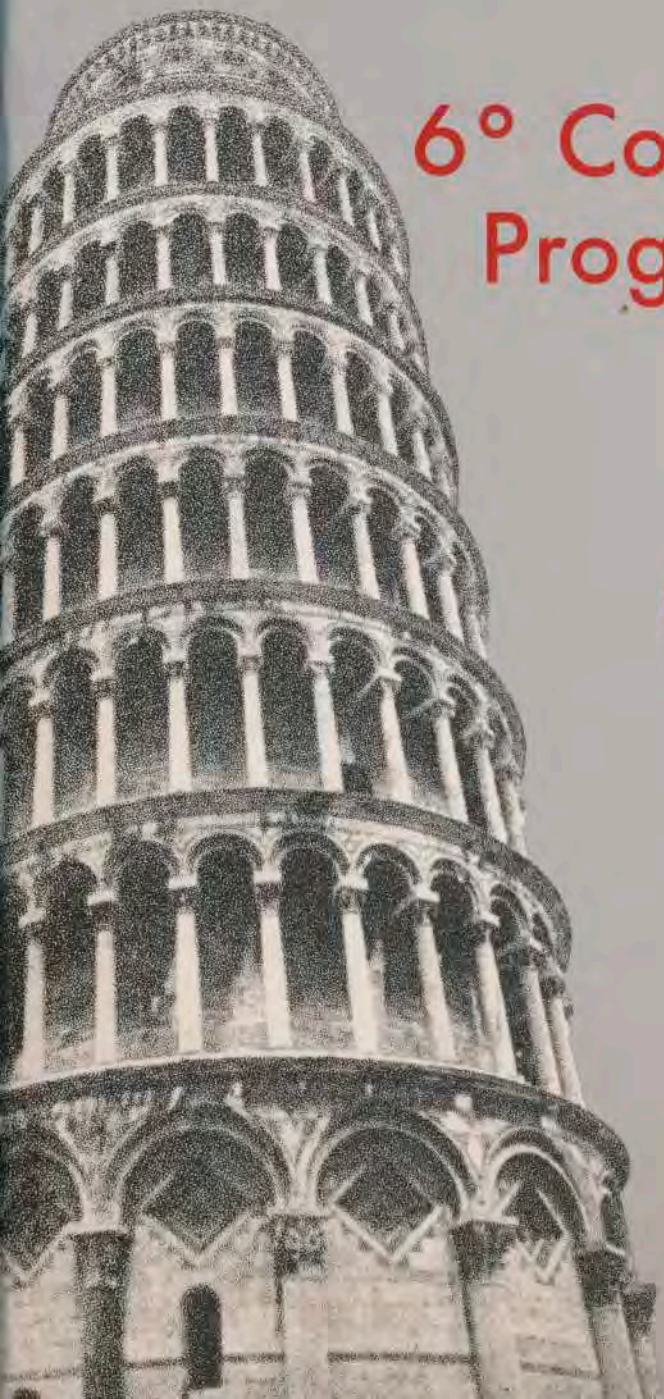


GULP '91

Atti del

6° Convegno sulla Programmazione Logica

Pisa, 12/14 giugno 1991

Centro Studi della
Cassa di Risparmio di Pisa
Piazza S. Paolo a Ripa d'Arno

:- gulp!

Gruppo Ricercatori e Utenti
di Logic Programming
affiliato alla Association
for Logic Programming

Gruppo Ricercatori ed Utenti
di Logic Programming

GULP '91

Atti del Sesto Convegno sulla Programmazione Logica

a cura di

Patrizia Asirelli

Pisa, 12-14 Giugno 1991
Centro Studi della Cassa di Risparmio di Pisa
P.za San Paolo a Ripa d'Arno

Coordinatrice:
Patrizia Asirelli, IEI – CNR – Pisa

Comitato di Programma:
Roberto Barbuti, Università di Pisa
Annalisa Bossi, Università di Padova
Luigia Carlucci Aiello, Università di Roma
Philippe Codognet, INRIA – Francia
Giovanni Criscuolo, Università di Napoli
Manuel Hermenegildo, Univ. Politecnica Madrid – Spagna
Paola Inverardi, IEI – CNR – Pisa
Gaetano Lanzarone, Università di Milano
Alberto Martelli, Università di Torino
Maurizio Martelli, Università di Genova
Paola Mello, Università di Bologna
Corrado Moiso, CSELT – Torino
Eugenio Omodeo, Università di Udine
Maria Vittoria Oneto, ELSAG – Genova
Mario Rodriguez–Artalejo, Univ. Complutense Madrid – Spagna
Domenico Saccà, Università della Calabria
Giuseppe Sardu, S & M – Pisa

Comitato Organizzatore:
Stefania Biagioni, IEI – CNR – Pisa
Antonio Brogi, Università di Pisa
Amedeo Cappelli, ILC – CNR – Pisa
Moreno Falaschi, Università di Pisa
Fosca Giannotti, CNUCE – CNR – Pisa

Organizzato da:
Istituto di Elaborazione della Informazione C.N.R. di Pisa

Con il contributo di:
AIS-Artificial Intelligence Software S.p.A.
Amministrazione Provinciale di Pisa
Azienda Promozione Turistica di Pisa
Banca Popolare di Novara
Cassa di Risparmio di Pisa
Centro Apple Education - Pisa
CNUCE - C.N.R. (Pisa)
Comune di Pisa
Consiglio Nazionale delle Ricerche
Consorzio Pisa Ricerche
Data Management S.p.A.
Delphi S.p.A.
Dipartimento di Informatica, Università di Pisa
Istituto di Linguistica Computazionale - C.N.R. (Pisa)
List S.p.A.

