND una sorte è rappresentata da un particolare predicato unario (es. $Naturale(.)$) in qualche modo "privilegiato", cioè tale da vincolare l'uso dei termini che lo soddisfano.

Nella Teoria dell'Astrazione sono stati considerati solo sistemi "semplici". Scopo di questo capitolo è quello di estendere tale teoria a sistemi multisorte, introducendo una definizione di astrazione estesa a tali sistemi e mostrando come questa estensione mantenga inalterate certe fondamentali proprietà delle astrazioni.

3.2 I sistemi formali multisorte

Il tradizionale punto di vista di molte "sorted logics" include un'estensione della nozione di modello in cui vi sono parecchi domini disgiunti, uno per ogni sorte.

Così, per esempio, nell'assiomatizzazione di spazi vettoriali sarebbe desiderabile avere un dominio di scalari ed uno di vettori. Questo sfortunatamente crea un effetto indesiderato: se si considerano, ad esempio, i reali come uno spazio vettoriale su sè stessi, si è costretti a concepire i reali scalari come disgiunti dai reali vettoriali. Si rende quindi necessaria l'introduzione di una funzione di pseudo-identità per identificare l'isomorfismo tra i due insiemi.

Tutto questo porta alla seguente idea. Non introduciamo domini disgiunti. Usiamo invece un solo dominio, ma permettiamo a un simbolo di predicato P di essere definito come **sorte**. Il risultato è equivalente a quello ottenuto aggiungendo l'assioma

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

alla nostra teoria. Dichiariamo quindi, in modo usuale, delle variabili il cui campo d'azione sia interno alla sorte P.

Introduciamo poi una relazione di pre-ordine [1] ,"*Più generale di*" ("$\succeq$"), sui simboli di sorte. Diremo che P *è più generale di* Q ($P \succeq Q$) se è vera la relazione:

$$\forall x.(Q(x) \supset P(x)),$$

cioè se ogni termine che soddisfa Q soddisfa anche P. Scriveremo talvolta $P \preceq Q$ al posto di $Q \succeq P$. L'asserzione $P \succeq Q$ è equivalente ad aggiungere il nuovo assioma

$$\forall x.(Q(x) \supset P(x))$$

alla teoria .

L'introduzione della relazione "$\succeq''$ ci permette di scrivere alcune interessanti relazioni, come ad esempio:

$$Reale \succeq Razionale \succeq Intero \succeq Naturale.$$

Cerchiamo ora di esplorare i modi in cui possiamo restringere i campi d'azione dei simboli individuali ed i domini delle funzioni. Per fare questo estendiamo l'esempio precedente.

Supponiamo che "+" sia un simbolo funzionale del nostro linguaggio, e vogliamo che esso denoti una funzioni da *coppie di reali* in *reali*. Questo è ottenibile associando "+" alla terna $\langle Reale, Reale, Reale\rangle$. Quest'informazione è equivalente all'introduzione dell'assioma:

$$\forall xy.((Reale(x) \wedge Reale(y)) \supset Reale(x+y)).$$

Se vogliamo ulteriormente estendere la nozione di "+" alla somma vettoriale, possiamo associare a tale simbolo la terna $\langle Vettore, Vettore, Vettore\rangle$. Si noti come la dichiarazione $Reale \succeq Naturale$ faccia sì che la somma tra due naturali sia un reale, che è equivalente all'introduzione della terna $\langle Naturale, Naturale, Reale\rangle$.

Anche alcune regole d'inferenza devono essere modificate per permettere l'utilizzo dell'informazione relativa alle sorti. Per esempio, l'introduzione della quantificazione universale $\forall I$ ("*forall introduction*") è possibile solo se la variabile introdotta spazia nell'ambito del dominio di appartenenza del parametro che si sostituisce. Tale nozione verrà illustrata meglio in seguito.

[1] Un pre-ordine è una relazione riflessiva e transitiva.

3.3 Sistemi formali SND

Introduciamo la definizione di sistemi formale basati su "Sorted Natural Deduction" (SND) [CG90], che cattura la nozione intuitiva di "sorted logic" descritta in precedenza.

Definizione 14 (SND) *Un sistema SND è una tripla $\langle \mathcal{L}, \mathcal{S}, \mathcal{D} \rangle$, dove $\mathcal{L}$ è un linguaggio per Natural Deduction, $\mathcal{S}$ è la* `SORT INFORMATION` *e $\mathcal{D}$ è l'insieme delle particolari regole d'inferenza di tali sistemi, detto "$\mathcal{P}-$deductive machinery".*

La `SORT INFORMATION` tiene conto di tutta l'informazione aggiuntiva necessaria alla gestione delle sorti all'interno di un sistema formale:

Definizione 15 (`SORT INFORMATION`) *La* `SORT INFORMATION` *è costituita da:*

- *un insieme finito e non vuoto di predicati unari $\mathcal{P}$* [2]*, tale che $U \in \mathcal{P}$, dove U è la* **sorte universale***;*
- *una relazione di preordine "$\succeq$" ("più generale di")* [3]*;*
- *una funzione $sort_v : \mathcal{I}_v \longmapsto \mathcal{P}$, che associ un simbolo di sorte ad ogni variabile individuale;*
- *una funzione $sort_p : \mathcal{I}_p \longmapsto \mathcal{P}$, che associ un simbolo di sorte ad ogni parametro individuale;*
- *una funzione $sort_c : \mathcal{I}_c \longmapsto \mathcal{P}$, che associ un simbolo di sorte ad ogni costante individuale;*
- *una funzione $sort_f : \mathcal{F} \longmapsto (\mathcal{P}^n \times \mathcal{P})^*$, che associ un insieme non vuoto di tuple $n+1$-arie di sorti ad ogni simbolo di funzione di generica arità n.*

Queste funzioni costituiscono uno strumento adeguato a descrivere l'informazione relativa alla sorte dei termini. Di conseguenza possiamo assumere che, per ogni simbolo do sorte P, esistano un'infinità numerabile di variabili v e parametri p tali che $sort_v(v) = P$ e $sort_p(p) = P$.

Definiamo ora in modo formale tutti i concetti necessari a comprendere come la `SORT INFORMATION` venga utilizzata all'interno di una generica deduzione.

---

[2] Negli articoli che questo capitolo riassume tale insieme viene indicato con il simbolo "Σ", ma in questo caso si creerebbe un'ambiguità, poiché nella Teoria dell'Astrazione il simbolo "Σ" è utilizzato per rappresentare sistemi formali.

[3] "$\succeq$" è riflessiva e transitiva; inoltre deve esistere un'elemento massimo U.

Definizione 16 (Sorte di un termine) *La sorte di un generico termine t può essere definita per induzione nel seguente modo:*

- *Una costante c è di sorte P sse $sort_c(c) \preceq P$.*
- *Un parametro p è di sorte P sse $sort_p(p) \preceq P$.*
- *Una variabile v è di sorte P sse $sort_v(v) \preceq P$.*
- *Sia $t = f(t_1, \ldots, t_n)$ un generico termine. t è di sorte P sse esiste una tupla $n+1$-aria $(P_1, \ldots, P_n, P_f) \in sort_f(f)$ tale che $P_f \preceq P$ e t_i sia di sorte P_i, per ogni indice $i \leq n$.*

Definizione 17 (assiomi SND) *L'insieme degli* **assiomi SND** *è scritto nella forma:*

$$\begin{array}{lll} \mathcal{A} = \{ & \forall x.U(x); & \\ & \forall x.P(x) & \mid \textit{for all } P \in \mathcal{P},\ sort_v(x) = P; \\ & \exists x.P(x) & \mid \textit{for all } P \in \mathcal{P},\ P \neq U,\ sort_v(x) = P; \} \end{array}$$

Definizione 18 ($\mathcal{P}$-deductive machinery) *L'insieme $\mathcal{D}$ delle regole d'inferenza di un SND è ottenuto sostituendo le usuali regole di Natural Deduction relative ai quantificatori con le regole seguenti* [4]*:*

$$(\mathcal{P}\forall E) \quad \begin{cases} \dfrac{\forall x.A}{A^x_t} & \textit{se } t \textit{ è di sorte } P_x \\[2ex] \dfrac{\forall x.A}{P_x(t) \supset A^x_t} & \textit{altrimenti} \end{cases}$$

$$(\mathcal{P}\forall I) \quad \frac{A}{\forall x.A^a_x} \quad \textit{applicabile sse } x \textit{ è di sorte } P_a$$

$$(\mathcal{P}\exists I) \quad \begin{cases} \dfrac{A^x_t}{\exists x.A} & \textit{se } t \textit{ è di sorte } P_x \\[2ex] \dfrac{A^x_t}{P_x(t) \supset \exists x.A} & \textit{altrimenti} \end{cases}$$

$$(\mathcal{P}\exists E) \quad \frac{\exists x.A \quad \begin{array}{c}[A^x_a] \\ B\end{array}}{B} \textit{applicabile sse } x \textit{ è di sorte } P_a.$$

"a" rappresenta qui un parametro tale che $sort_p(a) = P_a$, "x" una variabile tale che $sort_v(x) = P_x$ e t un generico termine.

[4]Le restrizioni delle usuali regole in ND sono ancora valide, ma qui non vengono specificate per evitare di appesantire la notazione.

3.4 Definizione di astrazione tra sistemi multisorte

Introduciamo subito la definizione che estende il concetto di astrazione a generici sistemi multisorte:

Definizione 19 (Astrazione Multisorte) *:*
Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ una quaterna $\langle \Sigma_1, \Sigma_2, f_\Lambda, f_s \rangle$, dove Σ_1 e Σ_2 sono due sistemi formali multisorte di qualsiasi tipo, $f_\Lambda : \Lambda_1 \longmapsto \Lambda_2$ è una mappa suriettiva tra i due linguaggi e $f_s : \mathcal{P}_1 \longmapsto \mathcal{P}_2$ è una mappa suriettiva tra i simboli di sorte. Se i seguenti vincoli sono verificati:

$$s_1 \succeq s_2 \Longrightarrow f_s(s_1) \succeq f_s(s_2) \tag{3.1}$$

$$t \textit{ is of sort } s \Longrightarrow f_\Lambda(t) \textit{ is of sort } f_s(s), \tag{3.2}$$

*allora f è detta un'***astrazione multisorte***.*

Se restringiamo il nostro ambito a sistemi Sorted Natural Deduction, abbiamo:

Definizione 20 (Astrazione SND) *:*
*Un'astrazione multisorte $f : \Sigma_1 \Rightarrow \Sigma_2$ in cui Σ_1 e Σ_2 sono due SND è detta un'***astrazione SND***.*

La proprietà 3.1 stabilisce che la mappa f_s deve preservare la relazione di maggior generalità tra simboli di sorte, mentre la proprietà 3.2 mostra come il comportamento di f_s dev'essere coerente con quello di f_Λ. Nessuna delle due relazioni implica l'altra.

Si noti come f_s non sia semplicemente una mappa tra simboli, bensì astragga tutta la struttura delle sorti di un sistema formale. In questo senso, non è semplicemente una mappa $\mathcal{P}_1 \longmapsto \mathcal{P}_2$, bensì $\mathcal{S}_1 \longmapsto \mathcal{S}_2$.

Le relazioni 3.1 e 3.2 valgono in una sola direzione. In generale, infatti, f_s e f_Λ possono "collassare" simboli, creando delle nuove relazioni. Ad esempio, se $s_1 \succeq s_2$, $s_3 \succeq s_4$ ed f_s collassa s_2 ed s_3 in un unico simbolo s, allora si ha che $f_s(s_1) \succeq f_s(s_4)$ senza che $s_1 \succeq s_4$. Possiamo introdurre un'ulteriore definizione:

Definizione 21 (Astrazione multisorte iniettiva) *Un'astrazione multisorte $f : \Sigma_1 \Rightarrow \Sigma_2$ è detta essere* **iniettiva** *sse f_Λ ed f_s sono iniettive e se le relazioni 3.1 e 3.2 valgono in entrambi i sensi:*

$$s_1 \succeq s_2 \Longleftrightarrow f_s(s_1) \succeq f_s(s_2)^5 \tag{3.3}$$

$$t \textit{ is of sort } s \Longleftrightarrow f_\Lambda(t) \textit{ is of sort } f_s(s). \tag{3.4}$$

Si noti come l'iniettività di f_Λ ed f_s non sia sufficiente per tale definizione. Infatti è richiesta l'iniettività di f_s intesa come funzione su tutta la struttura delle sorti e non solo sui simboli.

La definizione di astrazione SND iniettiva segue banalmente.

[5] Questa definizione è ridondante. Ad esempio $s_1 \succeq s_2 \Longleftarrow f_s(s_1) \succeq f_s(s_2)$ è ricavabile dalle relazioni 3.1 e 3.4. Si è preferito mantenere questa definizione per maggiore comprensibilità.

3.5 Proprietà delle astrazioni SND

Possiamo ora introdurre i risultati principali di questa sezione. In questo capitolo riporteremo anche le prove dei teoremi, anziché relegarle in un'apposita appendice, in quanto tali dimostrazioni sono **costruttive**, e la loro lettura risulta essere indispensabile alla comprensione della trattazione.

Teorema 4 *Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione SND tale che*

$$\begin{aligned} \Lambda_2 &= f_\Lambda(\Lambda_1) \\ \Omega_2 &= f_\Lambda(\Omega_1) \\ \Delta_2 = \Delta_1 &= \mathit{Regole\ d'inferenza\ SND} \\ f_\Lambda &= \mathit{identit\grave{a}} \end{aligned}$$

ed f_s una generica funzione sulle sorti, che verifichi le condizioni 3.1 e 3.2. Allora f è un'astrazione TI.

Prova:
Supponiamo di aver la prova Π_1 di un teorema ground φ. Mostriamo per induzione sulla lunghezza di Π_1 come costruire una prova Π_2 di $f_\Lambda(\varphi)$ nello spazio astratto.

- **(BASE)**: Se $\Pi_1 = \varphi$ è un'assioma, allora anche $f_\Lambda(\varphi)$ lo è.
- **(PASSO INDUTTIVO)**: Assumiamo

$$\Pi_1 \qquad = \qquad \frac{\begin{matrix} \vdots & & \vdots \\ \alpha_1 & \dots & \alpha_k \end{matrix}}{\varphi}\ (IR)$$

  dove IR rappresenta una generica regola d'inferenza SND. Per ipotesi induttiva possiamo costruire la dimostrazione delle formule astratte $f_\Lambda(\alpha_1)\dots f_\Lambda(\alpha_k)$. Mostriamo ora come costruire una deduzione di $f_\Lambda(\varphi)$ da $f_\Lambda(\alpha_1)\dots f_\Lambda(\alpha_k)$

$$\frac{\begin{matrix} f_\Lambda(\alpha_1) & \dots & f_\Lambda(\alpha_k) \\ & \vdots & \end{matrix}}{f_\Lambda(\varphi)}$$

  per ogni possibile regola IR.

  - **(CASO $\mathcal{P}\forall I$)**. IR è una "SND forall introduction".

$$\frac{\forall x.A}{A^x_a}\ (\mathcal{P}\forall I) \qquad \textit{dove } x \textit{ deve essere di sorte } P_a.$$

    Dall'equazione 3.2 possiamo dedurre che x (cioè $f_\Lambda(x)$) è di sorte $f_s(P_a)$, cosicché possiamo applicare la regola $\mathcal{P}\forall I$ nello spazio astratto per ottenere la deduzione richiesta:

$$\frac{\forall x.f_\Lambda(A)}{f_\Lambda(A^x_a)}\ (\mathcal{P}\forall I)$$

- **(CASO $\mathcal{P}\forall E$)**. IR è una "SND forall elimination". Sono possibili due sottocasi:
  1. IR è nella forma:
  $$\frac{\forall x.A}{A^x_t}\ (\mathcal{P}\forall E) \qquad \textit{dove } t \textit{ è di sorte } P_x.$$
  Dalla definizione 3.2 possiamo dedurre che t (cioè $f_\Lambda(t)$) è di sorte $f_s(P_x)$, cosicché possiamo applicare la regola $\mathcal{P}\forall E$ nello spazio astratto ed ottenere la deduzione richiesta:
  $$\frac{\forall x.f_\Lambda(A)}{f_\Lambda(A^x_t)}\ (\mathcal{P}\forall E)$$
  2. Alternativamente, IR può essere nella forma:
  $$\frac{\forall x.A}{P_x(t) \supset A^x_t}\ (\mathcal{P}\forall E) \qquad \textit{dove } t \textit{ non è di sorte } P_x.$$
  In questo caso non possiamo applicare la definizione 3.2. Tale relazione è valida in un senso solo, e noi non abbiamo introdotto alcuna ipotesi di iniettività di f. Per questo motivo dobbiamo introdurre due ulteriori sottocasi:
     * t **non è di sorte** $f_s(P_x)$. In tal caso
     $$\frac{\forall x.f_\Lambda(A)}{f_s(P_x)(t) \supset f_\Lambda(A^x_t)}\ (\mathcal{P}\forall E)$$
     è un'applicazione corretta di $\mathcal{P}\forall E$ in Σ_2, ed è l'inferenza richiesta.
     * t **è di sorte** $f_s(P_x)$. In tal caso non possiamo applicare direttamente la regola $\mathcal{P}\forall E$. Ciò nonostante, per soddisfare la richiesta, possiamo costruire la deduzione composta:
     $$\frac{\dfrac{\forall x.f_\Lambda(A)}{f_\Lambda(A^x_t)}\ (\mathcal{P}\forall E)}{f_s(P_x)(t) \supset f_\Lambda(A^x_t)}\ (\supset I).$$
- **(CASO $\mathcal{P}\exists E$)**. Assolutamente analogo a $\mathcal{P}\forall I$.
- **(CASO $\mathcal{P}\exists I$)**. Assolutamente analogo a $\mathcal{P}\forall E$.
- Se IR non è una regola quantificata, la deduzione richiesta risulta banalmente essere:
$$\frac{f_\Lambda(\alpha_1) \quad \dots \quad f_\Lambda(\alpha_k)}{f_\Lambda(\varphi)}\ (IR).$$

Q.E.D.

Il Teorema 4 fa esplicitamente riferimento a sistemi SND. Tale risultato è però generalizzabile ad ogni sistema multisorte Σ^* ad esso logicamente equivalente, cioè tale che

$$\vdash_{\Sigma} \varphi \Longleftrightarrow \vdash_{\Sigma^*} \varphi.$$

Infatti si può scrivere:

$$\vdash_{\Sigma_1^*} \varphi \Longrightarrow \vdash_{\Sigma_1} \varphi \Longrightarrow \vdash_{\Sigma_2} f_\Lambda(\varphi) \Longrightarrow \vdash_{\Sigma_2^*} f_\Lambda(\varphi).$$

Si noti come la dimostrazione del Teorema 4 sia **costruttiva**: non solo afferma che f è un'astrazione TI, ma insegna anche come costruire le prove nello spazio astratto.

Questo teorema dimostra come l'estensione di un'astrazione a simboli di sorte preservi la "truthfulness". Infatti, data un'astrazione SND $f : \Sigma_1 \Rightarrow \Sigma_2$, che supponiamo essere TI, possiamo "aggiungere" ad f una mappa sulle sorti f_s semplicemente componendola con un'astrazione che astragga solo i simboli di sorte (come nelle ipotesi del teorema precedente). L'astrazione risultante è ancora TI.

Possiamo inoltre osservare che, se la generica regola d'inferenza IR non è nè una $\mathcal{P}\forall E$ nè una $\mathcal{P}\exists I$, e se

$$\frac{\begin{matrix}\vdots & & \vdots \\ \alpha_1 & \dots & \alpha_k\end{matrix}}{\varphi}\ (IR)$$

è un'applicazione di IR nello spazio ground, allora

$$\frac{\begin{matrix}\vdots & & \vdots \\ f_\Lambda(\alpha_1) & \dots & f_\Lambda(\alpha_k)\end{matrix}}{f_\Lambda(\varphi)}\ (IR)$$

è a sua volta un'applicazione di IR nello spazio astratto.

Il punto critico della dimostrazione del Teorema 4 è quindi il comportamento delle regole $\mathcal{P}\forall E$ e $\mathcal{P}\exists I$: quando l'inverso della relazione 3.2 non è verificato, dobbiamo introdurre una tecnica alternativa per costruire la deduzione a livello astratto. Queste considerazioni suggeriscono un altro risultato:

Teorema 5 *Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione SND* **iniettiva** *tale che*

$$\Lambda_2 = f_\Lambda(\Lambda_1)$$

$$\Omega_2 = f_\Lambda(\Omega_1)$$

$$\Delta_2 = \Delta_1 = \textit{Regole d'inferenza SND}$$

$$f_\Lambda = \textit{identità}.$$

Allora f è un'astrazione "fortemente PC", cioè verifica la seguente proprietà:

$$\Pi_1 \textit{ è una dimostrazione di } \varphi \quad \Longleftrightarrow \quad f_\Lambda(\Pi_1) \textit{ è una dimostrazione di } f_\Lambda(\varphi).$$

Prova:

- ($\Rightarrow$) Osserviamo che, per l'iniettività di f, vale la relazione 3.4 (in entrambe le direzioni), e quindi non c'è più alcuna necessità di introdurre una "tecnica alternativa" per costruire le singole deduzioni nello spazio astratto in caso di $\mathcal{P}\forall E$ and $\mathcal{P}\exists I$. Per questo motivo, se

$$\frac{\alpha_1 \quad \dots \quad \alpha_k}{\varphi}\ (IR)$$

è un'applicazione di IR inΣ_1, allora

$$\frac{f_\Lambda(\alpha_1) \quad \dots \quad f_\Lambda(\alpha_k)}{f_\Lambda(\varphi)}\ (IR)$$

è un'applicazione corretta di IR in Σ_2, per ogni IR ($\mathcal{P}\forall E$ e $\mathcal{P}\exists I$ comprese). Per induzione sulla lunghezza di Π_1, ne segue che $f_\Lambda(\Pi_1)$ è una dimostrazione di $f_\Lambda(\varphi)$.

- ($\Leftarrow$) Poiché vale la relazione 3.4 (in entrambe le direzioni), e f_Λ è invertibile, possiamo ricavare che, se

$$\frac{f_\Lambda(\alpha_1) \quad \dots \quad f_\Lambda(\alpha_k)}{f_\Lambda(\varphi)}\ (IR)$$

è un'applicazione della regola IR in Σ_2, allora

$$\frac{\alpha_1 \quad \dots \quad \alpha_k}{\varphi}\ (IR) \qquad = \qquad \frac{f_\Lambda^{-1}(f_\Lambda(\alpha_1)) \quad \dots \quad f_\Lambda^{-1}(f_\Lambda(\alpha_k))}{f_\Lambda^{-1}(f_\Lambda(\varphi))}\ (IR)$$

è un'applicazione corretta di IR in Σ_1. Per induzione, come nel caso precedente, $\Pi_1 = f_\Lambda^{-1}(f_\Lambda(\Pi_1))$ risulta essere una dimostrazione di φ.

Q.E.D.

Questo risultato mostra come, se "aggiungiamo" ad f una mappa sulle sorti f_s nel modo visto, l'iniettività di f sia una condizione sufficiente per preservare non solo le proprietà di conservazione della deducibilità ("T*-ness"), ma anche quelle relative alla struttura delle prove (PI-ness). Infatti la composizione di f con un'astrazione fortemente PC preserva tutte queste proprietà di f.

3.6 Sistemi SND ed astrazioni PI

3.6.1 Il problema delle astrazioni PI tra sistemi multisorte

Ricordiamo la nozione di **astrazione PI**:

se Π_1 *è una dimostrazione di* φ, *allora esiste una prova* Π_2 *di* $f_\Lambda(\varphi)$ *tale che* $\Pi_2 \subseteq f_\Lambda(\Pi_1)$.

Ci potremmo chiedere se l'astrazione $f : \Sigma_1 \Rightarrow \Sigma_2$ dell'ipotesi del Teorema 4 sia non solo TI, ma anch PI. Sfortunatamente, questo non è vero. Infatti

$$\frac{\forall x. f_\Lambda(A)}{f_s(P_x)(t) \supset f_\Lambda(A^x_t)} \qquad \textit{dove } t \textit{ è di sorte } f_s(P_x)$$

non è un'applicazione **diretta** di $\mathcal{P}\forall E$. In tal caso dobbiamo introdurre un'ulteriore applicazione di $\supset I$:

$$\cfrac{\cfrac{\forall x. f_\Lambda(A)}{f_\Lambda(A^x_t)}\,(\mathcal{P}\forall E)}{f_s(P_x)(t) \supset f_\Lambda(A^x_t)}\,(\supset I) .$$

Considerazioni simili seguono banalmente per il caso $\mathcal{P}\exists I$.

Per questo motivo l'immagine della prova a livello ground *sussume strettamente* la prova astratta: $f_\Lambda(\Pi_1) \subset \Pi_2$. Dunque f non è PI.

Come abbiamo visto nei capitoli precedenti, la "*PI-ness*" è una proprietà essenziale per effettuare l'operazione di "*mapping back*" correttamente. Sarebbe fortemente auspicabile che ogni astrazione PI rimanesse tale anche dopo l'introduzione di mappe sui simboli di sorte. Nella prossima sezione descriveremo un modo per risolvere questo problema.

3.6.2 Una definizione equivalente di SND

Presentiamo una nozione equivalente di *SND deductive machinery*, per permettere di introdurre il mapping di simboli di sorte senza perdere la "PI-ness".

Definizione 22 (SND deductive machinery $\mathcal{D}$) *$\mathcal{D}$ è ottenuto sostituendo le regole "unsorted" dei quantificatori con le seguenti* [6]*.*

Quì di seguito "a" sarà un parametro tale che $sort_p(a) = P_a$, "x" una variabile tale che $sort_v(x) = P_x$ e "t" un generico termine.

$(\mathcal{P}\forall E)$	$\dfrac{\forall x.A}{A^x_t}$	*applicabile sse t è di sorte P_x*
$(\mathcal{P}\forall E')$	$\dfrac{\forall x.A}{P_x(t) \supset A^x_t}$	**sempre applicabile**
$(\mathcal{P}\forall I)$	$\dfrac{A^x_a}{\forall x.A}$	*applicabile sse x è di sorte P_a*
$(\mathcal{P}\exists I)$	$\dfrac{A^x_t}{\exists x.A}$	*applicabile sse t è di sorte P_x*
$(\mathcal{P}\exists I')$	$\dfrac{A^x_t}{P_x(t) \supset \exists x.A}$	**sempre applicabile**
$(\mathcal{P}\exists E)$	$\dfrac{\exists x.A \quad \begin{matrix}[A^x_a] \\ \vdots \\ B\end{matrix}}{B}$	*applicabile sse a è di sorte P_x.*

La definizione è ottenuta dalla Definizione 18 aggiungendo due (ridondanti) regole:

$$\frac{\forall x.A}{P_x(t) \supset A^x_t} \qquad \textit{se } t \textit{ è di sorte } P_x$$

$$\frac{A^x_t}{P_x(t) \supset \exists x.A} \qquad \textit{se } t \textit{ è di sorte } P_x$$

le quali possono essere banalmente derivate dalle regole SND originali tramite un'applicazione della regola "*imply introduction*" ($\supset I$), come precedentemente illustrato. Possiamo quindi introdurre il seguente risultato, che può essere visto come un'estensione del Teorema 4 alla definizione 22:

[6] Le restrizioni sulle usuali regole ND sono ancora valide, ma qui non vengono specificate per non appesantire la notazione.

Teorema 6 *Sia $f : \Sigma_1 \Rightarrow \Sigma_2$ un'astrazione SND tale che*

$$\Lambda_2 = f_\Lambda(\Lambda_1)$$

$$\Omega_2 = f_\Lambda(\Omega_1)$$

$$\Delta_2 = \Delta_1$$

le regole d'inferenza SND come nella definizione 22,

$$f_\Lambda = identit\grave{a}.$$

Allora f è un'astrazione PI.

Prova:

Dalla definizione 22 e dalla relazione 3.2 segue che, se

$$\frac{\alpha_1 \quad \ldots \quad \alpha_k}{\varphi}\ (IR)$$

è un'applicazione della regola generica IR ($\mathcal{P}\forall E$, $\mathcal{P}\forall E'$, $\mathcal{P}\exists I$ e $\mathcal{P}\exists I'$ comprese) nello spazio ground, allora

$$\frac{f_\Lambda(\alpha_1) \quad \ldots \quad f_\Lambda(\alpha_k)}{f_\Lambda(\varphi)}\ (IR)$$

è una corretta applicazione della stessa regola a livello astratto. Segue per induzione sulla lunghezza di Π_1 che, se Π_1 è una dimostrazione di φ, allora $f_\Lambda(\Pi_1)$ è una prova di φ. **Q.E.D.**

Poiché la composizione di due astrazioni PI è ancora PI, possiamo "aggiungere" una mappa sui simboli di sorte, f_s, ad un'astrazione SND $f : \Sigma_1 \Rightarrow \Sigma_2$ (cioè possiamo comporre f con un'astrazione simile a quella del Teorema 6) senza che essa perda la proprietà di essere PI. Possiamo quindi affermare che *astrarre simboli di sorte preserva la "PI-ness"*.

Il Teorema 6 non è in generale estensibile ad ogni tipo di sistema multisorte, in quanto riguarda la struttura delle prove. Si può però pensare che per molti sistemi multisorte equivalenti esistano risultati analoghi.

Capitolo 4

Definizione di una funzione tra linguaggi

4.1 Introduzione

Secondo la definizione formale introdotta nel Capitolo 2, un'astrazione è costituita da una coppia di sistemi formali del primo ordine, detti *contesti*, e da una funzione suriettiva, computabile e totale tra i relativi linguaggi. Ne consegue che la definizione di astrazione si riconduce sostanzialmente a quella della funzione f_Λ [1].

A questo punto sorge un problema, poiché, contrariamente alla nozione semplificativa introdotta nella teoria, per poter meccanizzare un'astrazione si deve considerare un linguaggio come una tripla $\langle alfabeto, insieme\ dei\ termini, insieme\ delle\ formule \rangle$, e non come un semplice insieme di espressioni. Si rende quindi necessario introdurre una nozione *effetiva* di "*funzione tra linguaggi del primo ordine*", f_Λ.

In questo capitolo descriveremo, in primo luogo, le proprietà richieste alla definizione "effettiva" di una generica mappa f_Λ. Successivamente, introdurremo un formalismo che descriva tali funzioni e che soddisfi le proprietà menzionate.
Il risultato di tale lavoro, essenzialmente teorico, consiste nella formulazione di un **linguaggio** con cui definire astrazioni, e in un **algoritmo** che descrive un interprete per tale linguaggio. Queste nozioni costituiscono i fondamenti teorici per la realizzazione dell'astrazione in `GETFOL`, che verrà illustrata nei capitoli successivi.

4.2 Requisiti di f_Λ

Nella teoria dell'astrazione un *linguaggio del primo ordine* Λ è implicitamente considerato come l'insieme dei termini e delle formule[2] generati da un'insieme di simboli (alfabeto) mediante le usuali regole di buona formatezza della logica del primo ordine. I simboli dell'alfabeto stesso non incorrono quindi in tale definizione: ad esempio, un

[1] In questo modo è giustificato l'uso frequente della parola "*mappa*" come sinonimo di "*astrazione*".
[2] Useremo talvolta l'acronimo inglese "*wff*" (Well Formed Formula) come sinonimo di *formula*.

singolo predicato P non è considerato appartenere a Λ, in quanto non è nè un termine nè una formula.

In questa trattazione raffineremo la nozione di linguaggio Λ, includendo anche l'*alfabeto* $\mathcal{A}$ di Λ: f_Λ non si limita a mappare formule e termini, ma anche singoli simboli (proposizionali, predicativi, funzionali ed individuali). Tale distinzione è piú significativa di quanto possa sembrare a prima vista: si noti, ad esempio, che una costante individuale a (elemento dell'alfabeto) viene "vista" da f_Λ in modo distinto dal termine corrispondente a, e come tale potrebbe venir mappata in modo diverso. In questo senso, possiamo pensare ad f_Λ come ad una funzione polimorfa.

Nel paragrafo 2.2.1 abbiamo introdotto il significato di funzionalità **generativa** di un'astrazione: tra le altre cose, abbiamo prospettato la possibilità di generare automaticamente l'alfabeto astratto: $\mathcal{A}_2 := f_\Lambda(\mathcal{A}_1)$. Questo è possibile in base alla nozione "estesa" del linguaggio appena introdotta.

Vogliamo quindi, come primo requisito per la definizione di f_Λ, che essa realizzi una **doppia funzionalità**:

- **Astrazione dell'alfabeto**: $\mathcal{A}_2 := f_\Lambda(\mathcal{A}_1)$. In questo caso l'astrazione ha una funzionalità *generativa*, in quanto l'alfabeto del contesto astratto viene costruito come immagine di quello del contesto ground. Operativamente, dato un generico simbolo s, il meccanismo di mapping crea il simbolo $f_\Lambda(s)$ e lo **dichiara** come appartenente allo spazio astratto.

- **Astrazione delle espressioni**: f_Λ crea una relazione funzionale tra espressioni (wff e termini) ground e espressioni astratte: $\varphi_2 = f_\Lambda(\varphi_1)$. *Tutti i simboli occorrenti in* φ_2 *devono già essere presenti* Λ_2. In questo senso, φ_2 deve poter essere ottenuta dall'alfabeto $\mathcal{A}_2$ mediante le regole di buona formatezza.

In secondo luogo, richiediamo che la formulazione della nostra funzione sia **il piú possibile generale**. Come abbiamo accennato nel Capitolo 2, la definizione formale data in [Giu] ha il pregio di catturare la maggior parte delle astrazioni definite in letteratura e di essere piú generale di ogni precedente tentativo di definizione. La conservazione di tale generalità è quindi da considerarsi come un requisito primario.

Esistono molti formalismi possibili con i quali si può definire una funzione effettivamente computabile tra due linguaggi. Potremmo pensare di costruirla mediante un programma in qualche linguaggio procedurale, che utilizzi strutture di controllo del tipo *if-then-else*, *while*, *for* ecc. (es. Pascal, C); potremmo scrivere in alternativa un programma in Prolog, o un insieme di funzioni in Lisp; potremmo anche definire un nuovo linguaggio di programmazione "ad hoc" (procedurale, dichiarativo, funzionale...). Noi vogliamo però che un'astrazione possa essere definita da un utente *eventualmente digiuno di nozioni di informatica*: l'unica richiesta è una conoscenza, per lo meno intuitiva, della logica dei predicati del primo ordine. Vogliamo inoltre che la dichiarazione di una astrazione sia *leggibile*, cioè tale che la sua funzionalità sia comprensibile dalla sua lettura.

Per questo motivo richiediamo alla definizione di un'astrazione di essere **semplice** ed **intuitiva**.

Riassumendo, possiamo elencare i requisiti fondamentali di una possibile funzione tra linguaggi f_Λ:

(i) **Doppia funzionalità**, cioè possibilità di generare l'alfabeto astratto e mapping delle espressioni.

(ii) **Generalità**, cioè capacità di catturare pienamente il concetto intuitivo di `astrazione` .

(iii) **Semplicità** ed **intuitività** nella dichiarazione.

4.3 Una prima descrizione informale di f_Λ

La definizione effettiva di una funzione tra i linguaggi rappresenta un concetto cruciale per questa trattazione. Per questo motivo presentiamo prima una nozione simile, piú informale, che aiuti a comprendere le motivazioni delle scelte effettuate. Tale nozione viene sviluppata progressivamente, seguendo una serie di idee intuitive.

L'idea essenziale che vogliamo introdurre sta nella distinzione tra mapping dei **simboli dell'alfabeto** e quello della **struttura logica delle espressioni**.

Possiamo rappresentare la funzione tra i due *alfabeti* mediante delle semplici **coppie di simboli**, ad esempio:

$$\begin{array}{lll} cucchiaio(.) & \longmapsto & posata(.) \quad [predicati] \\ madre(.) & \longmapsto & genitore(.) \quad [funzioni] \\ Pussy & \longmapsto & MioGatto \quad [individui], \end{array}$$

con l'ovvia restrizione che non possano occorrere due coppie con lo stesso simbolo nella parte sinistra.

Per quanto riguarda il mapping della *struttura di termini e wff*, possiamo pensare di utilizzare delle **regole di trascrittura** per le espressioni. Tali regole potrebbero essere descritte in molte forme diverse, come accennato in precedenza (programmi pascal, funzioni lisp, ecc.). Una prima rappresentazione informale, semplice ed intuitiva, è quella basata su una formulazione ricorsiva, come ad esempio:

$$\begin{array}{llll} f_\Lambda(\alpha \wedge \beta) & := & f_\Lambda(\alpha) \wedge f_\Lambda(\beta) & [conserva\ la\ congiunzione] \\ f_\Lambda(\neg\neg\alpha) & := & f_\Lambda(\alpha) & [semplifica] \\ f_\Lambda(\alpha \supset \beta) & := & \neg f_\Lambda(\alpha) \vee f_\Lambda(\beta) & [elimina\ l'implicazione] \\ f_\Lambda((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & := & f_\Lambda(\alpha) \wedge f_\Lambda(\beta \vee \gamma) & [distribuisce] \\ & & & \\ f_\Lambda(Implem(\underline{in}, \underline{out}, \underline{int})) & := & Spec(f_\Lambda(\underline{in}), f_\Lambda(\underline{out})) & [riduce\ gli\ argomenti] \\ f_\Lambda(MoglieDi(x)) & := & DonnaSposata & [elimina\ l'argomento] \\ f_\Lambda(Intero(x) \wedge Positivo(x)) & := & Intero(f_\Lambda(x)) & [elimina\ un\ congiungente] \end{array}$$

dove x, in, out, int, e α, β, γ sono delle meta-variabili, istanziabili rispettivamente in termini e formule.

Risulta evidente la ricorsività di tali espressioni: in generale le metavariabili rappresentano dei termini/formule ground (anche molto complesse), che devono a loro volta venire mappate. Ad esempio, la quarta regola dell'esempio di p.71 potrebbe essere ulteriormente applicata a $\beta \vee \gamma$ se β fosse scritto come $\alpha' \wedge \beta'$ e γ come $\alpha' \wedge \gamma'$.

Possiamo fare in modo tale che in una regola non occorrano mai simboli dello spazio astratto. È sufficiente scrivere tali simboli nella forma f_Λ(simbolo ground). Per esempio, la quinta regola dell'esempio di p.71 può essere espressa nella forma:

$$f_\Lambda(Implem(\underline{in}, \underline{out}, \underline{int})) := [f_\Lambda(Implem)](f_\Lambda(\underline{in}), f_\Lambda(\underline{out})), \tag{4.1}$$

purché esista già la mappa:

$$Implem(.,.,.) \longmapsto Spec(.,.).$$

Si noti come in questo modo si renda necessaria la possibilità di scrivere delle mappe "ibride" del tipo

$$funzione \longmapsto costante\ individuale$$

$$predicato \longmapsto costante\ proposizionale.$$

La cosa non deve stupire, in quanto, secondo la logica del primo ordine, possiamo pensare ad una costante individuale [proposizionale] come ad una funzione [predicato] di arità zero. Per esempio, la penultima regola dell'esempio di p.71 può essere riscritta tramite una mappa $funzione \longmapsto costante\ individuale$:

$$MoglieDi(.) \longmapsto Donnasposata \tag{4.2}$$

ed una regola:

$$f_\Lambda(MoglieDi(x)) := f_\Lambda(MoglieDi). \tag{4.3}$$

In quest' ultima assegnazione risulta evidente l'ambiguità nell'uso del simbolo f_Λ: a sinistra rappresenta l'astrazione di un'*termine* ($MoglieDi(x)$), mentre sulla destra il mapping di un *simbolo di funzione* ($MoglieDi(.)$). Per questo motivo nella definizione formale sono stati introdotti due tipi di "chiamata" ricorsiva a $f_\Lambda(.)$: "ABS(.)", che rappresenta l'astrazione di espressioni, e "MAP[.]", che chiama la mappa dei simboli. In questo modo, l'assegnazione 4.3 può essere riscritta, come sempre in forma informale:

$$ABS(MoglieDi(x)) := MAP[MoglieDi] \tag{4.4}$$

$$ABS(Implem(\underline{in}, \underline{out}, \underline{int})) := MAP[Implem](f_\Lambda(\underline{in}), f_\Lambda(\underline{out})), \tag{4.5}$$

In questo modo, se voglio astrarre i termini $MoglieDi(Mario)$ e $MoglieDi(Piero)$, l'applicazione in sequenza della regola 4.4 e della mappa 4.2 genera in entrambi i casi il termine $DonnaSposata$.