Premessa

Il Gruppo ricercatori e Utenti di Logic Programming (GULP) è affiliato alla Associazione internazionale di Logic Programming (ALP). Scopo del Gruppo è la divulgazione della Programmazione Logica e la creazione di un ambiente fertile per lo scambio di informazioni ed esperienze tra ricercatori ed utilizzatori che operano sia in ambito pubblico che privato. A tal fine il GULP promuove diverse attività, quali la pubblicazione di un bollettino, l'organizzazione di workshop, scuole avanzate e di un Convegno annuale.

Il Convegno annuale si articola, normalmente, in tre giornate e si propone principalmente di:

1) servire da punto d'incontro di tutti coloro che lavorano in questo ambito o ad esso sono interessati;
2) delineare lo stato dell'arte della ricerca nel settore, con relazioni invitate tenute da ricercatori di chiara fama;
3) avvicinare studenti e ricercatori alle principali tematiche della Programmazione Logica attraverso brevi corsi introduttivi.

I precedenti Convegni si sono svolti a Genova, Torino, Roma, Bologna e Padova, ed hanno registrato l'attiva e sempre crescente partecipazione di studiosi, sia del mondo accademico che di quello industriale.

Il programma scientifico di questa edizione include, per la prima volta, la presentazione di lavori di colleghi francesi e spagnoli che, numerosi, hanno aderito alla apertura di partecipazione ai Paesi vicini, offerta dal GULP. Questo aspetto della Sesta edizione della Conferenza annuale e l'ottimo livello delle relazioni presentate è indubbia conferma del successo e dell'approvazione, da parte della comunità scientifica del settore, delle attività che il GULP promuove.

A nome del GULP ringrazio Patrizia Asirelli e gli altri colleghi pisani per l'ottima organizzazione del Convegno.

Giorgio Levi
Presidente del GULP

Foreword

The "Gruppo ricercatori e Utenti di Logic Programming" (GULP) (which stands for Logic Programming Researcher and User Group) is an affiliate of the Association for Logic Programming (ALP). The goals of the group are to make Logic Programming more popular and to create opportunities for the exchange of experience and information between researchers and users working both in public and private organizations. For this purpose GULP promotes many different activities such as the publication of a bulletin, and the organization of workshops, advanced schools and its annual Conference.

This Conference is generally of three days and its main aims are:

1) to serve as an occasion for those working or interested in this area to meet and exchange experiences;

2) to illustrate the curent state of the art in the area through invited talks given by well-known researchers;

3) to enable students and researchers from other areas of Computer Science to learn more about Logic Programming by means of introductory tutorials.

The previous annual Conferences were held in Genoa, Turin, Rome, Bologna and Padua, and were attended each year by a growing number of participants.

The scientific programme of this Conference includes, for the first time, papers from French and Spanish colleagues. In fact, there has been a very positive reaction to the invitation to participate, made by GULP to all colleagues from France and Spain. This aspect of the Sixth Conference, considered together with the good technical quality of the papers accepted for presentation, is a further confirmation of the success and appreciation of the activities promoted by GULP.

On behalf of GULP, I should like to thank Patrizia Asirelli and all the other colleagues from Pisa for the excellent organization of the Conference.

Giorgio Levi
President of GULP

Prefazione

Questo volume raccoglie i lavori presentati al Sesto Convegno sulla Programmazione Logica: GULP '91. Il Convegno che, per la prima volta, vede la partecipazione di colleghi francesi e spagnoli è una occasione d'incontro e discussione fra studenti e ricercatori che svolgono la loro attività all'interno di strutture sia pubbliche che private quali Università, CNR e Industria. Il Convegno prevede inoltre la presentazione di prodotti e prototipi industriali.

Il convegno si tiene a Pisa dal 12 al 14 Giugno 1991 e prevede quattro relazioni invitate e la presentazione di 35 lavori, nonchè la presentazione di 4 posters.
I lavori ed i posters si collocano nelle seguenti aree:
Applicazioni e Relazioni con altre aree dell'Informatica; Estensioni ed Integrazioni; Interpretazione Astratta; Modularizzazione; Parallelismo; Programmazione con vincoli; Teoria e Fondamenti; Trasformazione.

Desidero ringraziare il GULP, ed in particolare il suo Presidente Giorgio Levi, per la fiducia dimostratami nell'affidarmi il coordinamento di questa edizione del Convegno annuale.
Vorrei ringraziare Giorgio Levi anche per i consigli elargiti durante tutta l'organizzazione e con lui un particolare ringraziamento va ad Amedeo Cappelli per l'instancabile aiuto e incoraggiamento.