4.4 Un formalismo per f_Λ

4.4.1 Un esempio introduttivo

Proponiamo ora un semplice esempio di dichiarazione ed uso di un'astrazione in GETFOL [3], in modo che la nozione di f_Λ, che descriveremo nella prossima sezione, possa risultare più comprensibile. La descrizione dettagliata dei comandi qui utilizzati verra' esposta nei prossimi capitoli.

Esempio 9

```
World:: ABSTRACTION DECLARE  FirstExample
    GROUND World
    ABSTRACT AbstractWorld
    FLAMBDA
        MAPS
            SMAPS
            PSMAPS
            PMAPS:
                Maestro -> Insegnante
            FUMAPS:
                padre -> genitore
            FIMAPS
            IMAPS:
                Paolo_Bianchi -> Paolo
        RULES
            (forall $x. $A) -> (forall IMAP[$x]. WABS($A))
            (exists $x. $A) -> (exists IMAP[$x]. WABS($A))
            ($A and $B) -> (WABS($A) and WABS($B))
            ($A or $B)  -> (WABS($A) or WABS($B))
            ($A imp $B) -> (WABS($A) imp WABS($B))
            ($A iff $B) -> (WABS($A) iff WABS($B))
            (not $A )   -> (not WABS($A))
            $P1($x)  ->  PMAP[$P1](TABS($x))
            $f1($x)  ->  FUMAP[$f1](TABS($x))
            $x   -> IMAP[$x];

World:: ABSTRACT WFF Maestro(padre(Mario)) and Operaio(Paolo_Bianchi)
         BY FirstExample;
You are now in the abstract context 'AbstractWorld'.
Insegnante(genitore(Mario)) and Operaio(Paolo)
is the mapped wff from ground context 'World'.

AbstractWorld::
```

[3] In questo esempio le parole chiave vengono scritte in caratteri maiuscoli. "`World::` " ed "`AbstractWorld::` " sono i prompt del sistema (in GETFOL il prompt è dichiarabile da utente, ed in questo caso è stato definito con il nome del contesto corrente).

$\square$

I simboli "$" costituiscono le metavariabili, istanziabili a simboli, termini e formule del linguaggio ground. Gli operatori WABS e TABS (**W***ff* **ABS***tract* e **T**erm **ABS**tract) rappresentano le chiamate ricorsive a f_Λ, mentre gli operatori di tipo ...MAP richiamano le corrispondenti funzioni dell'alfabeto.

Come avevamo anticipato, f_Λ è ripartita in *mappe tra simboli* ("map" [4]) e *regole di trascrittura* delle espressioni logiche ("rule").

L'interpretazione delle map è intuitiva: il predicato "$Maestro(.)$" viene mappato in "$Insegnante(.)$", la funzione "$padre(.)$" in "$genitore(.)$", la costante individuale "$Paolo\_Bianchi$" in "$Paolo$". Ogni simbolo non citato esplicitamente, come vedremo in seguito, viene mappato per default in una copia di sè stesso.

Anche l'interpretazione delle regole è semplice. Ad esempio

```
($A and $B) -> (WABS($A) and WABS($B))
```

è interpretabile come: $f_\Lambda(\alpha \wedge \beta) \quad := \quad f_\Lambda(\alpha) \wedge f_\Lambda(\beta)$, mentre

```
$P1($x)  ->  PMAP[$P1](TABS($x))
```

può essere visto come: $f_\Lambda(P(x)) \quad := \quad [f_\Lambda(P)](f_\Lambda(x))$.

Applicando mentalmente le regole e le mappe dell'esempio, si può verificare come la formula:

```
Maestro(padre(Mario)) and Operaio(Paolo_Bianchi)
```

venga astratta in:

```
Insegnante(genitore(Mario)) and Operaio(Paolo).
```

L'astrazione risultante è in questo caso *teoria-invariante*, in quanto le rule mantengono inalterata la struttura logica delle espressioni.

[4]Si preferisce mantenere i nomi inglesi per "*map*" e "*rule*", in quanto l'uso dei corrispondenti termini italiani "*mappa*" e "*regola*" possono generare confusione rispettivamente con le funzioni tra linguaggi e con le regole d'inferenza.

4.4.2 Le "map"

Una **map** è definita da una coppia

$$simbolo\ ground \longmapsto simbolo\ astratto\ [eventuale\ informazione\ supplementare].$$

Essa rappresenta un'istanza della funzione tra simboli dell'alfabeto. Nella dichiarazione di f_Λ ne esistono di sei tipi diversi:

- SMAP (**S**_entence_ **MAP**), tra simboli proposizionali;
- PSMAP (**P**_redicate-to-_**S**_entence_ **MAP**), tra un predicato ed un simbolo proposizionale;
- PMAP (**P**_redicate_ **MAP**), tra predicati;
- FUMAP (**FU**_nction_ **MAP**), tra funzioni;
- FIMAP (**F**_unction-to-_**I**_ndividual_ **MAP**), tra una funzione ed un simbolo individuale;
- IMAP (**I**_ndividual_ **MAP**), tra simboli individuali.

Possiamo riportare alcuni semplici esempi:

```
smaps:
        A -> B
psmaps:
        P -> A
pmaps:
        Forchetta   -> Posata
        Sollevabile -> Movibile
        DeviceImpl  -> DeviceSpec arity 2
fumaps:
        add   -> +         infix
        Padre -> Genitore
fimaps:
        MoglieDi -> DonnaSposata
imaps:
        MarioRossi -> Mario
```

Nell'interpretazione delle "map" esistono alcune scelte di default:

- Se non ulteriormente specificato da qualche informazione aggiuntiva, il simbolo di destra possiede le stesse caratteristiche di quello di sinistra. Per esempio "$Posata(.)$" è un predicato di arità uno perché "$Forchetta(.)$" lo è.
- Se un simbolo s non occorre nella parte sinistra di alcuna map, si considera un'implicita mappa $s \longmapsto s$, cioè viene mappato in una copia di sè stesso (identità di default).

La definizione delle mappe sull'alfabeto è inoltre soggetta a due vincoli:

- Non devono esistere due distinte map con la stessa parte sinistra.
- Se due simboli ground vengono mappati nello stesso simbolo astratto, entrambe le occorrenze di quest'ultimo devono avere le stesse caratteristiche (arità ecc.).

Ad esempio, la sequenza:

```
pmaps:
     P1 -> S
     Q2 -> S,
```

dove $P1$ e $Q2$ hanno caratteristiche diverse (ad es. arità 1 e 2 rispettivamente), non è accettabile, in quanto il predicato astratto S non può avere arità 1 e 2 contemporaneamente. Nell'implementazione in `GETFOL` è stato introdotto un test, in fase di dichiarazione di un'astrazione, che genera un messaggio di errore se tali vincoli vengono disattesi.

La tecnica di dichiarazione delle mappe sull'alfabeto risulta essere semplice anche per un utente inesperto. L'introduzione delle scelte di default sopra citate riduce notevolmente il tempo necessario a tale dichiarazione, in quanto è necessario definire solo le mappe che introducono delle *differenze* tra $\mathcal{A}_1$ ed $\mathcal{A}_2$. Un'ulteriore semplificazione si ha con l'introduzione dei prototipi, come vedremo in seguito.

4.4.3 Le "rule"

La definizione formale di una regola di trascrittura (rule) si basa su alcuni particolari oggetti sintattici: le **abstract-expression**, considerabili come delle particolari espressioni di ordine superiore.

Siano Λ_1 e Λ_2 due linguaggi del primo ordine tali che $\Lambda_i = \langle \mathcal{A}_i, \mathcal{T}_i, \mathcal{W}_i \rangle$, dove $\mathcal{A}_i$, $\mathcal{T}_i$ e $\mathcal{W}_i$ sono rispettivamente l'alfabeto, l'insieme dei termini e l'insieme delle formule di Λ_i. Definiamo innanzitutto alcuni operatori:

- WABS e TABS (**W***ff* **ABS***tract* e **T***erm* **ABS***tract*), che definiscono le funzioni:

  $$WABS : \mathcal{W}_1 \longmapsto \mathcal{W}_2 \qquad TABS : \mathcal{T}_1 \longmapsto \mathcal{T}_2$$

  e rappresentano la chiamata ricorsiva al meccanismo di astrazione delle espressioni.

- ...MAP (SMAP, PSMAP, ecc.) che definiscono le mappe:

  $$...MAP : \mathcal{A}_1 \longmapsto \mathcal{A}_2$$

  e rappresentano l'astrazione dell'alfabeto.

Possiamo ora introdurre una serie di definizioni:

(i) Un **Mapped Symbol**[5] è scritto nella forma $...MAP[simbolo\ ground]$. Alcuni esempi possono essere:

$$PMAP[Cucchiaio],\ IMAP[MarioRossi],\ FIMAP[MoglieDi],$$

dove $Cucchiaio(.)$ è un predicato, $MarioRossi$ una costante individuale e $MoglieDi(.)$ una funzione che vogliamo mappare in una costante individuale. Ciascuno di essi rappresenta un simbolo dell'alfabeto astratto.

(ii) Un **Abstract Term** o è nella forma $TABS(termine\ ground)$, oppure è costruito secondo le usuali regole di ben formatezza della logica da *Mapped Symbol* e/o altri *Abstract Term*. Anche qui riportiamo un paio di esempi:

$$FUMAP[Padre](IMAP[MarioRossi]) \qquad FUMAP[\$f](TABS(\$x)).$$

(iii) analogamente un' **Abstract Wff** o è nella forma $WABS(formula\ ground)$ oppure è costruito da *Mapped Symbol*, *Abstract Term* ed altre *Abstract Wff*:

$$forall\ IMAP[\$x].\ WABS(\$A) \qquad WABS(\$A)\ \mathit{or}\ WABS(\$B)$$

$$PMAP[\$P2](TABS(\$x),TABS(\$y)) \quad FIMAP[MoglieDi] \quad IMAP[\$x].$$

(iv) Un **Abstract Expression** (cioè un Abstract Term o un'Abstract Wff) è quindi un'espressione di ordine superiore atta a "guidare" il meccanismo di astrazione. A tutti gli effetti essa rappresenta un'espressione di Λ_2. Si noti come nessun simbolo appartenente all'alfabeto astratto possa occorrere all'interno di un'*Abstract Expression*. Tali espressioni costituiscono le parti destre delle delle regole di trascrizione (rule):

(v) Una **Rule** (regola di trascrittura) è costituita da una coppia

$$termine \longmapsto Abstract\ Term \qquad \text{oppure} \qquad wff \longmapsto Abstract\ Wff$$

soggetta al seguente vincolo: tutti i simboli ground e "\$" occorrenti nella parte destra di una rule devono occorrere anche nella corrispondente parte sinistra. Ad esempio:

$$\begin{array}{rcl}
\$A\ \mathit{and}\ \$B & \longmapsto & WABS(\$A)\ \mathit{and}\ WABS(\$B) \\
\$A\ \mathit{imp}\ \$B & \longmapsto & \mathit{not}\ WABS(\$A)\ \mathit{or}\ WABS(\$B) \\
\$f(\$x) & \longmapsto & FUMAP[\$f](TABS(\$x)) \\
Posata(\$x) & \longmapsto & PMAP[Posata](TABS(\$x)) \\
MoglieDi(\$x) & \longmapsto & FIMAP[MoglieDi] \\
Intero(\$x)\ \mathit{and}\ Positivo(\$y) & \longmapsto & PMAP[Intero](TABS(\$x)).
\end{array}$$

[5]Anche in questi casi si usano i vocaboli inglesi per non creare ambiguità.

Nel caso di conflitto tra due o piú rule, i simboli appartenenti al contesto ground "prevalgono" sulle meta-variabili "\$". Considerando, ad esempio, la terza e la quinta regole di p.78, ed applicandole al termine $MoglieDi(Piero)$, "$MoglieDi(.)$" vince il conflitto con "$\$f(.)$", e viene quindi applicata la quinta regola. Si noti come

$$MoglieDi(\$x) \longmapsto FIMAP[MoglieDi]$$

costituisca un'eccezione alla regola piú generale:

$$\$f(\$x) \longmapsto FUMAP[\$f](TABS(\$x)).$$

La definizione delle rule nella forma vista permette una definizione dell'algoritmo di mapping in una forma quasi [6] completamente **dichiarativa**. La sintassi introdotta, oltre ad essere semplice ed intuitiva, risulta in qualche modo simile a quella della logica del primo ordine.

Da questa trattazione traspare un aspetto che si rivelerà essenziale in seguito, cioè l'esistenza di *due tipi di regole*:

- **Regole indipendenti dal contesto**, in cui non compaiono simboli appartenenti ad alcun alfabeto. Esse trascrivono solo le regole di buona formatezza delle espressioni, senza lasciarsi "condizionare" dalla presenza di determinati simboli. All'interno di un'astrazione esse tipicamente rappresentano il modo "principale" di tradurre una certa espressione. Le prime tre regole di p.78 appartengono a questa categoria.

- **Regole dipendenti dal contesto**, in cui compaiono simboli di un alfabeto (tipicamente quello ground poiché, come vedremo in seguito, l'occorrenza dei simboli di $\mathcal{A}_2$ è eliminabile). Le ultime tre regole di p.78 appartengono a questa seconda categoria. Nella penultima il termine[7] generico $MoglieDi(x)$ viene mappato in $DonnaSposata$, ma *non tutti i termini nella forma $f(x)$ sono mappati in un termine costante.* L'applicazione di tali regole è subordinata all'occorrenza di determinati simboli nell'espressione da astrarre. Nell'uso generale, queste regole costuiscono le "eccezioni" al modo "principale" di mappare un termine od una formula. Se, per esempio, consideriamo la prima e l'ultima delle regole viste sopra, ogni occorrenza del simbolo di congiunzione "$\wedge$" viene preservata, ad eccezione del caso "$Intero(x) \wedge Positivo(x)$", in cui il secondo congiungente "$Positivo(x)$" viene eliminato.

L'esistenza di regole contesto-dipendenti è la conseguenza di un'implicita caratteristica della nozione di `astrazione` : *il modo di astrarre la struttura logica di una wff o di un termine non è sempre indipendente dall'alfabeto.*

[6] a meno dell'implicita proceduralità introdotta dall'uso di regole "*contesto-dipendenti*", come si vedrà tra poco.

[7] Si noti che si mappa un *termine* nella forma $f(x)$ in un *termine* a, e non una *funzione* f in una *costante* a!

La presenza di regole contesto-dipendenti introduce un'implicita **proceduralità** nella dichiarazione di una mappa f_Λ. Ad esempio, l'aggiunta della regola:

$$\vdots$$

```
(P($x) and Q($Y)) imp $A  -> PMAP[Q](TABS($y)) imp WABS($A)
```

$$\vdots$$

equivale all'introduzione di una sequenza if-then-else del tipo:

```
if
    {"exp" UNIFICA CON "(P($x) and Q($Y)) imp $A"}
then
    {TAGLIA LA PRECONDIZIONE "P($x)"}
else
    {APPLICA LA REGOLA PIU' GENERALE}
```

4.4.4 f_Λ e le astrazioni

A questo punto possediamo tutti gli elementi per introdurre la nozione effettiva di f_Λ:

Una **funzione tra linguaggi del primo ordine** f_Λ:$\Lambda_1 \longmapsto \Lambda_2$ *è formata da:*

(a) *Sei distinti gruppi di "map", definite come in sez. 4.4.2, che mappano i simboli dell'alfabeto* $\mathcal{A}_1$*;*

(b) *una lista di "rule", definite come in sez. 4.4.3, incaricate della trascrittura delle espressioni logiche.*

Di conseguenza, seguendo la definizione formale n. 2, un'**astrazione** viene ad essere costituita da:

(i) *un nome;*

(ii) *il nome del contesto ground;*

(iii) *il nome del contesto astratto;*

(iv) *una funzione* f_Λ *tra i due linguaggi come definita in precedenza.*

Per illustrare meglio tali concetti, introduciamo un ulteriore esempio di dichiarazione ed uso di un'astrazione nel sistema GETFOL:

Esempio 10
Supponiamo di voler realizzare un meccanismo che converta un generico circuito elettronico digitale in uno che utilizzi solo porte "nand" e "not" (es. TTL). Questo corrisponde ad un'astrazione ("TTL") tra contesti proposizionali che mantenga inalterati i simboli alfabetici, e cambi la struttura logica delle espressioni. Tale mappa è TC.

```
World:: ABSTRACTION DECLARE TTL
    GROUND World
    ABSTRACT TTLworld
    FLAMBDA
        MAPS
            SMAPS
            PSMAPS
            PMAPS
            FUMAPS
            FIMAPS
            IMAPS
         RULES
                not (not $A ) -> WABS($A)
                not ($A and $B) -> WABS($A) nand WABS($B)
                ($A and $B) -> not (WABS($A) nand WABS($B))
                ($A or $B)  -> WABS(not $A) nand WABS(not $B)
                (not $A or not $B)  -> WABS ($A) nand WABS ($B)
                ($A or not $B)  -> WABS (not $A) nand WABS ($B)
                (not $A or $B)  -> WABS ($A) nand WABS (not $B)
                not $A    -> not WABS($A)
                $A -> SMAP[$A];

World:: ABSTRACT WFF (A and not B) or (not A and B) BY TTL;
You are now in the abstract context 'TTLworld'.
(A nand (not B)) nand ((not A) nand B)
is the mapped wff from 'World'.
```

L'assenza di mappe è dovuta al fatto che l'alfabeto astratto è uguale a quello ground. Si noti come la formula dell'esempio rappresenti una delle uscite di un circuito "*half adder*". □

Nella sezione 4.2 abbiamo introdotto i requisiti che avremmo voluto vedere soddisfatti da una definizione operativa di f_Λ. Analizziamoli ora uno ad uno.

(i) La definizione data è in grado di realizzare la **doppia funzionalità** richiesta (generazione dell'alfabeto $\mathcal{A}_2$ ed astrazione di formule e termini). Utilizzando esclusivamente le "*map*" è infatti possibile generare l'alfabeto del contesto astratto come immagine di quello a livello ground ($\mathcal{A}_2$:=MAP($\mathcal{A}_1$)). Le due interpretazioni di default suggerite nella sezione 4.4.2 fanno sì che tutta l'informazione necessaria sia ricavabile dalle map stesse.

La dichiarazione dell'alfabeto astratto come immagine di quello ground è un processo non solo molto comodo, ma anche *vivamente consigliabile* all'utente, in quanto garantisce la **suriettività** di f_Λ (Λ_2:=$f_\Lambda(\Lambda_1)$).

Una volta dichiarato $\mathcal{A}_2$, l'uso combinato di "rule" e "map" permette a f_Λ di astrarre intere espressioni logiche.

(ii) La nozione di f_Λ introdotta è sufficientemente **generale**. Tramite le "rule" è possibile definire tutte le relazioni tra espressioni scrivibili nella forma:

$$\alpha(\underline{s}, \underline{\$v}) \longmapsto \beta(\underline{s}', \underline{\$v}')$$

dove $\underline{s}$, $\underline{s}'$ sono due generici gruppi di dimboli ground e $\underline{\$v}$, $\underline{\$v}'$ due gruppi dimeta-variabili, tali che $\underline{s}' \subseteq \underline{s}$ e $\underline{\$v}' \subseteq \underline{\$v}$. Una prima analisi sommaria, e sicuramente incompleta, delle potenzialità espressive di questa formulazione ha mostrato come essa catturi praticamente tutti gli esempi presentati in [Giu].

(iii) La separazione tra map e rule e la descrizione ricorsiva utilizzata rendono la dichiarazione di f_Λ **semplice** ed **intuitiva**. Un utente dotato di una conoscenza, per lo meno intuitiva, della logica dei predicati, dovrebbe essere in grado di dichiarare facilmente un'astrazione (direttamente o tramite istanziazione di un prototipo, come vedremo in seguito), e di comprendere il significato di una mappa precedentemente definita. Tutto questo senza che siano richieste particolari conoscenze informatiche.

Possiamo quindi concludere che la nozione di f_Λ descritta soddisfa i requisiti che ci eravamo proposti.

4.5 I prototipi di astrazione

Molte astrazioni che appaiono completamente diverse tra loro in realtà hanno in comune il modo di astrarre la *struttura logica* di un'espressione. Per esempio, una funzione che mappi tutte le relazioni di parentela in analoghe relazioni "asessuate":

$$\vdots$$

$$moglie(x, y),\ marito(x, y) \longmapsto coniuge(x, y)$$

$$\vdots$$

ed una che mappi oggetti simili in uno piú generico:

$$\vdots$$

$$forchetta(x),\ cucchiaio(x), coltello(x) \longmapsto posata(x)$$

$$\vdots$$

differiscono solo per i rispettivi alfabeti usati, mentre entrambe preservano la struttura delle espressioni. Analogamente, potremmo pensare a due funzioni con domini completamente diversi che collassino costanti individuali ed eliminino le occorrenze dell'implicazione:

$$\alpha \supset \beta \longmapsto \neg\alpha \vee \beta.$$

Anche queste mappano la struttura delle espressioni allo stesso modo.

Possiamo quindi concepire l'esistenza di **prototipi di astrazioni**, indipendenti dai contesti, ciascuno dei quali rappresenti un modo di astrarre la struttura delle espressioni logiche. Potremo definire delle nuove astrazioni semplicemente **istanziando** tali prototipi con le mappe sull'alfabeto richiesto.

La definizione di un *prototipo di astrazione* è ricavabile dalla nozione di f_Λ introdotta a p.81. Esso è infatti costituito da:

- *un nome;*
- *un insieme di regole contesto-indipendenti.*

Un prototipo individua un'algoritmo di trascrittura della struttura logica di una formula [termine], che può essere comune a molte astrazioni apparentemente diverse. Esso può venire istanziato ad un particolare contesto tramite la dichiarazione delle "map", ed eventualmente l'aggiunta di ulteriori regole contesto-dipendenti (cfr. Capitolo 5).

Ogni prototipo è potenzialmente instanziabile ad infiniti contesti diversi, e può quindi generare una grande quantità di astrazioni. Si noti, per esempio, come l'importe categoria delle *astrazioni teoria-invarianti* sia ottenibile istanziando un unico prototipo. L'eventuale aggiunta di "eccezioni", tramite regole contesto-dipendenti, permette di incrementare la versatilità della dichiarazione di astrazioni mediante prototipi. Seguendo questa idea si possono inoltre creare delle **librerie di prototipi**, in modo che un'utente possa dichiarare le astrazioni volute limitandosi a definire le map, operazione semplice e veloce.

Esempio 11
Riprendiamo l'esempio del gatto che miagola e del cane che abbaia (p. 31), osservando come l'astrazione in questione sia *teoria-invariante*. Per questo motivo possiamo utilizzare un prototipo standard (*"theory_invariant"*) che mantiene inalterata la struttura delle espressioni logiche, ed istanziarlo nella nuova astrazione "pets". Potremo in futuro riutilizzare tale prototipo in molti altri casi.

```
World:: ABSTRACTION PROTOTYPE theory_invariant
    FLAMBDA
         RULES
                (forall $x. $A) -> (forall IMAP[$x]. WABS($A)),
                (exists $x. $A) -> (exists IMAP[$x]. WABS($A)),
                ($A and $B) -> (WABS($A) and WABS($B)),
                ($A or $B)  -> (WABS($A) or WABS($B)),
                ($A imp $B) -> (WABS($A) imp WABS($B)),
                ($A iff $B) -> (WABS($A) iff WABS($B)),
                (not $A )   -> (not WABS($A))
                ($x = $y) -> (TABS($x) = TABS($y))
                $P1($x)  ->  PMAP[$P1](TABS($x))
                $f1($x)  ->  FUMAP[$f1](TABS($x))
                $x   -> IMAP[$x];
```

```
World:: ABSTRACTION INSTANTIATE theory_invariant TO pets BY
    GROUND World
    ABSTRACT PetWorld
    FLAMBDA
        MAPS
            SMAPS
            PSMAPS
            PMAPS :
                Cat -> Pet
                Dog -> Pet
                Meows -> Calls
                Barks -> Calls
            FUMAPS
            FIMAPS
            IMAPS :
                Pussy -> Poppy
                Tom   -> Poppy;

World:: ABSTRACT WFF ((forall x.(Cat(x) imp Meows(x))) and
                      (forall y.(Dog(y) imp Barks(y)))) and
                     (Cat(Pussy) and Dog(Tom)) BY pets;
You are now in the abstract context ‘PetWorld’.
((forall x. (Pet(x) imp Calls(x))) and
(forall y. (Pet(y) imp Calls(y))))
and (Pet(Poppy) and Pet(Poppy))
is the mapped wff from ‘World’.
```

□

4.6 Un interprete per le astrazioni

L'interprete riceve in ingresso la descrizione di f_Λ e l'espressione logica "ground" da astrarre, e restituisce la corrispondente espressione del linguaggio astratto. Tale funzionalità è piuttosto semplice. Essa può essere ripartita in quattro blocchi, due dei quali si chiamano in modo mutualmente ricorsivo (fig. 4.1):

- **Applicatore di map**: Dato un *mapped-sym* nella forma $*MAP[s]$, restituisce il corrispondente simbolo dell'alfabeto astratto. Se non esiste alcuna mappa *MAP in cui s occorra come parte sinistra, restituisce una copia "astratta" di s. Ad esempio, se nell'astrazione applicata occorre la PMAP:

  ```
  Cucchiaio  ->  Posata
  ```

  e viene passato il *mapped-symbol* "$PMAP[Cucchiaio]$", allora tale dispositivo dovrà restituire il simbolo "$Posata$".

- **Unificatore**: [8] Data una generica espressione di livello ground $exp_g^{(i)}$, cerca una regola la cui parte sinistra unifichi con essa. Se non la trova, da una risposta negativa, altrimenti restituisce la corrispondente *abstract-expression*, con i parametri "$" opportunamente istanziati. Per esempio, se nella descrizione di f_Λ è presente la regola:

  ```
  $P1($x)  ->  PMAP[$P1](TABS($x))
  ```

  e viene passata la formula ground "$Uomo(Socrate)$", l'unificatore restituisce l'*abstract wff*: "$PMAP[Uomo](TABS(Socrate))$."

- **Interprete**: riceve l'espressione exp_g da astrarre, e restituisce l'espressione exp_a desiderata. Il suo funzionamento è piuttosto semplice. All'i-esimo passo di ricorsione, chiama l'unificatore passandogli l'espressione $exp_g^{(i)}$ ricevuta in ingresso. Se la risposta è negativa, genera un messaggio d'errore. Altrimenti prende l'abstract expression ricevuta ($absexp^{(i)}$) e la passa al traduttore. Il risultato ottenuto è riportato direttamente in uscita.

- **Traduttore di abs-expression** è il vero e proprio meccanismo di astrazione, in quanto traduce un'*abstract-expression* $absexp^{(i)}$ nella corrispondente espressione $exp_a^{(i)}$ appartenente al linguaggio astratto.

  Esso agisce ricorsivamente sulla costruzione di $absexp^{(i)}$:

  - Se $absexp^{(i)}$ è un *mapped-symbol* "$*MAP[s]$", lo traduce nel corrispondente simbolo astratto effettuando una chiamata all'*applicatore di map*.

[8] Nella realizzazione in `GETFOL` si è utilizzato un meccanismo di unificazione preesistente nel sistema, basato sul concetto di "**amf**" (**A**ssociative **M**emory **F**ormula) [GP91], in cui un'espressione è usata come chiave per reperire dell'informazione ad essa associata (nel nostro caso l'*abstract-expression* relativa).

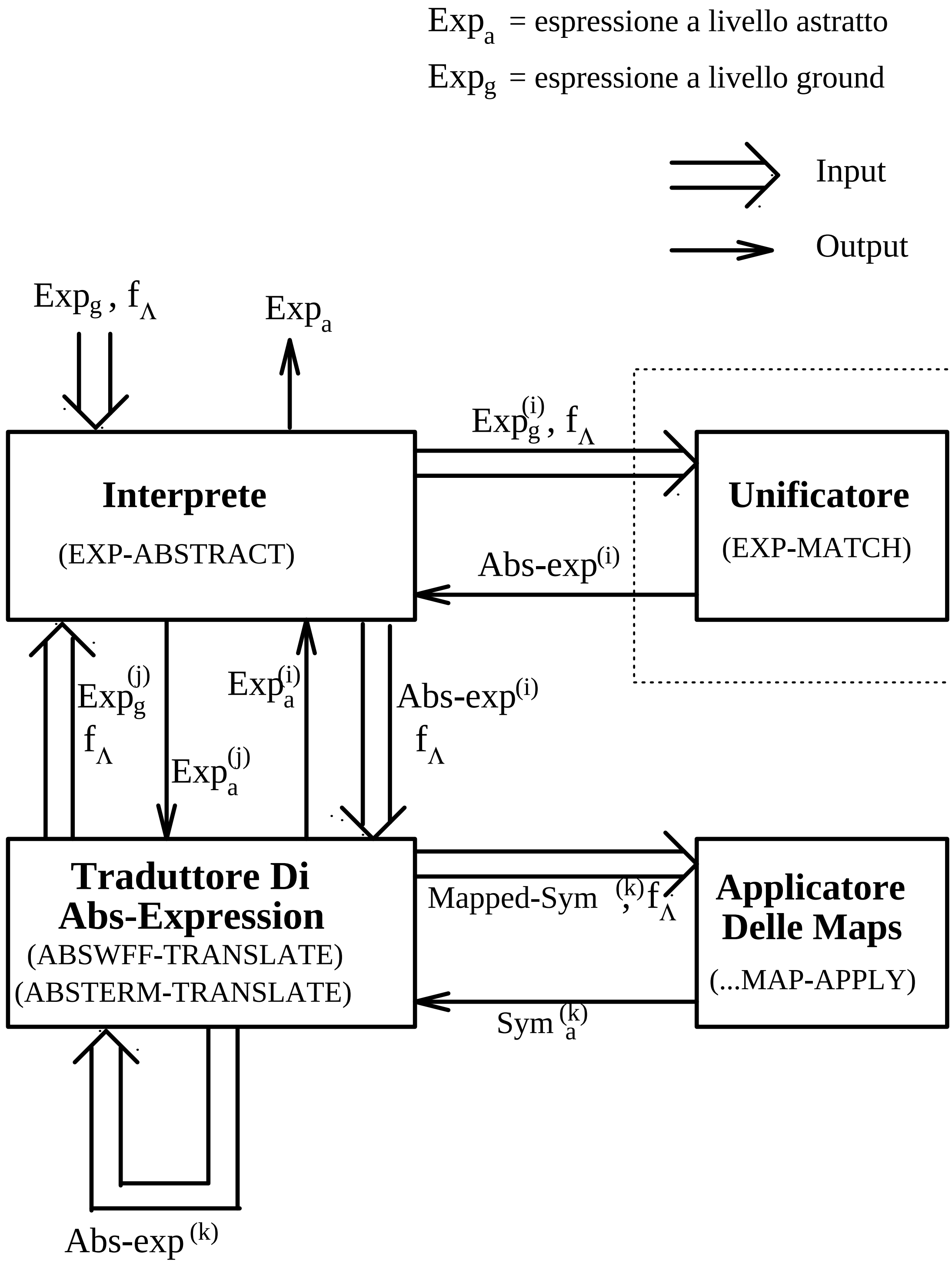


Figura 4.1: Schema dell'interprete di astrazioni.

- Se $absexp^{(i)}$ è nella forma "$WABS(wff)$" o "$TABS(term)$", richiama l'*interprete* passandogli il contenuto di "$*ABS(.)$".
- Se infine $absexp^{(i)}$ è composta, mediante le regole di buona formatezza della logica, da *mapped-symbols* e *abstract-expressions*, il modulo si richiama ricorsivamente per tradurre ogni componente nel modo visto.

Tutti e quattro i moduli ricevono in ingresso anche la descrizione di f_Λ.

Esempio 12
Consideriamo l'esempio 9, dove viene presentata un'astrazione che conserva la struttura logica delle espressioni e mappa rispettivamente il predicato $Maestro$ in *Insegnante*, la funzione $padre$ in $genitore$ e la costante individuale *Paolo_Bianchi* in $Paolo$. Tutti gli altri simboli sono astratti in una copia di sè stessi. Proviamo a tracciare il comportamento delle routine relative alle quattro funzionalità menzionate: [9]

```
World:: ABSTRACT WFF Maestro(padre(Mario)) and Operaio(Paolo_Bianchi)
         BY FirstExample;

  1> (EXP-ABSTRACT (and (Maestro (padre Mario)) (Operaio Paolo_Bianchi)))
    2> (EXP-MATCH (and (Maestro (padre Mario)) (Operaio Paolo_Bianchi)))
    <2 (EXP-MATCH (MATCH and (WABS (Maestro (padre Mario))) (WABS
(Operaio Paolo_Bianchi))))
    2> (ABSWFF-TRANSLATE (and (WABS (Maestro (padre Mario))) (WABS
(Operaio Paolo_Bianchi))))
      3> (ABSWFF-TRANSLATE (WABS (Maestro (padre Mario))))
        4> (EXP-ABSTRACT (Maestro (padre Mario)))
          5> (EXP-MATCH (Maestro (padre Mario)))
          <5 (EXP-MATCH (MATCH PMAP[Maestro] (TABS (padre Mario))))
          5> (ABSWFF-TRANSLATE (PMAP[Maestro] (TABS (padre Mario))))
            6> (PMAP-APPLY PMAP[Maestro])
            <6 (PMAP-APPLY Insegnante)
            6> (ABSTERM-TRANSLATE (TABS (padre Mario)))
              7> (EXP-ABSTRACT (padre Mario))
                8> (EXP-MATCH (padre Mario))
                <8 (EXP-MATCH (MATCH FUMAP[padre] (TABS Mario)))
                8> (ABSTERM-TRANSLATE (FUMAP[padre] (TABS Mario)))
                  9> (FUMAP-APPLY FUMAP[padre])
                  <9 (FUMAP-APPLY genitore)
                  9> (ABSTERM-TRANSLATE (TABS Mario))
                    10> (EXP-ABSTRACT Mario)
                      11> (EXP-MATCH Mario)
```

[9]Per motivi di leggibilità in tutte le chiamate è stato eliminato il secondo parametro, che contiene la descrizione di f_Λ. Inoltre i nomi delle funzioni sono stati sostituiti con nomi più comprensibili: "EXP-ABSTRACT" per l'interprete, "EXP-MATCH" per l'unificatore, "ABSWFF-TRANSLATE" ed "ABSTERM-TRANSLATE" per il traduttore, "*MAP-APPLY" per l'applicatore di mappe.

```
                    <11 (EXP-MATCH (MATCH . IMAP[Mario]))
                    11> (ABSTERM-TRANSLATE IMAP[Mario])
                      12> (IMAP-APPLY IMAP[Mario])
                      <12 (IMAP-APPLY Mario)
                    <11 (ABSTERM-TRANSLATE Mario)
                  <10 (EXP-ABSTRACT Mario)
                <9 (ABSTERM-TRANSLATE Mario)
              <8 (ABSTERM-TRANSLATE (genitore Mario))
            <7 (EXP-ABSTRACT (genitore Mario))
          <6 (ABSTERM-TRANSLATE (genitore Mario))
        <5 (ABSWFF-TRANSLATE (P (genitore Mario)))
      <4 (EXP-ABSTRACT (Insegnante (genitore Mario)))
    <3 (ABSWFF-TRANSLATE (Insegnante (genitore Mario)))
    3> (ABSWFF-TRANSLATE (WABS (Operaio Paolo_Bianchi)))
      4> (EXP-ABSTRACT (Operaio Paolo_Bianchi))
        5> (EXP-MATCH (Operaio Paolo_Bianchi))
        <5 (EXP-MATCH (MATCH PMAP[Operaio] (TABS Paolo_Bianchi)))
        5> (ABSWFF-TRANSLATE (PMAP[Operaio] (TABS Paolo_Bianchi)))
          6> (PMAP-APPLY PMAP[Operaio])
          <6 (PMAP-APPLY Operaio)
          6> (ABSTERM-TRANSLATE (TABS Paolo_Bianchi))
            7> (EXP-ABSTRACT Paolo_Bianchi)
              8> (EXP-MATCH Paolo_Bianchi)
              <8 (EXP-MATCH (MATCH . IMAP[Paolo_Bianchi]))
              8> (ABSTERM-TRANSLATE IMAP[Paolo_Bianchi])
                9> (IMAP-APPLY IMAP[Paolo_Bianchi])
                <9 (IMAP-APPLY Paolo)
              <8 (ABSTERM-TRANSLATE Paolo)
            <7 (EXP-ABSTRACT Paolo)
          <6 (ABSTERM-TRANSLATE Paolo)
        <5 (ABSWFF-TRANSLATE (Operaio Paolo))
      <4 (EXP-ABSTRACT (Operaio Paolo))
    <3 (ABSWFF-TRANSLATE (Operaio Paolo))
  <2 (ABSWFF-TRANSLATE (and (Insegnante (genitore Mario)) (Operaio
Paolo)))
 <1 (EXP-ABSTRACT (and (Insegnante (genitore Mario)) (Operaio Paolo)))

You are now in the abstract context ‘AbstractWorld’.
Insegnante(genitore(Mario)) and Operaio(Paolo)
is the mapped wff from ground context ‘World’.

AbstractWorld::
```

□

Capitolo 5

I comandi per l'astrazione in GETFOL

5.1 Introduzione

Il sistema `GETFOL` [GT91, GAC+91] risulta essere particolarmente adatto alla realizzazione di un "*abstract theorem prover*", a causa della possibilità di lavorare tra contesti diversi. La nozione di *contesto* in `GETFOL` è infatti analoga a quella di *sistema formale* della Teoria dell'Astrazione (cfr. def. 1 di p.17). Possiamo quindi descrivere un'astrazione in `GETFOL` tramite una mappa tra i linguaggi di due differenti contesti.

Scopo del lavoro di realizzazione dell'astrazione in `GETFOL` è implementare un nuovo *modulo* del sistema che permetta all'utente, in modo più o meno interattivo, di dimostrare un teorema per astrazione, secondo la tecnica descritta nel capitolo 2.

Con il lavoro di questa tesi è stata realizzata la prima parte di tale obiettivo. Sono stati definiti ed implementati i **comandi** per permettereall'utente di *definire* un'astrazione e di *applicarla* agli oggetti desiderati (formule, termini, assiomi, semplici simboli dell'alfabeto ...), generando quindi i corrispondenti oggetti nello spazio astratto.

In questo capitolo vengono illustrate le modalità d'uso di tali comandi.

La trattazione è scritta secondo un formato analogo al manuale utente di `GETFOL` [GT91]. Infatti nella descrizione della sintassi dei comandi ci riferemo alle convenzioni sintattiche utilizzate in tale manuale.

Useremo quindi le parentesi quadre "[...]" per indicare parti opzionali; scriveremo in **neretto** le parole chiave (es. "**abstract**"), ed in *corsivo* il resto (es. "*nome dell'astrazione*"); useremo infine il font "`ttype`" per rappresentare gli esempi presi da esecuzioni nel sistema `GETFOL`.

5.2 Comandi di definizione delle astrazioni

NOME

Abstraction - definizione di un'astrazione.

SINTASSI

Abstraction declare
nome dell'astrazione
corpo dell'astrazione
Abstraction prototype
nome del prototipo
corpo del prototipo
Abstraction instantiate
nome prototipo **to**
nome dell'astrazione **by**
informazione aggiuntiva

DESCRIZIONE

"**Abstraction**" rappresenta una famiglia di tre comandi dedicati alla dichiarazione di astrazioni.