Al buon esito del Convegno hanno contribuito in molti, tra i quali in particolare ringrazio:

- gli autori dei lavori sottoposti;
- i membri del Comitato di Programma;
- gli esperti che hanno aiutato i membri del Comitato di Programma nel revisionare i lavori:
M. Alpuente, G. Attardi, M. Bellia, P. Bonatti, P. Bosco, A. Cappelli, M. Cialdea, N. Cocco, L. Colussi, L. Console, P. Coscia, S. Dulli, M. Falaschi, G. Filè, M. G. Gatti, R. Giacobazzi, L. Giordano, E. Lamma, G. Levi, L. Lucchesi, P. Mancarella, D. Nardi, D. Pedreschi, A. Pettorossi, M. Proietti, F. Rossi, M. L. Sapino, P. Terenziani, D. Theseider Duprè, F. Turini;
- i membri del Comitato Organizzatore: Stefania Biagioni, Antonio Brogi, Amedeo Cappelli, Moreno Falaschi, Fosca Giannotti ed inoltre Anna Maria Giacobazzi per la preziosa collaborazione;
- l 'Istituto di Elaborazione della Informazione, per il supporto finanziario e la disponibilità delle sue strutture, ed in particolare il Direttore, Franco Denoth, per la collaborazione dimostrata;
- Annalisa Bossi e Paolo Mancarella per i preziosi consigli sull' organizzazione;
-gli enti che hanno contribuito alla organizzazione del GULP '91:
AIS-Artificial Intelligence Software S.p.A.; Amministrazione Provinciale di Pisa; Azienda Promozione Turistica di Pisa; Banca Popolare di Novara; Cassa di Risparmio di Pisa; Centro Apple Education - Pisa; CNUCE - C.N.R. Pisa; Comune di Pisa; Consiglio Nazionale delle Ricerche; Consorzio Pisa Ricerche; Data Management S.p.A.; Delphi S.p.A.; Dipartimento di Informatica - Università di Pisa; Istituto di Linguistica Computazionale - C.N.R. Pisa; List S.p.A.

Ringrazio inoltre i relatori invitati, gli autori che hanno presentato i propri lavori e tutti i partecipanti.

Patrizia Asirelli

Preface

This book contains the papers presented at the Sixth Conference on Logic Programming: GULP '91. The annual Conference is an occasion for Italian students and researchers to meet and discuss, in particular this year, for the first time, we have the pleasure to welcome our colleagues from France and Spain.The Conference also provides an occasion for industries to demonstrate their products or prototypes.

The Conference is held in Pisa, 12-14 June 1991. We have been able to schedule four invited talks, 35 papers, and a four posters.

The presentations fall into the following areas:
Abstract Interpretation; Applications and Relations with other areas of Computer Science; Constraint Logic Programming; Extensions and Integrations; Modularization; Parallelism; Theory and Foundations; Transformation.

I wish to thank GULP and, in particular, its President, Giorgio Levi, for giving me the opportunity to coordinate the organization of GULP '91.
I also wish to thank Giorgio Levi for all his valuable advice and Amedeo Cappelli for his cheerful and ever ready assistance.

Many people have given their contribution to help the conference achieve its goals and in particular I should like to thank:

- all the authors of the submitted papers;
- all members of the Program Committee;
- all scientists who, in addition to the Program Committee members, have acted as referee:
M. Alpuente, G. Attardi, M. Bellia, P. Bonatti, P. Bosco, A. Cappelli, M. Cialdea, N. Cocco, L. Colussi, L. Console, P. Coscia, S. Dulli, M. Falaschi, G. Filè, M. G. Gatti, R. Giacobazzi, L. Giordano, E. Lamma, G. Levi, L. Lucchesi, P. Mancarella, D. Nardi, D. Pedreschi, A. Pettorossi, M. Proietti, F. Rossi, M. L. Sapino, P. Terenziani, D. Theseider Duprè, F. Turini;
-all members of the Organizing Committee: Stefania Biagioni, Antonio Brogi, Amedeo Cappelli, Moreno Falaschi, Fosca Giannotti and furthermore Anna Maria Giacobazzi for their valuable collaboration;
- the "Istituto di Elaborazione della Informazione" and the Director, Franco Denoth, for financial support and for allowing me to use all the facilities of I.E.I., and C.N.R.
- Annalisa Bossi and Paolo Mancarella for their important advices about the organization;
- all enterprises that have supported the organization of GULP '91:
AIS-Artificial Intelligence Software S.p.A.; Amministrazione Provinciale di Pisa; Azienda Promozione Turistica di Pisa; Banca Popolare di Novara; Cassa di Risparmio di Pisa; Centro Apple Education - Pisa; CNUCE - C.N.R. Pisa; Comune di Pisa; Consiglio Nazionale delle Ricerche; Consorzio Pisa Ricerche; Data Management S.p.A.; Delphi S.p.A.; Dipartimento di Informatica - Università di Pisa; Istituto di Linguistica Computazionale - C.N.R. Pisa; List S.p.A.

Last but not least, I should like to express my gratitute to all the invited speakers, all the authors and all the participants.

Patrizia Asirelli

INDICE

Premessa (Premise) III - V

Prefazione (Preface) VII - IX

ABDUZIONE ***(Abduction)***

Abduzione e Negazione Intensionale in Programmazione Logica 3
P. Mancarella, F. Toni, F. Turini

The Completion Approach to Abduction and its Application to Classification in Taxonomic Hierarchies. 15
L. Console, D. Theseider Dupré

APPLICAZIONI ***(Applications)***

Uso di Conoscenza lessicale e grammaticale per la risoluzione dell'anafora in italiano 31
D. Bianchi, P. Frugoni, R. Delmonte

Configurazione di L.A.N.: Uno Strumento Intelligente di Supporto alle Decisioni 45
F. Medelin, F. Rota, L. Tanca, A. Zarattini

Time Granularity in Logical Specifications 63
E. Corsetti, A. Montanari, A. Morzenti, E. Ratto

EPSILON Adaptation on the EDS Machine: Some Implementation Issues 79
N. Leone, L. Palopoli, P. Restuccia, M.Romeo

FONDAMENTI ***(Foundations)***

Una Semantica Composizionale per Programmi Logici Aperti 95
A. Bossi, M. Menegus

Fair Computation Rules for Prolog Interpreters 111
S. Janot, J.P. Delahaye

On the Signature of Logic Programs 125
M. Gabbrielli, D. Turi

Notes on Legality Concepts for Logic Programs using Three-valued Logic 133
G. Nota, S. Orefice, G. Pacini, F. Ruggiero, G. Tortora

Equations Sets as a Tool for the Declarative Semantics of Perpetual Processes 149
A. Sperduti, X. J. Chen

Une Semantique Denotationelle Simple pour Prolog avec Predicats d'Entrees-Sorties 161
M. Billaud

Unification: Σ-terms or Substitutions? 177
M. Bellia, M.E. Occhiuto

INTEGRAZIONE CON ALTRI PARADIGMI (*Integration with other Paradigms*)

El Indeterminismo en un Lenguaje Logico-Aplicativo 193
A. Sarmiento

A Prolog Object-Oriented System: an Exercise in Contextual Logic Programming 209
C. Ruggieri, M. Bugliesi

A Concurrent Object-oriented Logic Language 227
E. Pimentel, J.M. Troya

{log}: A Language for Programming in Logic with Finite Sets 241
A. Dovier, E.G. Omodeo, E. Pontelli, G.Rossi

METAPROGRAMMAZIONE (*Metaprogramming*)

On the Optimization of Metalogic Programs 259
S. Costantini, G.A. Lanzarone, A.Laudani, A. Zanzi

Meta-Programmazione Logica in 'Log 275
I. Cervesato, G. Rossi

NEGAZIONE (*Negation*)

SLDNI-resolution: a non-floundering interpreter for general programs 293
A. Di Pierro, M. Martelli, C. Palamidessi

Consistency of Clark's: Completion and Existence of Stable Models 309
F. Fages

A Well-founded Semantics for Ordered Logic Programming 321
N. Leone, G. Rossi

PARALLELISMO E CONCORRENZA (*Parallelism and Concurrency*)

A Parallel Logic Programming Language for Distributed Architectures 333
A. Brogi, A. Ciampolini, E. Lamma, P. Mello

Constraints for Synchronizing Coarse-grained Sequential Logic Processes 349
A. Brogi, M. Gabbrielli, G. Levi

PROGRAMMAZIONE CON VINCOLI (*Constraint Logic Programming*)

VEL: un linguaggio logico a vincoli per l'integrazione di conoscenza simbolica e numerica 365
C. Moiso e M. Porta

Some Results on the Partial Evaluation of Constraint Logic Programs 377
C. Bendrame, C. Blanco, M. Marré, D. Yankelevich

Correctness and Completeness of Logic Programs under the CLP Scheme 391
B. Bertolino, P. A. Bonatti, D. Montesi, S.Pelagatti

Using Boolean constraints in Prolog III 407
J-M Boi, A.Rauzy

The Unfolding Semantics for Constraint Logic Programs 425
R. Giacobazzi

PROVE DI PROPRIETÀ (*Proving Properties*)

The abstract domain EXP for groundness, sharing, freeness and compoundness analysis 439
Á. Cortesi, G. Filé

Proving Universal Termination of a Logic Program 455
A. Bossi, N. Cocco, M. Fabris

Comprobacion de la integridad en bases de datos deductivas: una aproximacion basada en metaprogramacion 471
M. Celma, J.C. Casamayor, L. Mota, M.A. Pastor, F. Marquès

TRASFORMAZIONE DI PROGRAMMI
(*Program Transformation*)

Towards a Translation Algorithm from Prolog to the Andorra Kernel Language 489
F. Bueno, M. Hermenegildo

Modelling Control 505
R. Barbuti, T. Castagnetti, R. Giacobazzi, G. Levi

Andorra Prolog in FCP(:) 521
M. Falaschi , Y. Kesten, E. Shapiro

POSTERS

Logic Modelling of VLSI Circuits: the ITALTEL Experience 541
M. Bombana, P. Cavalloro, U. Pelizzari, G. Zaza

Development of Real-time Diagnostic System for Data Acquisition 543
R. Campanini, I. D'Antone, G. Di Caro, G. Giusti

Un Generatore di codice sorgente per Applicazioni Transazionali su Basi di Dati Realizzato in Prolog 545
S. Dulli, R. Sprugnoli, L. Veronese

A Friendly Interactive Tutorial of the Warren Abstract Machine 547
J. Garcia-Martin, J.J. Moreno-Navarro

ABDUZIONE

(Abduction)

Abduzione e Negazione Intensionale in Programmazione Logica

P. Mancarella, F. Toni e F. Turini

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
e-mail: paolo,turini@dipisa.di.unipi.it

Sommario

In questo lavoro vengono definiti frameworks abduttivi in programmazione logica, rispetto ad una sottoclasse dei programmi logici generali. In questi schemi coesistono due forme di negazione: da un lato una versione costruttiva della negazione come fallimento per relazioni non abducibili, e dall'altro un trattamento aperto della negazione di predicati abducibili. Vengono definiti dapprima una semantica model-theoretic e quindi un meccanismo di risoluzione corretto e completo. Infine viene presentato un esempio di utilizzo di tali frameworks nell'ambito della diagnosi di guasti.