Un'astrazione è costituita da (cfr. p.99):

- Un **nome**. Tale nome deve individuare un'astrazione in modo univoco.
- Il **nome del spazio ground**, che deve sempre riferirsi al contesto corrente all'atto di dichiarazione.
- Il **nome dello spazio astratto**, che deve sempre riferirsi ad un contesto già esistente, eventualmente vuoto. Si raccomanda all'utente di riferirsi a contesti astratti vuoti e di generarne l'alfabeto tramite il comando *abstract alphabet* (cfr. p.114).
- La descrizione di una mappa *suriettiva*,*computabile* e *totale* tra i due linguaggi (**flambda**). Tale rapresentazione enfatizza due aspetti: il mapping dei simboli dell'alfabeto e quello della struttura delle espressioni. Il primo viene rappresentato tramite le *map*, il secondo per mezzo delle *rule*.
  - Una **map** è scrivibile nella forma:

    simbolo ground $\longmapsto$ *simbolo astratto [informazione supplementare].*

    Esistono sei diverse categorie di map. Quattro di esse (*smap, pmap, fumap* ed *imap*) si riferiscono a funzioni tra simboli dello stesso tipo, cioè proposizioni, predicati, funzioni, individui. Le altre due, (*psmap* e *fimap*), permettono di mappare un predicato [una funzione] in un simbolo proposizionale [individuale]. Si noti, per esempio, come una costante individuale possa essere considerata in questo modo come una funzione di arità zero.

Le caratteristiche del simbolo di destra di una mappa (ad es. arità) possono venire specificate tramite dell'informazione aggiuntiva. Se questo non avviene, si considera che tale simbolo conservi le caratteristiche del simbolo di sinistra. Inoltre, se un simbolo ground non occorre come parte sinistra di una map, verrà astratto in una copia di sè stesso (identità per default).
Esistono due vincoli sulla dichiarazione delle map. L'astrazione di un simbolo dev'essere determinata univocamente, cioè non devono esistere due map distinte con lo stesso simbolo sulla sinistra. In secondo luogo, due simboli astratti con lo stesso nome devono avere le stesse caratteristiche. Un test interno, attivato in fase di dichiarazione, assicura che tali vincoli vengano rispettati.

– Una **rule** è rappresentata nella forma:

$$\begin{array}{c} \textit{formula} \longmapsto \textit{abstract-wff} \\ \textit{termine} \longmapsto \textit{abstract-term}. \end{array}$$

  Essa descrive il modo con cui un'espressione generica viene mappata nello spazio astratto. Un' "abstract expressions" utilizza alcuni operatori, come WABS (**W***ff*-**ABS***tract*), TABS (**T***erm*-**ABS***tract*) e **...MAP**, che rappresentano delle chiamate ricorsive alla mappa delle sotto-espressioni e dei simboli dell'alfabeto. Un gruppo di metavariabili di sistema ("\$") vengono utilizzate per rappresentare simboli, termini e formule ground.
  Esiste un vincolo alla dichiarazione di una rule: ogni simbolo (ground e "\$") occorrente nella parte destra deve occorrere anche nella rispettiva parte sinistra. Nessun simbolo appartenente all'alfabeto astratto può occorrere in una rule.
  Una regola in cui non occorrano simboli ground è detta *contesto-indipendente*; in caso contrario viene detta *contesto-dipendente*.

La dichiarazione di un'astrazione può avvenire secondo due possibili modalità:

(i) **diretta**: (cfr p.99). L'astrazione viene dichiarata per esteso, definendo tutte le sue parti. Tale sistema è sicuramente il meno comune, in quanto la scrittura del testo per esteso può risultare onerosa. Per facilitare la scrittura, l'utente può scrivere il testo della dichiarazine su un file esterno (mediante un editor, un foglio elettronico o addirittura un programma apposito) e richiamarlo all'interno di `GETFOL` tramite il comando *fetch* [GT91].

(ii) **indiretta**: tramite l'istanziazione di un *prototipo* (cfr. pp.102 e 104). Molte astrazioni apparentemente diverse possono avere in comune il modo di astrarre la *struttura* delle espressioni logiche. Tale elemento comune è dato da un insieme di *rule*, che costituiscono per l'utente la parte più difficile da scrivere. Egli può definire un *prototipo* di astrazione, cioè un insieme di regole indipendenti da contesto, ed istanziarlo in varie astrazioni. Inoltre egli può istanziare direttamente un prototipo già presente in libreria. In questo modo la dichiarazione di un'astrazione risulta estremamente semplificata.

Le tre opzioni del comando "**Abstraction**" sono descritte in dettaglio nelle pagine seguenti. Si noti come, ad eccezione di "*abstraction prototype*", la chiamata al comando venga effettuata nell'ambito del contesto ground dell'astrazione utilizzata.

5.2.1 La dichiarazione "diretta" di astrazioni

NOME

abstraction declare - dichiarazione "diretta" di un'astrazione.

SINTASSI

abstraction declare
    nome dell'astrazione
  ground
    nome del contesto ground
  abstract
    nome del contesto astratto
  flambda
    maps
      smaps
        [**:** *lista di map tra proposizioni*]
      psmaps
        [**:** *lista di map predicato-proposizione*]
      pmaps
        [**:** *lista di map tra predicati*]
      fumaps
        [**:** *lista di map tra funzioni*]
      fimaps
        [**:** *lista di map funzione-individuo*]
      imaps
        [**:** *lista di map tra individui*]
    rules
      lista di rule

DESCRIZIONE

Con questo comando un'astrazione viene dichiarata per esteso. Poiché la scrittura diretta di un'astrazione può risultare anche piuttosto lunga, è consigliabile scriverne il testo su di un file separato e utilizzare il comando *fetch* [GT91] per richiamarlo in `GETFOL`.

ESEMPIO

Come primo esempio consideriamo una semplice astrazione che collassa due predicati unari, "$Traslabile(.)$" ed "$Afferrabile(.)$", in un unico predicato dal significato più generale "$Movibile(.)$". La struttura logica delle espressioni viene mantenuta inalterata.

Per rendere più evidente le operazioni di salto tra contesti, in tutti gli esempi il prompt del sistema viene ridefinito con il nome del contesto corrente mediante il comando "*setprompt*" [GT91].

```
GETFOL:: namecontext c0;
You have named the current context: c0
```

```
GETFOL:: setprompt to (curcname\-get);

c0:: declare indconst a;
a has been declared to be a Indconst

c0:: declare predconst Bottle Shiftable Graspable;
Bottle has been declared to be a Predconst
Shiftable has been declared to be a Predconst
Graspable has been declared to be a Predconst

c0:: abstraction declare collaps
    ground c0
    abstract c1
    flambda
        maps
            smaps
            psmaps
            pmaps :
                Shiftable -> Movable
                Graspable -> Movable
            fumaps
            fimaps
            imaps
        rules
                (forall $x. $A) -> (forall IMAP[$x]. WABS($A))
                (exists $x. $A) -> (exists IMAP[$x]. WABS($A))
                ($A and $B) -> (WABS($A) and WABS($B))
                ($A or $B)  -> (WABS($A) or WABS($B))
                ($A imp $B) -> (WABS($A) imp WABS($B))
                ($A iff $B) -> (WABS($A) iff WABS($B))
                (not $A )   -> (not WABS($A))
                $P1($x)  ->  PMAP[$P1](TABS($x))
                $f1($x)  ->  FUMAP[$f1](TABS($x))
                $x   -> IMAP[$x];

c0::
```

5.2.2 La dichiarazione di prototipi di astrazione

NOME

abstraction prototype - dichiarazione di un prototipo.

SINTASSI

abstraction prototype
nome del prototipo
flambda
rules
lista di rule

DESCRIZIONE

Questo comando crea un nuovo **prototipo di astrazione**. Quest'ultimo è costituito da:

- Un **nome**. Tale nome non deve essere già utilizzato per un'altro prototipo.
- Un **insieme di regole** indipendenti da contesto.

Si noti come un prototipo sia indipendente da ogni sistema formale. Per questo motivo la sua dichiarazione può essere fatta in qualsiasi contesto.

ESEMPIO Presentiamo qui due esempi distinti.

Il primo rappresenta un tipo di astrazione "proposizionale", usata in logica matematica [Bor84], il quale elimina i quantificatori e mappa le formule atomiche in costanti proposizionali, una per ogni distinto predicato ground.

Il secondo esempio descrive un prototipo che rapresenta una categoria molto importante di astrazioni, le "*teoria-invarianti*" (cfr. def.7 di p.22), le quali mantengono inalterata la struttura logica delle espressioni.

```
GETFOL:: abstraction prototype star
    flambda
         rules
            (forall $x. $A) -> WABS($A)
            (exists $x. $A) -> WABS($A)
            ($A and $B) -> (WABS($A) and WABS($B))
            ($A or $B)  -> (WABS($A) or WABS($B))
            ($A imp $B) -> (WABS($A) imp WABS($B))
            ($A iff $B) -> (WABS($A) iff WABS($B))
            (not $A )   -> (not WABS($A))
            $P1($x)  -> PSMAP[$P1]
            $P2($x,$y) -> PSMAP[$P2];
```

```
GETFOL:: abstraction prototype theory_invariant
    flambda
        rules
            (forall $x. $A) -> (forall IMAP[$x]. WABS($A))
            (exists $x. $A) -> (exists IMAP[$x]. WABS($A))
            ($A and $B) -> (WABS($A) and WABS($B))
            ($A or $B)  -> (WABS($A) or WABS($B))
            ($A imp $B) -> (WABS($A) imp WABS($B))
            ($A iff $B) -> (WABS($A) iff WABS($B))
            (not $A )   -> (not WABS($A))
            $P1($x)      ->  PMAP[$P1](TABS($x))
            $P2($x,$y)   ->  PMAP[$P2](TABS($x),TABS($y))
            $f1($x)      ->  FUMAP[$f1](TABS($x))
            $f2($x,$y)   ->  FUMAP[$f2](TABS($x),TABS($y))
            $x   -> IMAP[$x];
```

5.2.3 L'istanziazione di prototipi

NOME

abstraction instantiate - istanziazione di un prototipo in un'astrazione.

SINTASSI

abstraction instantiate
nome del prototipo **to**
nome dell'astrazione **by**
ground
nome del contesto ground
abstract
nome del contesto astratto
flambda
maps
smaps
[**:** *lista di map tra proposizioni*]
psmaps
[**:** *lista di map predicato-proposizione*]
pmaps
[**:** *lista di map tra predicati*]
fumaps
[**:** *lista di map tra funzioni*]
fimaps
[**:** *lista di map funzione-individuo*]
imaps
[**:** *lista di map tra individui*]
[**newrules**
lista di nuove rule]

DESCRIZIONE

Questo comando crea una nuova astrazione istanziando un prototipo. L'atto di istanziazione è molto simile alla dichiarazione "diretta". La differenza sta nel fatto che non è più necessario scrivere le rule, poiché vengono richiamate quelle del prototipo.

È possibile aggiungere alcune nuove rule, generalmente contesto-dipendenti, per estendere la mappa ad espressioni particolari per cui le regole del prototipo risultino inadeguate.

ESEMPIO

In questo esempio presentiamo una possibile istanziazione per i due prototipi di p.102.

Nel primo caso viene generata un'astrazione proposizionale "star1" aggiungendo una serie di mappe predicato-proposizione al prototipo "star". Il risultato è illustrato mediante il comando "*show*" [GT91].

```
GETFOL:: namecontext groundstar;
You have named the current context: groundstar

GETFOL:: setprompt to (CAPITALIZE (curcname\-get));

GROUNDSTAR:: declare predconst P Q R S T 1;
P has been declared to be a Predconst
Q has been declared to be a Predconst
R has been declared to be a Predconst
S has been declared to be a Predconst
T has been declared to be a Predconst

GROUNDSTAR:: declare predconst P2 Q2 R2 S2 T2 2;
P2 has been declared to be a Predconst
Q2 has been declared to be a Predconst
R2 has been declared to be a Predconst
S2 has been declared to be a Predconst
T2 has been declared to be a Predconst

GROUNDSTAR:: makecontext absstar;
You have created the empty context: absstar

GROUNDSTAR:: abstraction instantiate star to star1 by
    ground groundstar
    abstract absstar
    flambda
        maps
            smaps
            psmaps :
                P -> PS
                Q -> QS
                R -> RS
                S -> SS
                T -> TS
                P2 -> PS2
                Q2 -> QS2
                R2 -> RS2
                S2 -> SS2
                T2 -> TS2
            pmaps
            fumaps
            fimaps
            imaps;
```

```
GROUNDSTAR:: show abstraction star1;

ABSTRACTION: star1
GROUND: groundstar
ABSTRACT: absstar
FLAMBDA
MAPS:
Smaps
Psmaps:
P  ->  PS
Q  ->  QS
R  ->  RS
S  ->  SS
T  ->  TS
P2  ->  PS2
Q2  ->  QS2
R2  ->  RS2
S2  ->  SS2
T2  ->  TS2
Pmaps
Fumaps
Fimaps
Imaps
RULES:
forall $x. $A  ->  WABS $A
exists $x. $A  ->  WABS $A
$A and $B  ->  (WABS $A) and (WABS $B)
$A or $B  ->  (WABS $A) or (WABS $B)
$A imp $B  ->  (WABS $A) imp (WABS $B)
$A iff $B  ->  (WABS $A) iff (WABS $B)
not $A  ->  not (WABS $A)
$P1($x)  ->  PSMAP[$P1]
$P2($x,$y)  ->  PSMAP[$P2]
;
```

Come secondo esempio creiamo un'atrazione teoria-invariante, "*family*", un po' più complessa della precedente.

In primo luogo essa collassa una serie di predicati binari, che descrivono alcune relazioni di parentela, nelle corrispondenti relazioni "asessuate" (ad esempio "$padre(.)$" in "$genitore(.)$").

In secondo luogo sostituisce i nomi con cui le persone sono conosciute nel mondo esterno con quelli con cui vengono chiamate nel contesto locale "SmithHome" (per esempio "$Mary\_Smith$" in "$Mary$").

Infine, essa elimina un argomento dei predicati "$UomoSposatoCon(.,.)$" e "$DonnaSposataCon(.,.)$", mappandoli nei predicati unari "$UomoSposato(.)$" e "$DonnaSposata(.)$".

```
GROUNDSTAR:: makecontext world;
You have created the empty context: world

GROUNDSTAR:: switchcontext world;
You are now using context: world.

WORLD:: declare predconst mother father son doughter brother
sister 2;
mother has been declared to be a Predconst
father has been declared to be a Predconst
son has been declared to be a Predconst
doughter has been declared to be a Predconst
brother has been declared to be a Predconst
sister has been declared to be a Predconst

WORLD:: declare predconst ManMarriedWith WomanMarriedWith 2;
ManMarriedWith has been declared to be a Predconst
WomanMarriedWith has been declared to be a Predconst

WORLD:: declare indconst Mary_Smith John_Smith Mark_Smith
Anne_Smith;
Mary_Smith has been declared to be an Indconst
John_Smith has been declared to be an Indconst
Mark_Smith has been declared to be an Indconst
Anne_Smith has been declared to be an Indconst

WORLD::
declare indconst Anne_Hobbs Charles_Hobbs William_Hobbs
Donna_Hobbs;
Anne_Hobbs has been declared to be an Indconst
Charles_Hobbs has been declared to be an Indconst
William_Hobbs has been declared to be an Indconst
Donna_Hobbs has been declared to be an Indconst
```

```
WORLD:: declare indvar x y z ;
x has been declared to be an Indvar
y has been declared to be an Indvar
z has been declared to be an Indvar

WORLD:: makecontext SmithHome;
You have created the empty context: SmithHome

WORLD:: abstraction instantiate theory_invariant to family by
    ground world
    abstract SmithHome
    flambda
        maps
            smaps
            psmaps
            pmaps :
                mother   -> parent
                father   -> parent
                son      -> child
                doughter -> child
                sister   -> brother
                ManMarriedWith -> MarriedMan arity 1
                WomanMarriedWith -> MarriedWoman arity 1
            fumaps
            fimaps
            imaps :
            imaps :
                Mary_Smith -> Mary
                John_Smith -> John
                Mark_Smith -> Mark
                Anne_Smith -> Anne
        newrules
            ManMarriedWith($x,$y) ->
                        PMAP[ManMarriedWith](TABS($x))
            WomanMarriedWith($x,$y) ->
                        PMAP[WomanMarriedWith](TABS($x));
```

Si noti come l'introduzione delle due nuove regole contesto-dipendenti si sia resa necessaria per gestire le ultime due mappe tra predicati, le quali "riducono" l'arità del simbolo da 2 a 1. In genere, le rule del prototipo stabiliscono il modo "principale" di astrarre le espressioni, mentre le eventuali regole aggiunte creano delle "eccezioni".

L'astrazione risultante viene illustrata qui di seguito.

```
WORLD:: Show abstraction family;

ABSTRACTION: family
GROUND: world
ABSTRACT: SmithHome
FLAMBDA
MAPS:
Smaps
Psmaps
Pmaps:
mother  ->  parent
father  ->  parent
son  ->  child
doughter  ->  child
sister  ->  brother
ManMarriedWith  ->  MarriedMan
WomanMarriedWith  ->  MarriedWoman
Fumaps
Fimaps
Imaps:
Mary_Smith  ->  Mary
John_Smith  ->  John
Mark_Smith  ->  Mark
Anne_Smith  ->  Anne
RULES:
ManMarriedWith($x,$y)  ->  PMAP[ManMarriedWith](TABS $x)
WomanMarriedWith($x,$y)  ->  PMAP[WomanMarriedWith](TABS $x)
forall $x. $A  ->  forall IMAP[$x]. (WABS $A)
exists $x. $A  ->  exists IMAP[$x]. (WABS $A)
$A and $B  ->  (WABS $A) and (WABS $B)
$A or $B  ->  (WABS $A) or (WABS $B)
$A imp $B  ->  (WABS $A) imp (WABS $B)
$A iff $B  ->  (WABS $A) iff (WABS $B)
not $A  ->  not (WABS $A)
$P1($x)  ->  PMAP[$P1](TABS $x)
$P2($x,$y)  ->  PMAP[$P2](TABS $x,TABS $y)
$f1($x)  ->  FUMAP[$f1](TABS $x)
$f2($x,$y)  ->  FUMAP[$f2](TABS $x,TABS $y)
$x  ->  IMAP[$x]
;
```

5.3 Comandi di applicazione delle astrazioni

NOME

Abstract - applicazione di un'astrazione.

SINTASSI

Abstract *opzione oggetto* **by** *nome dell'astrazione*

DESCRIZIONE

Abstract rappresenta una famiglia di comandi. In generale applica una data astrazione ad un generico oggetto (simbolo, termine, formula, assioma ...), restituendo il corrispondente oggetto nel contesto astratto.

Si noti come in ogni chiamata venga effettuato un cambio di contesto. Questo significa che ogni comando di questo tipo può essere visto come una regola "ponte" tra due sistemi formali.

A seconda del tipo di oggetto mappato, lo spazio astratto può venire modificato o meno. I differenti tipi di oggetti che possono venire astratti, che corrispondono alle possibili opzioni del comando, vengono raccolti in tre gruppi principali:

- **Simboli dell'alfabeto**. Il risultato dell'astrazione di un simbolo viene *dichiarato* come appartenente all'alfabeto astratto, a meno che non vi sia già presente. Di conseguenza il contesto astratto viene modificato. Quest' opzione permette di generare i simboli dell'alfabeto astratto come immagine dei simboli di livello ground ($\mathcal{A}_2 := f_\Lambda(\mathcal{A}_1)$).
- **Espressioni logiche** Un termine [formula] viene "mostrata" nello spazio astratto come risultato dell'astrazione di un corrispondente termine [formula] di livello ground. In questo caso il contesto astratto non viene modificato.
- **Assiomi** Alcuni nuovi assiomi e/o teorie (cioè interi gruppi di assiomi) vengono introdotti nel contesto astratto come immagine di assiomi di livello ground. In questo senso è possibile (ma non obbligatorio) realizzare astrazioni di tipo Λ/Ω (cfr. def.4 di p.21), cioè tali che $\Omega_2 := f_\Lambda(\Omega_1)$.