1 Introduzione

Solo recentemente l'abduzione è stata riconosciuta come una regola di inferenza estremamente potente e in grado di catturare diversi aspetti di ragionamento non monotono in Intelligenza Artificiale [CP86,Poo88a,EK89,Lev89,KM90]. Nella sua forma più semplice l'abduzione consiste nell'inferire α come spiegazione plausibile dell'osservazione β in presenza dell'informazione $\alpha \supset \beta$ [CMcD85,Lev89].
L'uso di questa tecnica, che permette di inferire le premesse di una regola a partire dalla conoscenza delle sue conseguenze, è particolarmente appropriata per applicazioni di diagnostica, soprattutto rispetto a basi di conoscenza che evolvono nel tempo e/o sono strutturate mediante relazioni di tipo gerarchico. In questi ambiti è naturale rappresentare la conoscenza in moduli separati che esprimono informazioni parzialmente specificate sul dominio, ma tali da completarsi nel complesso. L'incompletezza della conoscenza relativa ad un singolo modulo può essere espressa dalla presenza di relazioni abducibili prive di qualsiasi definizione. In tali contesti non è sempre ragionevole effettuare inferenze basate su una assunzione del mondo chiuso [Rei78], inappropriata per il trattamento dell'incompletezza locale ad un modulo. Parte della conoscenza incompleta può invece essere modellata attribuendo ad alcune relazioni la natura di abducibili, ovvero ipotizzabili. Secondo tali presupposti, il processo di inferenza è autorizzato ad ipotizzare istanze di relazioni abducibili, ove la loro presenza risultasse indispensabile per la dimostrazione di una data osservazione (goal).

Vari autori hanno recentemente studiato il problema dell'integrazione di un meccanismo di ragionamento abduttivo in un sistema deduttivo, ed in particolare in sistemi basati sul paradigma di programmazione logica [Poo88a,CDT89b,FG85,EK89,KM90]. In questo lavoro ci proponiamo di dare un ulteriore contributo in questa direzione, presentando un framework abduttivo, basato sulla programmazione logica, in cui l'incompletezza della conoscenza modellata da un programma logico è rappresentata dalla presenza di predicati abducibili, le cui istanze possono essere ipotizzate durante la risoluzione di un goal. Prendendo spunto da analoghi frameworks abduttivi presentati ad esempio in [Poo88a,EK89,KM90], faremo l'ipotesi che i predicati definiti come abducibili non compaiano nelle teste delle clausole del programma, senza peraltro perdere in generalità, come mostrato ad esempio in [KM89].
I programmi logici che tratteremo sono (una sottoclasse dei) programmi logici generali, in cui il simbolo di negazione può comparire nei corpi delle clausole. Come vedremo più formalmente in seguito, la diversa natura dei predicati (abducibili e non) presenti nei programmi logici da noi trattati, richiede un trattamento differenziato della negazione, problema che, per quanto risulta agli autori, non è stato affrontato in maniera approfondita in letteratura.
Infatti, se da un lato la negazione di un letterale il cui simbolo di predicato è definito come abducibile è essa stessa abducibile, ciò non vale per la negazione di letterali non abducibili. Per questi ultimi, si assume che la conoscenza espressa dal programma sia completa e ciò, come usuale in programmazione logica, corrisponde al trattamento della negazione secondo la regola di negazione come fallimento finito [Cla78].
Dunque, nei frameworks abduttivi da noi trattati coesistono sia una forma di assunzione del mondo chiuso (quella corrispondente alla negazione come fallimento finito per i predicati non abducibili), sia l'assunzione complementare per i predicati abducibili (Open World Assumption, si veda ad esempio [Llo87]).
Utilizzeremo inoltre un approccio *costruttivo* alla negazione per i predicati standard, anziché l'approccio classico basato sulla risoluzione SLDNF [Llo87]. Ciò consente di superare alcuni dei limiti tipici della risoluzione SLDNF, quali il *floundering* ed il trattamento asimmetrico delle interrogazioni positive e negative.
La forma di negazione costruttiva da noi utilizzata è essenzialmente basata sulla cosiddetta *Negazione Intensionale* descritta in [Man88,BMPT90]. Essa prevede la rappresentazione *esplicita* della conoscenza negativa *implicita* nel completamento di un programma logico. Intuitivamente, a partire da un programma logico la negazione intensionale genera, per ogni simbolo di predicato p che occorre in esso, una definizione per un nuovo simbolo di predicato $\tilde{p}$ che rappresenta il complemento di p nel completamento del programma. In tal modo, un programma P viene trasformato in una coppia $\langle P^+, P^- \rangle$, dove P^+ è ottenuto da P rimpiazzando ogni letterale negativo $not\, p(t)$ con un il letterale positivo $\tilde{p}(t)$, e P^- contiene le definizioni calcolate per i nuovi predicati $\tilde{p}$.
Per mancanza di spazio, siamo costretti ad omettere in questo lavoro la definizione formale della negazione intensionale, rimandando il lettore a [Man88,BMPT90]. Faremo tuttavia due ipotesi restrittive sulla classe di programmi logici da noi trattati, che da un lato garantiscono l'applicabilità della tecnica di Negazione Intensionale, e dall'altro consentono di utilizzare meccanismi di risoluzione standard, la cui estensione al trattamento di predicati abducibili risulta poi molto naturale. La prima ipotesi riguarda la *linearità sinistra* dei

programmi [1]: essa garantisce la rappresentabilità finita della negazione di un termine, e quindi il conseguimento di un numero finito di clausole per i nuovi predicati $\tilde{p}$ generati dalla trasformazione (per maggiori dettagli si vedano [Man88,BMPT90]). La seconda ipotesi riguarda invece l'assenza di variabili locali nelle clausole[2], che assicura il fatto che il programma logico $\langle P^+, P^- \rangle$, ottenuto come negazione intensionale di P, è un programma definito, sul quale è possibile operare mediante risoluzione SLD. La classe di programmi generali che soddisfano tali restrizioni gode tra l'altro di interessanti proprietà algebriche, anche rispetto ad altri operatori, che ne fanno una sorta di *algebra* di programmi [MPRT90].
Nella sezione 2 presentiamo più formalmente i frameworks abduttivi da noi trattati e ne definiamo la semantica, ottenuta adattando quella presentata in [KM90] per frameworks analoghi in cui però la negazione è essa stessa trattata abduttivamente seguendo l'approccio di [EK89]. Nella sezione 3 definiamo un meccanismo di risoluzione per i nostri frameworks abduttivi, ottenuto estendendo la risoluzione SLD al trattamento dei predicati abducibili, e ne discutiamo le proprietà formali rispetto alla semantica data. Infine, nella sezione 4 viene presentata una esemplificazione dell'uso del meccanismo descritto nell'ambito dei sistemi diagnostici.

2 I Frameworks Abduttivi e la loro Semantica

Come accennato nell'introduzione, un *framework abduttivo* è costituito da un programma logico P lineare sinistro e senza variabili locali. Inoltre, alcuni dei simboli di predicato che occorrono in P sono definiti come *abducibili* e si suppone che per essi il programma non contenga definizioni[3]. Dunque, un programma P è visto come una base di conoscenza incompleta e l'incompletezza è proprio rappresentata dai predicati abducibili.
Intuitivamente, in presenza di una osservazione (goal) G l'idea è di cercare in P una dimostrazione (spiegazione) per G che può richiedere l'assunzione di alcune ipotesi, scelte opportunamente tra le istanze di predicati abducibili. È spesso necessario, oltre che utile, far sì che gli insiemi di ipotesi generati per la spiegazione abduttiva di una osservazione siano coerenti con una collezione di *vincoli* associati al framework abduttivo (si veda ad esempio [Poo88a,EK89,KM90]). Tali vincoli possono esprimere, ad esempio, il fatto che due diverse ipotesi non possono coesistere nella spiegazione di un generico goal, come nel caso dei frameworks da noi trattati. Tuttavia, l'approccio semantico che presentiamo in questa sezione, ispirato a quello descritto in [KM90], non fa alcuna ipotesi sulla natura di tali vincoli.
Si è già detto nella sezione introduttiva che la presenza della negazione nei corpi delle clausole richiede un trattamento accorto. Da un lato si vuole gestire in modo *costruttivo* la negazione di predicati standard, utilizzando la tecnica di negazione intensionale, la quale però deve essere leggermente modificata per garantire un trattamento "aperto" della negazione dei predicati abducibili. Infatti la costruzione testuale della componente negativa di un programma, indistintamente rispetto a tutti i simboli di predicato, causerebbe la presenza di definizioni per tutte le versioni negative dei predicati abducibili, forzandone il *completamento*. Illustriamo quanto appena accennato mediante un semplice esempio, che

[1] un programma logico è *lineare sinistro* se le teste delle clausole contengono esclusivamente termini *non ristretti*, ovvero senza occorrenze multiple di variabili.
[2] una variabile è detta *locale* se occorre nel corpo ma non nella testa di una clausola
[3] in altre parole, un predicato che compare in una testa di clausola in P non può appartenere a tale insieme.

fornisce anche un'idea del comportamento della negazione intensionale.

Esempio
Consideriamo il seguente programma generale P:
$p(x) :- q(x), s(x)$
$q(x) :- not\ r(x)$
La negazione intensionale di P trasforma i letterali negativi in letterali positivi con nuovi simboli di predicato, la cui definizione viene poi generata mediante una opportuna trasformazione applicata alle regole di P (si veda [BMPT90,Man88]). Il risultato, come già detto, è una coppia di programmi $\langle P^+, P^- \rangle$ che nel nostro esempio sono, rispettivamente:

$p(x) :- q(x), s(x)$ e $\tilde{p}(x) :- \tilde{q}(x)$
$q(x) :- \tilde{r}(x)$ $\tilde{p}(x) :- \tilde{s}(x)$
$\tilde{q}(x) :- r(x)$
$\tilde{s}(x) :-$
$\tilde{r}(x) :-$

Come si vede, la componente negativa contiene due asserzioni per i predicati $\tilde{s}$ e $\tilde{r}$, che intuitivamente corrispondono alla presenza, nel completamento di P, degli assiomi $\forall x.\ s(x) \iff false$ e $\forall x.\ r(x) \iff false$.
Assumendo ora che s sia un predicato abducibile, e dunque lo sia anche $\tilde{s}$, dobbiamo far sì che la componente negativa del programma generato per negazione intensionale sia della forma:
$\tilde{p}(x) :- \tilde{q}(x)$
$\tilde{p}(x) :- \tilde{s}(x)$
$\tilde{q}(x) :- r(x)$
$\tilde{r}(x) :-$
La definizione per $\tilde{s}$ deve cioè essere omessa.