Le opzioni del comando "**Abstract**" sono descritte in dettaglio nelle pagine seguenti. Si noti come, in tutti i casi previsti, la chiamata al comando venga effettuata dal contesto ground dell'astrazione utilizzata.

5.3.1 La generazione automatica dell'alfabeto astratto

NOME

abstract alphabet (e altri) - generazione automatica dell'alfabeto astratto.

SINTASSI

(i) **abstract alphabeth by** *nome dell'astrazione*
(ii) **abstract sentsyms by** *nome dell'astrazione*
...
abstract indsyms by *nome dell'astrazione*
abstract sentconsts by *nome dell'astrazione*
...
abstract indvars by *nome dell'astrazione*
(iii) **abstract sentconst** *sentconst* **by** *nome dell'astrazione*
...
abstract indvar *indvar* **by** *nome dell'astrazione*

DESCRIZIONE

Queste opzioni sono utilizzate per generare l'alfabeto astratto come immagine di quello ground. Il risultato dell'astrazione di un simbolo viene *dichiarato* come appartenente all'alfabeto astratto a meno che non vi sia già presente, nel qual caso viene prodotto un messaggio di warning.

I simboli possono venire mappati:

(i) **tutti insieme** (opzione "*alphabet*")
(ii) **in singoli gruppi** (opzioni "*sentsyms ... indvars*")
(iii) **singolarmente** (opzioni "*sentconst ... indvar*").

Si noti come l'uso di "**abstract alphabet**" assicuri la suriettività della funzione f_Λ sull'alfabeto stesso. Per questo motivo l'uso di tale opzione è fortemente raccomandato.

ESEMPIO

In quest'esempio e nei successivi faremo riferimento all'astrazione "*family*" definita a p.108 e successive.

Vediamo un esempio di generazione automatica dell'alfabeto astratto. Come unica osservazione, si noti come i simboli che non compaiono esplicitamente in "*family*" vengano mappati in una copia di sè stessi (ad esempio "*Anne_Hobbs*").

```
WORLD:: abstract alphabet by family;

There is no Sentconst to abstract.

There is no Sentpar to abstract.
```

```
You are now in the abstract context ‘SmithHome’.
Mary is declared to be an Indconst of sort UNIVERSAL.
John is declared to be an Indconst of sort UNIVERSAL.
Mark is declared to be an Indconst of sort UNIVERSAL.
Anne is declared to be an Indconst of sort UNIVERSAL.
Anne_Hobbs is declared to be an Indconst of sort UNIVERSAL.
Charles_Hobbs is declared to be an Indconst of sort UNIVERSAL.
William_Hobbs is declared to be an Indconst of sort UNIVERSAL.
Donna_Hobbs is declared to be an Indconst of sort UNIVERSAL.
You have moved back to the ground context ‘world’.

There is no Indpar to abstract.

You are now in the abstract context ‘SmithHome’.
x is declared to be an Indvar of sort UNIVERSAL.
y is declared to be an Indvar of sort UNIVERSAL.
z is declared to be an Indvar of sort UNIVERSAL.
You have moved back to the ground context ‘world’.

You are now in the abstract context ‘SmithHome’.
parent is declared to be a Predconst of arity 2.
parent has already been declared in ‘SmithHome’.
child is declared to be a Predconst of arity 2.
child has already been declared in ‘SmithHome’.
brother is declared to be a Predconst of arity 2.
brother has already been declared in ‘SmithHome’.
MarriedMan is declared to be a Predconst of arity 1.
MarriedWoman is declared to be a Predconst of arity 1.
You have moved back to the ground context ‘world’.

There is no Predpar to abstract.

There is no Funconst to abstract.

There is no Funpar to abstract.

WORLD::
```

5.3.2 L'applicazione di un'astrazione ad espressioni

NOME

abstract term - applicazione di un'astrazione a termini.
abstract wff - applicazione di un'astrazione a formule.

SINTASSI

abstract term *termine* **by** *nome dell'astrazione*
abstract wff *formula* **by** *nome dell'astrazione*

DESCRIZIONE

Quest'opzione "mostra" il risultato dell'applicazione di un'astrazione ad un termine/wff. Il contesto astratto non risulta essere modificato.

ESEMPIO

```
WORLD:: abstract term John_Smith by family;

You are now in the abstract context 'SmithHome'.
John
is the mapped term from 'world'.

SMITHHOME:: switchcontext world;
You are now using context: world.

WORLD:: abstract wff ManMarriedWith(John_Smith,Mary_Smith)
by family;

You are now in the abstract context 'SmithHome'.
MarriedMan(John)
is the mapped wff from 'world'.

SMITHHOME:: switchcontext world;
You are now using context: world.

WORLD:: abstract wff
(mother(x,y) and mother(x,z)) imp (brother(y,z) or sister(y,z))
by family;

You are now in the abstract context 'SmithHome'.
(parent(x,y) and parent(x,z)) imp (brother(y,z) or brother(y,z))
is the mapped wff from 'world'.

SMITHHOME::
```

5.3.3 L'applicazione di un'astrazione ad assiomi

NOME

abstract axiom - applicazione di un'astrazione ad assiomi.
abstract theory - applicazione di un'astrazione ad intere teorie.

SINTASSI

abstract axiom *nome dell'assioma* **by** *nome dell'astrazione* [**to** *nome del nuovo assioma*]
abstract theory *nome della teoria* **by** *nome dell'astrazione*

DESCRIZIONE

Quest'opzione permette ad assiomi ed intere teorie di essere generati a livello astratto come immagine di corrispondenti assiomi/teorie ground. Il nome di una teoria a livello astratto è definito per default concatenando il nome dell'astrazione con il nome della corrispondente teoria ground.

ESEMPIO

```
WORLD:: axiom sisterhood :
(mother(x,y) and mother(x,z)) imp sister(y,z);

WORLD:: theory SmithFamily :
mother(Mary_Smith,Mark_Smith)
mother(Mary_Smith,Anne_Smith)
father(John_Smith,Mark_Smith)
father(John_Smith,Anne_Smith)

WORLD:: abstract axiom sisterhood by family to brotherhood;
You are now in the abstract context 'SmithHome'.
brotherhood : (parent(x,y) and parent(x,z)) imp brother(y,z)
has been declared to be a new axiom in 'SmithHome'.

SMITHHOME:: switchcontext world;
You are now using context: world.

WORLD:: abstract theory SmithFamily by family;
You are now in the abstract context 'SmithHome'.
family.SmithFamily
family.SmithFamily1 : parent(Mary,Mark)
family.SmithFamily2 : parent(Mary,Anne)
family.SmithFamily3 : parent(John,Mark)
family.SmithFamily4 : parent(John,Anne)
has been declared to be a new theory in 'SmithHome'.

SMITHHOME::
```

5.4 Comandi di utilità per astrazioni

Nome

abswff - Verifica la correttezza sintattica di un'*abstract-wff*.
absterm - Verifica la correttezza sintattica di un *abstract-term*.

Sintassi

abswff *abstract-wff*
absterm *abstract-term*

Descrizione

Questi semplici comandi servono per verificare la correttezza sintattica di un'*abstract-expression*. In fase di dichiarazione di astrazioni e prototipi, la scrittura delle *abstract-espression* costituisce la maggiore difficoltà sintattica per utente (e quindi la maggiore sorgente di errori). L'uso di tali comandi può quindi essere utile, specialmente nel caso si debbano scrivere regole molto complesse. Si noti come un messaggio d'errore venga generato sia nel caso non vengano verificate le regole di ben formatezza della logica, sia nel caso vengano usati simboli sconosciuti.

Esempio

```
WORLD:: abswff PMAP[ManMarriedWith](TABS($x));
PMAP[ManMarriedWith](TABS $x) is an <abswff>.

WORLD:: absterm FUMAP[$f1](TABS($x));
FUMAP[$f1](TABS $x) is an <absterm>.

WORLD:: absterm FUMAP[$f2](TABS(John_Smith),TABS($y));
FUMAP[$f2](TABS John_Smith,TABS $y) is an <absterm>.

WORLD:: absterm FUMAP[$f2](TABS(Mario_Rossi),TABS($y));
absterm FUMAP[$f2](TABS Mario_Rossi,TABS $y);
                        ^
The parser requires a <term> here
```

In questo caso la costante individuale "*Mario_Rossi*" non appartiene all'alfabeto ground.

```
WORLD:: absterm  PMAP[ManMarriedWith](TABS($x));
absterm  PMAP[ManMarriedWith](TABS $x);
         ^
The parser requires an <absterm> here
```

In questo secondo caso l'espressione "*PMAP[ManMarriedWith](TABS $x)*" è un'*abstract-wff* e non un *abstract-term*.

5.5 Potenzialità, limiti e difetti

Abbiamo osservato nel capitolo 4 come la definizione di f_Λ data verifichi i requisiti che ci eravamo proposti, e permetta inlotre la definizione e l'uso dei prototipi di astrazione. Cerchiamo ora di approfondire l'analisi delle sue potenzialità, cercando di individuare anche i problemi che si possono manifestare.

Osservarviamo innanzitutto come esistano due possibili approcci alla definizione di un'astrazione:

1. **Approccio standard**: si istanzia un prototipo di libreria, definendo essenzialmente le mappe sull'alfabeto. In aggiunta, possono essere create delle semplici regole contesto-dipendenti, tipicamente utilizzate per gestire il comportamento "anomalo" di qualche simbolo o gruppo di simboli. Ad esempio:
$$MoglieDi(\$x) \longmapsto FIMAP[MoglieDi]$$
$$Intero(\$x)\ and\ Positivo(\$y) \longmapsto WABS(Intero(\$x)).$$
Si noti come questo approccio sia essenzialmente **dichiarativo**. Riteniamo che la grande maggioranza delle astrazioni verrà definita secondo questa modalità, in quanto essa risulta essere semplice e sufficientemente espressiva.

2. **Approccio "procedurale"** L'uso delle rule contesto-dipendenti permette di scrivere astrazioni anche molto complicate. Questa possibilità, le cui potenzialità non sono state ancora esplorate a sufficienza, estende la potenzialità espressiva del linguaggio di dichiarazione delle astrazioni, rendendolo però piú difficilmente controllabile. Tale approccio è perciò consigliabile solo ad un utente che abbia compreso molto bene il funzionamento del meccanismo di astrazione e che abbia ben chiari i concetti ed i pericoli dell'uso di regole ricorsive.

Cerchiamo ora di analizzare gli eventuali problemi che possono insorgere nella dichiarazione ed uso delle astrazioni.

Per prima cosa, la **suriettività** di f_Λ è una proprietà che si può facilmente eludere: infatti, una volta definita la mappa, è sufficiente dichiarare qualche nuovo simbolo nel contesto astratto. Tale limitazione non dipende però dalla definizione operativa di astrazione. Essa è una caratteristica intrinseca della nozione stessa di suriettività di una funzione, in quanto ogni estensione del codominio rende vana tale proprietà.

L'uso *disciplinato* dei comandi di dichiarazione è quindi l'unica soluzione possibile a tale inconveniente.

Un secondo problema riguarda la scarsa "robustezza" in fase di dichiarazione di un'astrazione. La richiesta di *assoluta generalità* ci ha costretto a ricorrere ad una formulazione del linguaggio di definizione piuttosto **permissiva**, in quanto consente ad un utente distratto o "indisciplinato" di creare effetti indesiderati.

- Con la nozione di f_Λ data un utente poco pratico di ricorsione potrebbe definire una funzione **non totale**, ad esempio

  $$\$x \longmapsto TABS(\$x).$$

  In tal caso, lo stack di sistema andrebbe rapidamente in overflow, interrompendo l'esecuzione.
- È inoltre possibile definire un'astrazione **non corretta**. Per esempio, un utente potrebbe definire la regola:

  $$MoglieDi(\$x) \longmapsto FIMAP[MoglieDi]$$

  e dimenticarsi di inserire la mappa corrispondente:

  $$MoglieDi \longmapsto DonnaSposata.$$

  Nella realizzazione in `GETFOL` è stato introdotto un test in fase di applicazione: se tale operazione genera un oggetto che non è un simbolo, un termine o una wff dello spazio astratto, il comando è annullato e viene stampato un messaggio d'errore.
- Un utente sbadato potrebbe dimenticarsi di scrivere qualche regola, dando origine ad un'astrazione **incompleta**. Nell'esempio di p.73, l'ommissione della regola:

  $$\$x \longmapsto IMAP[\$x]$$

  renderebbe impossibile l'astrazione dei sotto-termini costituiti da costanti individuali. Non potremmo, ad esempio, applicare l'astrazione alla formula atomica "$Operaio(Mario)$", in quanto non esisterebbe una regola in grado di astrarre il termine "$Mario$". In tale caso, usando l'interprete `GETFOL`, verrebbe prodotto un messaggio d'errore del tipo:

  ```
  There is no rule matching 'Mario'.
  ```

  e l'esecuzione del comando verrebbe bloccata.

Si noti come questi ultimi problemi non riguardino l'approccio "standard" nella scrittura di un'astrazione. Se l'utente usa i prototipi di libreria e, nel caso voglia scrivere delle nuove regole, si attiene alla disciplina di scrittura (peraltro piuttosto semplice), non si presentano inconvenienti. Utilizzando l'approccio "procedurale", come abbiamo già premesso, l'utente deve avere cognizione di causa di ciò che sta facendo.

In ogni caso, come accennato, la realizzazione nel sistema `GETFOL` prevede una serie di test in fase di applicazione, che, in caso di definizione scorretta, bloccano il comando e generano messaggi di errore.

Capitolo 6

La descrizione del codice

6.1 Introduzione

Nel capitolo 5 sono state illustrate le modalità d' uso dei comandi relativi alle astrazioni. In questo capitolo verrà brevemente descritto il codice che realizza tali comandi.

Una descrizione esaustiva del codice esula dagli scopi di questa trattazione. Per motivi di chiarezza e brevità, ci si è limitati ad descriverne solo le parti più significative, in modo da rendere comprensibili l'approccio seguito, le problematiche che si sono presentate e le relative soluzioni.

In particolare, verrà data una rapida descrizione della struttura dati "astrazione", e verranno presentate con sufficiente dettaglio le routine per l'astrazione delle espressioni logiche.

6.2 L'organizzazione del codice in `GETFOL`

Il sistema `GETFOL` [1] è realizzato in `HGKM` [2] [GC91], un linguaggio funzionale derivato da Common Lisp [Ste84]. Il codice è scritto seguendo una filosofia essenzialmente object-oriented, facente cioè largo uso di tipi di dati astratti.

Il sistema è strutturato in una serie di *moduli*, che ne realizzano le singole funzionalità. Ogni modulo è a sua volta ripartito in un gruppo di file. Il nome di ogni file include un'estensione (`<nome>.<estensione>`), la quale descrive il tipo di file a cui esso si riferisce.

I file "**.rp**" contengono le routine di parsing e di stampa. Il parser del sistema agisce secondo una strategia *top-down*, prevedendo la possibilità di *backtracking*.

[1] FOL è un acronimo per First Order Logic, GET per Genova, Edimburgh, Trento, le tre città in cui tale sistema è stato sviluppato.

[2] `HGKM` è un acronimo per Hilbert, Goedel, Kleene, McCarty.

Le routine di parsing si dividono in due categorie, a seconda di come si comportano in caso di fallimento :

(i) Parsing **obbligatorio**, realizzato dalle funzioni del tipo "`<dato>#`". Se la lettura ha successo, la funzione restituisce il dato richiesto, in caso contrario abortisce l'intero comando.

(ii) Parsing **opzionale**, realizzato dalle funzioni del *tipo* "`<dato>@`". Se la lettura ha successo, restituisce il dato letto. In caso contrario, genera un valore di fallimento "`FVAL`", senza abortire.

Consideriamo ad esempio il parser delle "*pmap*". La routine:

```
(DEFLAM PMAP# () (REQ# (QUOTE PMAP@) "The parser" "a <pmap>"))
```

attiva la corrispondente routine opzionale, "*PMAP@*". Se essa restituisce un valore di fallimento, la funzione "*REQ#*" abortisce l'intero comando e genera un messaggio d'errore. Per quanto riguarda il parser opzionale, la procedura

```
(DEFLAM PMAP@  () (MAP@ 'PREDSYM@ 'PREDRIGHTSYM#))
```

richiama a sua volta il parser opzionale "*PREDSYM@*" per leggere la parte sinistra della map. Si noti che, se tale chiamata ha successo, allora il sistema è *forzato* a leggere la parte rimanente della mappa dall'uso dei parser obbligatori "*PREDRIGHTSYM#*" e, precedentemente, dalla funzione "*ARROW#*" chiamata da "*MAP@*".

I file "**.hgk**" includono le funzioni per la gestione delle strutture dati, che, essendo viste come tipi di dati astratti, risultano accessibili solamente attraverso le loro specifiche primitive. Possiamo individuare tre principali tipi di primitive:

(i) I **costruttori**. Essi creano un tipo di dato strutturando le sue componenti. Per esempio, la funzione "*map-mak*" (cfr. file "declabs.hgk"):

    ```
    (DEFSUB map-mak (sym abssym) (CONS sym abssym))
    ```

    costruisce la generica struttura dati "*map*" dai suoi due componenti, "*sym*" (simbolo ground) , ed "*abssym*" (rappresentazione temporanea di un simbolo astratto). Un costruttore può a sua volta richiamare altre primitive, come nel caso:

    ```
    (DEFSUB abstraction-mak (name gndctx absctx flambda)
       (alistelement-mak name (LIST gndctx absctx flambda)))
    ```

    in cui "*alistelement-mak*" è a sua volta un costruttore per il dato astratto "elemento di una lista associativa".

(ii) I **distruttori**. Essi accedono alle singole componenti di un dato astratto. Ad esempio, le funzioni:

```
(DEFSUB map-get-gndsym (map) (CAR map))
(DEFSUB map-get-abssym (map) (CDR map))
```

accedono rispettivamente alle parti sinistra e destra di una generica map, mentre le routine:

```
(DEFSUB abstraction-get-name    (abstraction)
        (alistelement-get-1st abstraction))
(DEFSUB abstraction-get-body   (abstraction)
        (alistelement-get-2nd abstraction))
(DEFSUB abstraction-get-gndctx  (abstraction)
        (HGKM-NTH 1 (alistelement-get-2nd abstraction)))
(DEFSUB abstraction-get-absctx  (abstraction)
        (HGKM-NTH 2 (alistelement-get-2nd abstraction)))
(DEFSUB abstraction-get-flambda (abstraction)
        (HGKM-NTH 3 (alistelement-get-2nd abstraction)))
```

restituiscono le componenti principali di un'astrazione. L'uso disciplinato dei tipi di dato astratto impone, per esempio, di usare "(map-get-gndsym ...)" al posto di "(car ...)" ogni volta che si voglia accedere alla parte sinistra di una mappa.

(iii) I **test** ed altri. Oltre ai costruttori e distruttori veri e propri, esistono altre primitive di test e di manipolazione dei dati (inializzazione, cancellazione, aggiunta di nuovi elementi, conversione, ...). Per esempio, la funzione:

```
(DEFSUB abstraction-test (abstraction)
 (maplists-test (abstraction-get-maplists abstraction)))
```

effettua un test di correttezza sintattica sulla struttura dati "astrazione", mentre la routine:

```
(DEFLAM abslist-add-abs (abs)
   (SETQ %ABSLIST% (alist-add %ABSLIST%
                              (abstraction-get-name abs)
                              (abstraction-get-body abs))))
```

aggiunge un nuovo elemento alla lista globale delle astrazioni dichiarate.

I file "**.fol**" contengono, in generale, il codice "principale" per ogni comando o gruppo di comandi. In GETFOL i comandi sono realizzati in modo uniforme, seguendo due fasi ben distinte:

(i) Una routine di comando, che viene tipicamente contenuta in un file chiamato "cmd... .fol", legge la sequenza di istruzioni per mezzo di apposite routine di parsing.