Definizione 2.1 *(Negazione Intensionale Abduttiva)*
Sia P un programma logico, $\langle P^+, P^- \rangle$ il risultato della negazione intensionale applicata a P e A un insieme di simboli di predicato che non contengono definizioni in P. Allora, la *negazione intensionale abduttiva* di P, che denotiamo con P_A, è il programma definito costituito dall'unione delle clausole in $\langle P^+, P^- \rangle$, dalle quali sono state rimosse tutte le asserzioni del tipo $\tilde{p}(x) :-$, con $p \in A$.

Per semplicità manterremo la notazione $\langle P^+, P^- \rangle$ per indicare le componenti "positiva" e "negativa" di P_A.
Possiamo dunque definire un framework abduttivo come segue:

Definizione 2.2 *(Framework Abduttivo)*
Dato un programma logico P, un insieme di simboli di predicato A (abducibili) che non contengono definizioni in P e un insieme di formule del prim'ordine I (vincoli di integrità), il *framework abduttivo* corrispondente è la tripla $\langle P_A, Abd, I \rangle$ dove P_A è la negazione intensionale abduttiva di P rispetto ad A e $Abd = \{p(t), \tilde{p}(t) \mid p \in A, t$ termine ground$\}$.

La caratterizzazione semantica di un framework abduttivo $\langle P_A, Abd, I \rangle$ è basata sull'idea di individuare un insieme di possibili modelli plausibili rispetto all'insieme dei vincoli in I. A questo proposito, facciamo riferimento all'idea presentata in [KM90] in cui si definisce dapprima una classe di pre–modelli associati ad un framework abduttivo, e successivamente

un modello come un pre–modello in cui sono vere le formule corrispondenti ai vincoli di integrità. Come in [KM90] un pre–modello non è altro che l'insieme delle conseguenze logiche atomiche deducibili a partire da un insieme di abducibili prefissato $\Delta \subseteq Abd$.

Definizione 2.3 *(pre–Modello Abduttivo)*
Siano $\langle P_A, Abd, I \rangle$ un framework abduttivo e Δ un sottoinsieme di Abd. Allora il modello minimo di Herbrand di $P_A \cup \Delta$, che denotiamo con $M(\Delta)$, è un pre–Modello Abduttivo (pre–MA) per $\langle P_A, Abd, I \rangle$.

Ricordiamo che, date le restrizioni sintattiche sui programmi logici da noi trattati, P_A è sempre un programma definito e dunque la precedente definizione è ben fondata. Vediamo un esempio.

Esempio
Consideriamo lo schema abduttivo in cui P^+, P^- e Abd sono i seguenti:

P^+: $p :- ab_1, q$ P^-: $\tilde{p} :- \widetilde{ab_1}$
$q :- ab_2$ $\tilde{p} :- \tilde{q}$
$q :- \widetilde{ab_1}$ $\tilde{q} :- \widetilde{ab_2}, ab_1$

$Abd = \{ ab_1, ab_2, \widetilde{ab_1}, \widetilde{ab_2} \}$.
I pre–modelli abduttivi corrispondenti a $\Delta_1 = \{ab_1, ab_2\}$ e $\Delta_2 = \{ab_1, \widetilde{ab_1}, ab_2\}$ sono rispettivamente $M(\Delta_1) = \{ab_1, ab_2, p, q\}$ e $M(\Delta_2) = \{ab_1, \widetilde{ab_1}, ab_2, p, q, \tilde{p}\}$. Si noti che per ogni possibile configurazione per $\Delta \subseteq Abd$, esiste il corrispondente pre-MA. Osserviamo invece che il modello minimo di P_A visto come normale programma logico è l'insieme vuoto.

Definizione 2.4 *(Modello Abduttivo)*
Siano $\langle P_A, Abd, I \rangle$ un framework abduttivo e Δ un sottoinsieme di Abd. Allora un pre–MA $M(\Delta)$ è un Modello Abduttivo (MA) per $\langle P_A, Abd, I \rangle$ se
$M(\Delta) \models I$.

Da questa definizione appare chiaro come il ruolo dei vincoli di integrità sia quello di rendere inaccettabili come modelli quei pre–modelli che non rendono veri i vincoli stessi. Intuitivamente, un modello abduttivo $M(\Delta)$ riflette lo "stato delle cose" associato ad un generico insieme di ipotesi prefissato, rappresentato dall'insieme Δ.

Esempio (segue)
Proseguendo nell'esempio precedente, sia I costituito dal seguente vincolo:
$not(p \wedge \widetilde{ab_1})$
Dei due pre–modelli $M(\Delta_1)$ e $M(\Delta_2)$ visti in precedenza, solo il primo è un MA.

Dunque la semantica di un framework abduttivo è data dall'insieme di tutti i modelli abduttivi ad esso associati. In quest'ottica il concetto di spiegabilità di un'osservazione coincide con quello di verità in tali modelli, come formalizzato nella seguente:

Definizione 2.5 *(Spiegazione Abduttiva)*
Dato un framework abduttivo $\langle P_A, Abd, I \rangle$ ed un goal G, un insieme Δ di abducibili è una *spiegazione abduttiva* (SA) per G se M(Δ) è un MA per $\langle P_A, Abd, I \rangle$ tale che $M(\Delta) \models G$.