(ii) L'informazione acquisita viene passata ad una routine "principale" (tipicamente contenuta in un file ".fol"), la quale svolge la funzionalità richiesta al comando stesso. A questo scopo tale routine si serve di funzioni di accesso alle strutture dati (file ".hgk"), di altre funzioni che svolgano compiti secondari (file ".fol") e di alcune routine di stampa per riprodurre i risultati e gli eventuali messaggi d'errore (file ".rp").

Consideriamo, per esempio, il comando "*abstract wff*" illustrato a p. 117, la cui sintassi è descritta nella forma:

abstract wff *formula* **by** *nome dell'astrazione.*

La lettura della sequenza di parole chiave "abstract wff" attiva la routine di comando (cfr. file "cmdabs.fol"):

```
(DEFLAM WFFABSTRACT# ()
 (LET ((wff (WFF#)))
  (SEQ
   (TKC# 'BY "BY")
   (LET ((absname (NAMEABS#)))
    (SEQ
     (ENDL)
     (wff-apply-abstraction wff absname))))))
```

la quale legge in sequenza la formula da astrarre, la parola chiave "*by*", il nome dell'astrazione usata ed il simbolo di fine istruzione, ";".

In seguito, essa chiama la routine principale (cfr. file "applyabs.fol"):

```
(DEFLAM wff-apply-abstraction (wff absname)
 (LET ((abstraction (abslist-get-abs absname)))
  (LET ((gndctx (abstraction-get-gndctx  abstraction))
        (absctx (abstraction-get-absctx  abstraction)))
   (IFN (EQUAL (curcname-get) gndctx)
    (ERRMESS "Ground context must be the same as current one.")
    (LET ((labelwff (expruleappl wff abstraction)))
     (SEQ
      (abs-ctx-switch absctx)
      (LET ((result (labexp2exp labelwff)))
       (IFN (WFF-REC result)
        (SEQ
         (PRINTMES3
           "The mapping of the ground wff is not a correct wff in `"
           absctx "'.")
         (gnd-ctx-switch gndctx))
        (SEQ
          (printexp result)
          (PRINTMES "")
          (PRINTMES3 "is the mapped wff from ground context `"
                     gndctx "'."))))))))))
```

la quale effettua l'astrazione della formula richiesta, mostrando il risultato nello spazio astratto. A tale fine essa utilizza un certo numero di funzioni ausiliarie.

Il funzionamento di questa routine verrà illustrato meglio in seguito.

6.3 Il problema della multi-contestualità

GETFOL è un sistema di deduzione **multicontestuale**, nel senso che l'utente può creare numerosi "ambienti" distinti, i *contesti*, all'interno dei quali egli può sviluppare delle deduzioni in modo indipendente. Ciascun contesto è dotato di un proprio linguaggio e di propri assiomi,che risultano inaccessibili da un contesto esterno. Esiste inoltre la possibilità di definire un contesto corrente, che rappresenta uno *stato* del sistema: se in un certo istante "*world*" è il contesto corrente, allora tutte le operazioni effettuate dall'utente andranno ad interessare solo *world*, e nessun oggetto appartenente ad altri contesti (simboli dell'alfabeto, assiomi, pezzi di dimostrazioni) potrà essere utilizzato o modificato. Sarà solo possibile "saltare" da un contesto all'altro tramite il comando "*switchcontext*". Per illustrare meglio tale concetto, riprendiamo l'esempio dell'introduzione, aggiungendo ulteriori informazioni:

Esempio 13

```
GETFOL:: setprompt to (curcname\-get);

NOTNAMED&:: namecontext AnticaGrecia;
You have named the current context: AnticaGrecia

AnticaGrecia:: declare indconst Socrate;
Socrate has been declared to be an Indconst

AnticaGrecia:: declare indvar x;
x has been declared to be an Indvar

AnticaGrecia:: declare predconst Uomo Animale 1;
Uomo has been declared to be a Predconst
Animale has been declared to be a Predconst

AnticaGrecia:: theory sillogismo :
Uomo(Socrate)
forall x.(Uomo(x) imp Animale(x));

sillogismo
sillogismo1 : Uomo(Socrate)
```

```
sillogismo2 : forall x. (Uomo(x) imp Animale(x))

AnticaGrecia:: makecontext Disneyland;
You have created the empty context: Disneyland

AnticaGrecia:: switchcontext Disneyland;
You are now using context: Disneyland

Disneyland:: declare indconst Pippo Pluto Paperino;
Pippo has been declared to be an Indconst
Pluto has been declared to be an Indconst
Paperino has been declared to be an Indconst

Disneyland:: term Socrate;
term Socrate;
     ^
The parser requires a <term> here

Disneyland:: andi sillogismo1 sillogismo2;
andi sillogismo1 sillogismo2;
     ^
The reference to a fact is required here.

Disneyland::
```

I comandi "*namecontext*", "*makecontext*" e "*switchcontext*" permettono, rispettivamente, di nominare contesti, di crearne di nuovi e di "saltare" da un contesto alll'altro.

All'interno dell'ambiente "Disneyland", il sistema non riconosce nè il simbolo "Socrate", nè gli assiomi "sillogismo1" e "sillogismo2", in quanto appartengono tutti al contesto "AnticaGrecia". Per questo motivo falliscono sia la chiamata "*term Socrate*" ('Socrate' è un termine?), sia il tentativo di applicare la regola d'inferenza "*and introduction*" ai due assiomi: "*andi sillogismo1 sillogismo2*". □

L'invisibiilità degli oggetti "esterni" al contesto corrente non riguarda soltanto l'interfaccia con l'utente, ma è una caratteristica intrinseca del codice di GETFOL: nessuna routine può accedere a simboli ed altri oggetti esterni al contesto corrente senza effettuare un salto di contesto.

In GETFOL un simbolo logico è rappresentato mediante un riferimento ad una tabella, all'interno della quale gli viene associata una certa quantità di informazione, come il proprio *nome* ("*label*"), il *tipo* sintattico, l'eventuale *arità* ecc. Tale riferimento è inaccessibile all'esterno del contesto di appartenenza del simbolo stesso.

Questa proprietà di invisibilità dei simboli ha causato grossi problemi nella realizzazione dei comandi per l'astrazione, sia nella fase *definizione*, sia per quanto riguarda l'*applicazione*.

(i) Secondo la definizione di f_Λ introdotta nei capitoli precedenti, una "map" è costituita da una coppia

$$\textit{simbolo ground} \longmapsto \textit{simbolo astratto}.$$

Per accedere al simbolo di sinistra, la struttura dati "astrazione" dev'essere quindi utilizzabile solo a livello ground. In questo modo, però, il simbolo astratto non è più accessibile. Abbiamo infatti unito in un unica struttura dati (la "map") due simboli che possono essere "visti" solo da contesti diversi.

Per risolvere questo problema, è stata introdotta una nuova struttura dati, detta "*rightsymbol*" (cfr. file "declabs.hgk"), costituita dal nome ("*label*") di un simbolo astratto e da tutta l'informazione ad esso relativa. Tale struttura, che rappresenta a tutti gli effetti un simbolo dell'alfabeto astratto, viene a costituire la parte destra di ogni "map".

(ii) Nell'atto di astrarre un'espressione logica, ad esempio una formula φ, ci troviamo a dover costruire una rappresentazione di $f_\Lambda(\varphi)$ all'interno del contesto ground. I simboli occorrenti in $f_\Lambda(\varphi)$ sono però "invisibili" da tale contesto.

Per questo motivo si è introdotta la nozione di "*labelled expression*", una particolare rappresentazione di un'espressione in cui ogni riferimento ad un simbolo è sostituito con il rispettivo nome ("*label*"). Nell'atto di astrazione di φ, viene creata per prima cosa una rappresentazione "labelled" di $f_\Lambda(\varphi)$; viene poi effettuato un salto al contesto astratto, dopodichè la rappresentazione "labelled" di $f_\Lambda(\varphi)$ viene convertita in un'espressione vera e propria, sostituendo ogni "*label*" con il riferimento al simbolo astratto che essa rappresenta.

I "*rightsyms*" e le "*labelled expression*" costituiscono entrambi delle strutture dati temporanee, utilizzate per rappresentare oggetti astratti a livello ground. Poiché vengono usate per poter "passare la frontiera" tra i due contesti, nella descrizione del codice esse vengono chiamate "*smuggling data structures*" [3].

6.4 La dichiarazione di un'astrazione

Nel capitolo 5 abbiamo descritto l'*uso* dei comandi per dichiarare un'astrazione. Vogliamo ora analizzare brevemente il processo stesso di dichiarazione, illustrando il funzionamento del comando "*declare*" e descrivendo per sommi capi

[3] In inglese "*to smuggle*" significa "*contrabbandare*".

la struttura dati utilizzata.

Secondo la definizione introdotta a pag.81, la struttura dati "astrazione" è composta da:

(i) *un nome;*
(ii) *il nome del contesto ground;*
(iii) *il nome del contesto astratto;*
(iv) *una funzione f_Λ tra i due linguaggi.*

Un'astrazione può essere dichiarata "direttamente", tramite il comando "*abstraction declare*" (cfr. p. 99):

Abstraction declare *nome dell'astrazione*
    ground
        nome del contesto ground
    abstract
        nome del contesto astratto
    flambda
        descrizione della funzione tra i linguaggi.

La sequenza di parole chiave "*abstraction declare*" richiama la routine di comando "*ABSDECLARE#*" (cfr. file "cmdabs.fol"):

```
(DEFLAM ABSDECLARE# ()
 (LET ((name (NAMEABSDEF#)))
  (LET ((gndctx (GNDCTX#)))
   (LET ((absctx (ABSCTX#)))
    (LET ((flambda (FLAMBDA#)))
       (SEQ
        (ENDL)
        (newabs (abstraction-mak name gndctx absctx flambda)))))))),
```

la quale legge sequenzialmente il nome dell'astrazione, i nomi dei due contesti, la descrizione della funzione tra i linguaggi ed il simbolo di fine comando ";". A tale scopo essa utilizza una serie di apposite routine di parsing (cfr. file "declabs.rp").

A questo punto l'astrazione corrispondente viene costruita dalle sue componenti (funzione "*abstraction-mak*"), e dichiarata nell'apposita struttura dati globale dalla routine "*newabs*", la quale effettua un'ulteriore test di correttezza sintattica.

I comandi per la dichiarazione "indiretta" di un'astrazione seguono uno schema di funzionamento analogo.

Il *nome dell'astrazione* dev'essere nuovo, nel senso che non deve essere già stata dichiarata alcuna astrazione con lo stesso nome.

Per quanto riguarda i *nomi dei contesti*, quello ground deve riferirsi al contesto corrente all'atto di definizione, mentre quello astratto deve riferirsi ad uno già esistente, eventualmente vuoto.

La parte più importante di un'astrazione è costituita dalla *funzione tra i due linguaggi*, f_Λ. Come illustrato nei precedenti capitoli, la rappresentazione di f_Λ è ripartita in due distinte funzionalità: una funzione tra gli alfabeti dei due contesti, definita attraverso sei tipi diversi di "map", e la mappa sulla struttura delle espressioni logiche, gestita tramite le "rules".

- Una "map" è rappresentata internamente come una coppia

  $$\textit{simbolo ground} \longmapsto \textit{rightsymbol},$$

  dove con "*rightsymbol*" intendiamo la struttura dati temporanea già illustrata in questo capitolo. Ad esempio, una "smap" viene internamente rappresentata nella forma:

  $$\textit{simbolo proposizionale ground} \longmapsto \langle \textit{tipo sintattico}, \textit{nome}, \textit{speech act} \rangle,$$

  dove con "*speech act*" si intende una certa quantità di informazione, utile al sistema in fase di stampa del simbolo.
- Una "rule" è anch'essa una coppia, del tipo:

  $$\textit{espressione ground} \longmapsto \textit{abstract expression}.$$

  La nozione di "*abstract expression*"è stata sufficientemente illustrata nel cap.4: sostanzialmente, essa rappresenta una forma ricorsiva adatta a descrivere una qualche espressione di livello astratto, senza includervi i simboli dichiarati a tale livello.

  All'insieme delle rules viene associata una particolare struttura dati di `GETFOL`, detta "*Associative Memory Formula*" (amf). Quest'ultima permette di utilizzare un'espressione logica come *chiave* per reperire l'informazione ad essa associata, nel nostro caso un'abstract-expression.
  Una descrizione approfondita della nozione di amf, che esula dagli scopi di questa trattazione, può essere trovata in [Pec90, GP91].

6.5 L'astrazione delle espressioni logiche

Abbiamo introdotto nel capitolo 4 una descrizione dell'algoritmo di astrazione delle espressioni logiche (cfr. p. 88). All'inizio di questo capitolo abbiamo inoltre illustrato i problemi relativi alla multi-contestualità del sistema, proponendo una soluzione basata sull'utilizzo di strutture dati temporanee, i "*rightsymbol*" e le "*labelled expression*". Analizziamo ora il funzinamento dei comandi relativi all'applicazione di astrazioni alle espressioni logiche.

6.5.1 I comandi

Consideriamo per esempio il comando "abstract wff" di p. 117, la cui sintassi è descritta nella forma:

abstract wff *formula* **by** *nome dell'astrazione.*

Analizziamo il comportamento di tale comando applicato ad una generica formula φ. La lettura della sequenza di parole chiave "*abstract wff*" attiva la routine di comando "*WFFABSTRACT#*" (cfr. file "cmdabs.fol"):

```
(DEFLAM WFFABSTRACT# ()
 (LET ((wff (WFF#)))
  (SEQ
   (TKC# 'BY "BY")
   (LET ((absname (NAMEABS#)))
    (SEQ
     (ENDL)
     (wff-apply-abstraction wff absname)))))),
```

la quale legge in sequenza la formula φ da astrarre, la parola chiave "*by*", il nome dell'astrazione usata ed il simbolo di fine istruzione ";". A questo punto, possiede tutta l'informazione per chiamare la routine principale "*wff-apply-abstraction*" (cfr. file "applyabs.fol"):

```
(DEFLAM wff-apply-abstraction (wff absname)
 (LET ((abstraction (abslist-get-abs absname)))
  (LET ((gndctx (abstraction-get-gndctx  abstraction))
        (absctx (abstraction-get-absctx  abstraction)))
   (IFN (EQUAL (curcname-get) gndctx)
    (ERRMESS "Ground context must be the same as current one.")
    (LET ((labelwff (expruleappl wff abstraction)))
     (SEQ
      (abs-ctx-switch absctx)
      (LET ((result (labexp2exp labelwff)))
       (IFN (WFF-REC result)
        (SEQ
         (PRINTMES3
           "The mapping of the ground wff is not a correct wff in `"
           absctx "'.")
         (gnd-ctx-switch gndctx))
        (SEQ
          (printexp result)
          (PRINTMES "")
          (PRINTMES3 "is the mapped wff from ground context `"
                     gndctx "'."))))))))))).
```

In primo luogo, essa controlla che il nome del contesto ground corrisponda con quello corrente. Dopodichè chiama il meccanismo di mapping delle espressioni (funzione "*expruleappl*", cfr. file "rew.fol") per generare, secondo l'algoritmo visto nel cap. 4, una rappresentazione "labelled" di $f_{\Lambda}(\varphi)$. Il funzionamento di *expruleappl* verrà descritto nella sezione 6.5.2.

A questo punto viene effettuato il "salto"di contesto (routine "*abs-ctx-switch*"), dove la "labelled wff" può essere finalmente convertita in una formula del linguaggio astratto dalla funzione "*labexp2exp*". Un ultimo test sulla correttezza della nuova espressione viene effettuato dalla funzione "*WFF-REC*", dopodichè la formula risultante viene mostrata all'utente.

Riprendendo l'esempio di p. 91, si ha:

```
World:: ABSTRACT WFF Maestro(padre(Mario)) and Operaio(Paolo_Bianchi)
        BY FirstExample;
You are now in the abstract context 'AbstractWorld'.
Insegnante(genitore(Mario)) and Operaio(Paolo)
is the mapped wff from ground context 'World'.

AbstractWorld::
```

Il funzionamento del comando "*abstract term*" è assolutamente analogo a quello visto. Anche i comandi "*abstract axiom*" ed "*abstract theory*" funzionano allo stesso modo; come unica differenza, ogni formula risultante non solo viene *mostrata* all'utente, ma viene anche *dichiarata* come un nuovo assioma del contesto astratto.

6.5.2 Il mapping delle espressioni

Nel capitolo 4 è stato presentato l'algoritmo di mapping delle espressioni. Descriveremo ora, per sommi capi, le modalità con cui esso è stato realizzato in GETFOL. Tutto il codice relativo è contenuto del file "absrew.fol".

Riprendiamo in considerazione lo schema a blocchi del capitolo 4, trascritto in fig. 6.1 [4]. All'interno dello schema distinguiamo quattro blocchi fondamentali, che corrispondono a quattro distinte funzionalità.

Un blocco "**top level**" riceve in ingresso l'espressione logica "ground", exp_g, e restituisce la corrispondente espressione astratta exp_a. Tale azione è svolta dalla funzione "*expruleappl*":

```
(DEFLAM expruleappl (exp abstraction)
```

[4] Contrariamente alla figura di p.88, qui i nomi delle funzioni sono effettivamente quelli utilizzati nel codice.

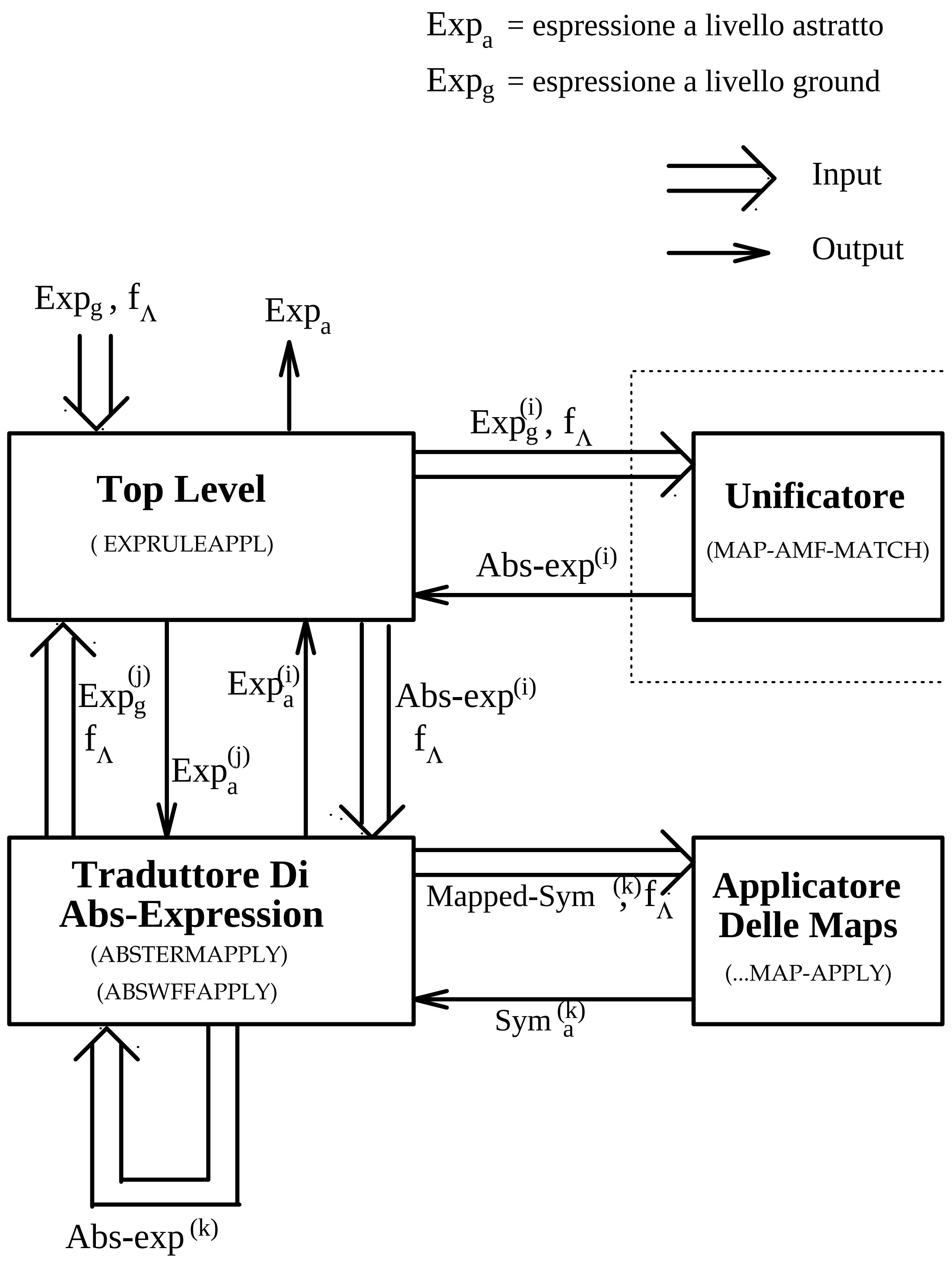


Figura 6.1: Schema dell'algoritmo di mapping delle espressioni.

```
 (LET ((amf (abstraction-get-amf abstraction)))
  (LET ((rr (map-amf-match exp amf
                   (abstraction-get-maplists abstraction))))
   (IFN (MATCH? rr)
    (ERRMES2-printexp "There is no rule matching ‘" exp "’.")
    (LET ((rightexp (rr-get-exp rr)))
     (IF (TERM exp)
      (abstermapply rightexp abstraction)
      (abswffapply rightexp abstraction))))))).
```

In primo luogo, essa riceve in ingresso exp_g e la passa, mediante la funzione "*map-amf-match*", ad un gruppo di **routine di unificazione**, pre-esistenti in GETFOL.

Queste utilizzano la struttura "*amf*" dell'astrazione per cercare una regola la cui parte sinistra unifichi con exp_g. Se non la trovano, "*map-amf-match*" genera un codice di fallimento, altrimenti restituisce l'abstract-expression *absexp* relativa ad exp_g, con i parametri "$" debitamente sostituiti. Per un'analisi più approfondita della struttura delle "*amf*" e del funzionamento delle routine di unificazione, si rimanda a [Pec90, GP91].

Se l'unificazione ha avuto successo, *absexp* viene a sua volta passata ad un **traduttore di abstract expression**, il quale restituirà al top level l'espressione mappata, exp_a. Quest'ultimo meccanismo viene realizzato tramite un gruppo di funzioni mutualmente ricorsive, e viene chiamato dal top level tramite le routine "*abstermapply*" ed "*abswffapply*". Per brevità di esposizione, analizziamo solo l'ultima delle due, in quanto il funzionamento di "*abstermapply*" risulta essere analogo.

```
(DEFLAM abswffapply (rightexp abstraction)
 (IF (WABS-APPLICATION? rightexp)
  (expruleappl (WABS-eliminate rightexp) abstraction)
  (IF (AABSWFF rightexp)
   (aabswffapply rightexp abstraction)
   (IF (QUANTABSWFF rightexp)
    (quantabswffapply rightexp abstraction)
    (complexpruleappl rightexp abstraction)))))
```

Il funzionamento di "*abswffapply*" è relativamente semplice.

Se la formula passata è del tipo " (*WABS wff*)", viene richiamata la routine di top level, passandole in ingresso la formula ground "*wff*".

In alternativa, *absexp* può essere atomica, quantificata o ottenuta per applicazione dei connettivi proposizionali. In tali casi, essa verrà passata come ingresso, rispettivamente, alle funzioni "*aabswffapply*", "*quantabswffapply*" o "*complexpruleappl*". Anche qui, per non appesantire troppo la trattazione, analizziamo solo la prima delle tre routine:

```
(DEFLAM aabswffapply (rightexp abstraction)
 (IF (SIMPLE rightexp)
  (IF (SMAP? rightexp)
    (smap-apply rightexp abstraction)
    (psmap-apply rightexp abstraction))
  (IF (ABSEQU rightexp)
   (predappl-mak
      (appl-get-functor rightexp)
      (abstermlistapply (predappl-get-args rightexp) abstraction))
   (predappl-mak
      (pmap-apply (appl-get-functor rightexp) abstraction)
      (abstermlistapply (predappl-get-args rightexp) abstraction))))).
```

absexp può essere "*semplice*", cioè scritta nelle forme:

$$\text{“}SMAP[\textit{simbolo proposizionale}]'' \quad o \quad \text{“}PSMAP[\textit{predicato}]''.$$

In tal caso vengono chiamate le rispettive routine di applicazione delle map ("*smap-apply*" o "*psmap-apply*"), le quale restituiscomo il simbolo proposizionale astratto corrispondente.

Alternativamente, *absexp* può essere nella forma "$abs\text{-}term_1 = abs\text{-}term_2$", nel qual caso il risultato è dato da:

$$term_{a_1} = term_{a_2},$$

dove $term_{a_1}$ e $term_{a_2}$ sono i risultati della "traduzione" di *abs-term*$_1$ ed *abs-term*$_2$, ottenuti mediante due chiamate ricorsive alla routine "*abstermapply*".

Come ultima possibilità, *absexp* può essere nella forma

$$PMAP[P_g](absterm_1, \ldots, absterm_n),$$

dove P_g è un generico predicato ground. In questo caso, risulterà:

$$P_a(term_{a_1}, \ldots, term_{a_n}),$$

dove "P_a" è ottenuto dall'applicazione delle mappe sui predicati a P_g (funzione "*pmap-apply*"), ed il generico termine astratto $term_{a_i}$ viene ottenuto, come prima, "traducendo" *abs-term*$_i$.

L'ultima funzionalità del sistema è data dalle **funzioni di applicazione delle mappe**. Dato un "mapped-symbol" nella forma "*...MAP[sym]*", esse restituiscono l'immagine "astratta" del simbolo *sym*. Tali funzioni sono piuttosto semplici, in quanto sono formate da sequenze di distruttori di strutture dati.

Per esempio, consideriamo la funzione applicatrice delle mappe tra predicati:

```
(DEFSUB pmap-apply (mappred abstraction)
 (maplists-get-abspred
    (mappred2pred mappred)
    (flambda-get-maplists
      (abstraction-get-flambda abstraction)))).
```

In sequenza, "*abstraction-get-flambda*" preleva da un'astrazione la rappresentazione di f_Λ ; "*flambda-get-maplists*" prende a sua volta la lista delle mappe; "*mappred2pred*" preleva il predicato ground $pred$ dall'interno ad un "mapped symbol" nella forma "*PMAP[pred]*"; infine, dati il predicato ground $pred$ la lista delle mappe, "*maplists-get-abspred*" preleva la "label" del predicato astratto $f_\Lambda(pred)$.

6.6 L'organizzazione del codice per le astrazioni

Riportiamo infine una breve descrizione dell'organizzazione del codice dei comandi di astrazione.

Secondo le convenzioni del sistema `GETFOL` illustrate nella sezione 6.2, il codice è strutturato in una serie di file:

- **cmdabs.fol**. Vi si possono trovare tutte le routine di comando per l'astrazione. In particolare, il codice viene qui ripartito in comandi di *dichiarazione*, *applicazione* ed *utilità*, secondo lo schema illustrato nel capitolo 5.
- **declabs.rp**. Vi sono contenute tutte le funzioni di parsing e stampa utilizzate nella dichiarazione delle astrazioni, in particolare quelle relative ai vari componenti della struttura dati "astrazione". Come unica eccezione, le funzioni per la lettura e la stampa delle "*abstract-expression*" sono riportate in un file a parte.
- **declabs.hgk** Contiene tutte le primitive per accedere alla struttura dati "astrazione". In particolare citiamo i costruttori, i distruttori, i test e le funzioni di manipolazione di tutte le sue strutture interne (map, rule, "rightsymbol" ecc.), oltre alle routines di accesso alle strutture dati globali.
- **absexp.rp** Il parser per le strutture dati "abstract-expression", e le relative procedure di stampa sono contenute in questo file. Esso è stato ottenuto modificando il parser delle espressioni logiche di `GETFOL` (cfr. file "exp.rp").
- **absexp.hgk** Vi si possono trovare una piccola quantità di primitive di accesso alle "abstract-expression".
- **applyabs.fol** Contiene le procedure, principali ed ausiliarie, relative ai comandi di *applicazione* delle astrazioni. Tra queste distinguiamo quelle relative alla generazione automatica del linguaggio astratto ("*abstract alphabet*" ed altri) da quelli di astrazione delle espressioni logiche e degli assiomi.

- **absrew.fol** Contiene la realizzazione dell'algoritmo di mapping delle espressioni logiche, come è stato descritto nella sezione 6.5.2. I quattro distinti blocchi dello schema di pag. 141 corrispondono alle quattro distinte sezioni in cui viene ripartito questo file.
- **iabs.fol** È il file di inizializzazione. Contiene tutte le procedure per inizializzare i comandi e le strutture dati descritte.

Questa ripartizione corrisponde allo stato del sistema relativo al novembre 1991. Il codice scritto viene allegato come appendice alla tesi.

Capitolo 7

Conclusioni

7.1 Risultati raggiunti e relativi limiti

Con il lavoro di questa tesi si è definita e realizzata la prima parte di un sistema di deduzione automatica fondato sull'astrazione. In particolare, sono stati implementati una serie di comandi, all'interno del sistema `GETFOL`, per *definire* e *applicare* astrazioni nella dimostrazione automatica e/o interattiva di teoremi.

Come premessa a tale lavoro, si è resa necessaria l'analisi teorica di una serie di problematiche legate alla nozione di funzione tra linguaggi del primo ordine (f_Λ). Questa analisi ha portato alla definizione di un particolare *linguaggio* per definire tali funzioni, e alla concezione di un possibile *interprete* per tale linguaggio.

Questi concetti sono poi stati applicati nella realizzazione dei comandi sopra menzionati.

Per quanto riguarda la fase di *dichiarazione*, l'introduzione del concetto di *prototipo* permette di definire un'astrazione in modo semplice e veloce. La possibilità di scrivere regole dipendenti da contesto consente inoltre, ad un utente più esperto, di creare astrazioni anche relativamente complesse. Infine, nonostante la generalità della definizione, la dichiarazione di una nuova astrazione risulta essere piuttosto semplice ed intuitiva.

Un'altra importante conseguenza della definizione di f_Λ introdotta consiste nella realizzazione di una doppia funzionalità dei comandi di applicazione delle astrazioni:

(i) è possibile creare automaticamente l'alfabeto del contesto astratto come immagine di quello ground: $\mathcal{A}_2 := f_\Lambda(\mathcal{A}_1)$. Questa opportunità velocizza notevolmente il processo generale di astrazione.

(ii) È possibile mappare termini, formule, assiomi ed intere teorie nello spazio astratto.

Nel Capitolo 5 sono stati illustrati alcuni problemi che si possono verificare nella fase di dichiarazione. Riassumendo brevemente, un utente inesperto che volesse definire astrazioni complesse in modo "diretto" (cioè senza utilizzare prototipi) e senza attenersi alle tecniche di scrittura delle "rule" descritte nel manuale, potrebbe creare delle astrazioni "scorrette", cioe' tali da generare espressioni sintatticamente scorrette nello spazio astratto. Come si è visto nel Capitolo 5, questo limite non dipende da una scarsa robustezza del codice, ma è intrinsecamente legato alla nozione di f_Λ introdotta.

Si noti però che, se è teoricamente possibile definire astrazioni "scorrette", sicuramente tali astrazioni non possono essere applicate con successo: una serie di test interni impediscono di produrre risultati nello spazio astratto che non siano espressioni logiche corrette sintatticamente.

Nel Capitolo 3 è stata introdotta una trattazione teorica che, pur non indispensabile per la realizzazione dei comandi visti, costituisce un supporto teorico per la futura evoluzione del sistema. In particolare, è stata introdotta una nuova definizione formale di astrazione, estesa a sistemi formali multisorte. Si è poi dimostrato (cfr. Teoremi 4, 5 e 6) come tale definizione estesa conservi, nel caso di sistemi "Sorted Natural Deduction", certe proprietà fondamentali delle astrazioni tra sistemi "ordinari".

Tale trattazione, elaborata sotto la guida degli stessi Giunchiglia e Walsh, costituisce un piccolo contributo teorico alla teoria dell'astrazione.

7.2 Sviluppi futuri

L'analisi del lavoro svolto, dei problemi incontrati e dei risultati ottenuti permette di tracciare un quadro del lavoro futuro necessario al completamento del modulo "*abstract theorem prover*".

Da un punto di vista prettamente teorico, lo studio iniziato nel Capitolo 3 potrà essere ulteriormente ampliato cercando, per esempio, di estendere i risultati visti, come il Teorema 6, ad altri tipi di sistemi multisorte (es. "Sorted Resolution"). Si dovrà quindi realizzare effettivamente un'estensione dei comandi `GETFOL` visti, per permettere di astrarre anche tutta l'informazione relativa ai simboli di sorte.

Nonostante le funzionalità realizzate siano di per sè autonome, alcune ulteriori aggiunte al sistema potranno facilitarne l'utilizzo da parte dell'utente. Tra queste, la creazione di una libreria di prototipi interna al sistema renderà estremamente agevole l'azione di creazione di nuove astrazioni.

Il lavoro futuro più consistente rimane ovviamente quello di definire e realizzare i comandi per gestire le fasi di "mapping back" e rifinitura (cfr. p.31). Per questo compito sarà necessaria una quantità di lavoro simile, se non maggiore, a quello svolto fino adesso. L'analisi formale del problema, contenuta in [GW90a], costituirà il supporto teorico principale per tale lavoro.

7.3 Prospettive

Come premessa a questa sezione, notiamo che, quando si parla di *theorem proving*, non ci si limita a descrivere sistemi utili ai matematici per dimostrare dei teoremi. Si parla invece, molto più in generale, di sistemi che permettano di dedurre dei risultati da un insieme di premesse, mediante l'applicazione di opportune regole di inferenza. In questo modo, la nozione di theorem proving cattura anche molti campi che interessano vari settori dell'Intelligenza Artificiale, dell'informatica e dell'ingegneria, come planning, teoria della decisione, hardware verification, progettazione "logica" di Data Base ecc. Per citare un esempio, un interprete Prolog può essere considerato un theorem prover.

In questo senso si possono intravvedere varie prospettive di utilizzo del sistema di astrazione realizzato in `GETFOL`.

In primo luogo si pensa di poter utilizzare il sistema in modo prettamente "logico matematico", cioè per dimostrare un certo numero di teoremi matematici, anche complessi. Questo permetterà anche di verificare le effettive potenzialità dell'idea di "*abstract theorem proving*" dal punto di vista dell'efficienza computazionale. Dati i risultati della teoria e quelli ottenuti per simulazione (ad es. cfr. [F.]), le prospettive a riguardo si rivelano incoraggianti.

In secondo luogo, si potrà estendere tale indagine ad applicazioni dei meccanismi di astrazione in altri campi, come planning, query languages, teoria della decisione. In questo modo si pensa di poter indagare ulteriormente, in modo effettivo, sulle potenzialità dell'uso di meccanismi di astrazione in questi campi.

Infine, l'esperienza nell'uso del sistema potrà aiutare a elaborare delle possibili euristiche per la scelta delle astrazioni adatte alla risoluzione dei vari problemi.

Bibliografia

[Bor84] M. Borga. *Elementi di logica matematica*. La Goliardica, 1984.

[CG90] A. Cimatti and F. Giunchiglia. A proof-theoretic characterization of GETFOL's logic, 1990. IRST Internal Report no.9004-02.

[Cha79] C.L. Chang. Resolution plans in theorem proving. In *Proc. of the 6th International Joint Conference on Artificial Intelligence*, pages 143–148, 1979.

[F.] substituted by giunchiglia32.

[GAC+91] F. Giunchiglia, A. Armando, A. Cimatti, E. Giunchiglia, P. Pecchiari, L. Serafini, A. Simpson, and P. Traverso. GETFOL Programmer Manual - GETFOL version 1. Technical Report 91-0007, DIST - University of Genova, Genova, Italy, 1991.

[GC91] F. Giunchiglia and A. Cimatti. HGKM User Manual - HGKM version 2. Technical Report 91-0009, DIST - University of Genova, Genova, Italy, 1991.

[Gen69] G. Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–128. North Holland Publishing Company, 1969.

[GG88] F. Giunchiglia and E. Giunchiglia. Building complex derived inference rules: a decider for the class of prenex universal-existential formulas. In *Proc. 7th European Conference on Artificial Intelligence*, pages 607–609, 1988. Extended version available as DAI Research Paper 359, Dept. of Artificial Intelligence, Edinburgh.

[Giu] removed Feb. 92.

[GP91] F. Giunchiglia and P. Pecchiari. Riscrittura sintattica in GETFOL. Technical Report 9105-23, IRST, Trento, Italy, 1991.

[GT91] F. Giunchiglia and P. Traverso. `GETFOL` User Manual - `GETFOL` version 1. Manual 9109-09, IRST, Trento, Italy, 1991. Also DIST Technical Report 91-0005, DIST, University of Genova.

[GW89a] F. Giunchiglia and T. Walsh. Abstract Theorem Proving. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 372–377, 1989. Also IRST-Technical Report 8902-03 and DAI Research Paper No 430, University of Edinburgh.

[GW89b] F. Giunchiglia and T. Walsh. Abstracting into inconsistent spaces (or the false proof problem). In *Proc. AI*IA 89*, 1989. Also IRST-Technical Report 8904-02 and DAI Research Paper no. 514, University of Edinburgh.

[GW89c] F. Giunchiglia and T. Walsh. Theorem Proving with Definitions. In *Proc. of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 175–183, 1989. Also IRST-Technical Report 8901-03 and DAI Research Paper No 429, University of Edinburgh.

[GW90a] F. Giunchiglia and T. Walsh. Abstract theorem proving: Mapping back. Technical Report 8911-16, IRST, Trento, Italy, 1990. Also DAI Research Paper no. 460a, University of Edinburgh.

[GW90b] F. Giunchiglia and T. Walsh. Abstraction in AI. *AISB Quarterly*, 73:22–26, 1990.

[GW90c] F. Giunchiglia and T. Walsh. The use of abstraction in automatic inference. In *Proc. UK Conference on Informatic Technology*, 1990. Also IRST-Technical Report 8909-11 and DAI Research Paper, University of Edinburgh.

[GW91] F. Giunchiglia and T. Walsh. Using abstraction. In *Proc. of the 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, Leeds, UK, 1991. Also IRST-Technical Report 9010-08 and DAI Research Paper 515, University of Edinburgh.

[GW93] F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. *Journal of Automated Reasoning*, 11:23–41, 1993. Also IRST-Technical Report 9006-16, IRST, Trento, Italy.

[Hob85] J.R. Hobbs. Granularity. In *Proc. of the 9th International Joint Conference on Artificial Intelligence*, pages 432–435, 1985.

[Imi87] T. Imielinski. Domain abstraction and limited reasoning. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pages 997–1003, 1987.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[Mel87] T.F. Melham. Abstraction mechanisms for hardware verification. Technical Report 106, University of Cambridge, Computer Laboratory, 1987.

[Nil80] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co., 1980.

[Pec90] P. Pecchiari. Meccanizzazione del concetto di modello di un dimostratore interattivo, 1990. IRST-Thesis 9009-15.

[Pla80] D.A. Plaisted. Abstraction mappings in mechanical theorem proving. In *5th Conference on Automated Deduction*, pages 264–280, 1980.

[Pla81] D.A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[Pra65] D. Prawitz. *Natural Deduction - A proof theoretical study*. Almquist and Wiksell, Stockholm, 1965.

[Rob65] J. A. Robinson. A Machine oriented Logic Based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Sac74] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Ste84] Guy L. Steele. *COMMON LISP - The language*. Digital Press, 1984.

[Ten87] J.D. Tenenberg. Preserving Consistency across Abstraction Mappings. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pages 1011–1014, 1987.

[Ten88] J.D. Tenenberg. *Abstraction in Planning*. PhD thesis, Computer Science Department, University of Rochster, 1988. Also TR 250.

[Wal90] T. Walsh. *A Theory of Abstraction*. PhD thesis, University of Edinburgh, 1990.

[Wey77] R.W. Weyhrauch. A Users Manual for `FOL`. Technical Report STAN-CS-77-432, Computer Science Department, Stanford University, 1977.

[Wey80] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13(1):133–176, 1980.