È chiaro che eventuali letterali negativi della forma *not* $p(t)$ presenti in G vanno considerati come occorrenze dei corrispondenti "intensionali" $\tilde{p}(t)$, e dunque G è un goal definito.
La precedente definizione esprime il fatto che un generico meccanismo abduttivo deve consentire la dimostrabilità di un fatto specificato come goal, ovvero la spiegabilità di un'osservazione data, in virtù delle ipotesi assunte e compatibilmente con i vincoli presenti. La semantica dei modelli abduttivi può essere adattata ai particolari frameworks analizzati in questo lavoro, che trattano programmi definiti derivati dalla trasformazione intensionale abduttiva di programmi generali. Il problema è quello di definire opportunamente i vincoli di integrità in modo da garantire la consistenza delle spiegazioni abduttive lecite. A questo scopo, bisogna fa sì che i modelli abduttivi, e dunque le corrispondenti spiegazioni, non contengano coppie complementari di abducibili, ovvero del tipo $p(t), \tilde{p}(t)$. Ciò può essere garantito imponendo che i vincoli di integrità associati ad un framework abduttivo contengano formule del tipo

$$not(p(t) \wedge \tilde{p}(t))$$

per ogni predicato abducibile p e termine ground t.
Esempio (segue)

Riprendendo l'esempio precedente, I deve contenere i seguenti vincoli:

$$not(ab_1 \wedge \widetilde{ab_1})$$
$$not(ab_2 \wedge \widetilde{ab_2}).$$

Nota: nel seguito tratteremo frameworks abduttivi $\langle P_A, Abd, I\rangle$ in cui i vincoli di integrità I sono tutti e soli del tipo appena descritto, per cui ometteremo di menzionare esplicitamente I, utilizzando la notazione $\langle P_A, Abd\rangle$.

Le spiegazioni abduttive in frameworks di questo tipo verranno riferite come ***Spiegazioni Abduttive Intensionali***, formalizzate dalla seguente:

Definizione 2.6 *(Spiegazione Abduttiva Intensionale)*
Dato uno schema abduttivo $\langle P_A, Abd\rangle$ e un goal G, una spiegazione abduttiva $\Delta \subseteq Abd$ è detta ***spiegazione abduttiva intensionale*** (SAI) per G in $\langle P_A, Abd\rangle$.

Dalla definizione di spiegazione abduttiva data in precedenza segue che Δ è una SAI per G in $\langle P_A, Abd\rangle$ se

- $M(\Delta)$=$M_{P_A \cup \Delta}$,
- $M(\Delta) \models not\ (p(t) \wedge \tilde{p}(t))$, $\forall p \in$ A, $\forall t$ termine ground,
- $M(\Delta) \models G$.

Esempio (segue)
Consideriamo lo schema abduttivo dell'esempio precedente. L'insieme di abducibili $\Delta_1 = \{ab_1, ab_2\}$ è una SAI per il goal $:-p$. Si noti che invece $\Delta_3 = \{ab_1\}$ non è una SAI per G, benché $M(\Delta_3) = \{ab_1\}$ sia un modello abduttivo per $\langle P_A, Abd\rangle$.

3 Un meccanismo di risoluzione abduttiva

In questa sezione definiamo un meccanismo di risoluzione abduttiva, ispirato a quello presentato in [EK89], e ne studiamo le proprietà formali rispetto alla semantica delle

spiegazioni abduttive intensionali. Il meccanismo descritto è una naturale estensione della risoluzione SLD [Llo87], che, in corrispondenza di un goal G, consente di ipotizzare i letterali abducibili contenuti in una sua SAI. Per semplicità ci limitiamo al trattamento di ipotesi abduttive *ground*, ovvero senza occorrenze di variabili, analogamente a quanto effettuato in [EK89]. A questo scopo, assumiamo che la regola di calcolo sottostante sia tale da selezionare un letterale abducibili L solo se ground.

Definizione 3.1 *(Derivazione abduttiva)*
Sia $\langle P_A, Abd\rangle$ un framework abduttivo, G un goal e R una regola di calcolo. Una derivazione abduttiva per G in $\langle P_A, Abd\rangle$ è una sequenza di coppie

$$(G_0, \Delta_0), \ldots, (G_i, \Delta_i), \ldots, (G_n, \Delta_n)$$

con $G_0 = G$ e $\Delta_i \subseteq Abd$ per $i = 0, \ldots, n$, tali che $\forall i = 0, \ldots, n-1$, se G_i è $:-(\alpha(t), Rest)$ [4], (G_{i+1}, Δ_{i+1}) è ottenuto secondo uno dei casi seguenti:

abd1) α non abducibile,
- G_{i+1} è ottenuto da G_i mediante un passo di risoluzione SLD rispetto al programma P_A
- $\Delta_{i+1} = \Delta_i$;

abd2) $\alpha(t) \in Abd$, $\alpha(t) \in \Delta_i$,
- $G_{i+1} = :-Rest$
- $\Delta_{i+1} = \Delta_i$;

abd3) $\alpha(t) \in Abd$, $\alpha(t) \notin \Delta_i$, $\tilde{\alpha}(t) \notin \Delta_i$,
- $G_{i+1} = :-Rest$
- $\Delta_{i+1} = \Delta_i \cup \{\alpha(t)\}$.

Nel seguito indichiamo con $(G_0, \Delta_0) \stackrel{*}{\rightarrow}_{abd} (G, \Delta)$ (risp. $(G_0, \Delta_0) \stackrel{n}{\rightarrow}_{abd} (G_n, \Delta_n)$) una derivazione abduttiva di lunghezza imprecisata ma finita (risp. di lunghezza n).

Definizione 3.2 *(Refutazione abduttiva)*
Una refutazione abduttiva di un goal G rispetto ad uno schema abduttivo $\langle P_A, Abd\rangle$, è una derivazione abduttiva di lunghezza finita n con $(G_0, \Delta_0) = (G, \emptyset)$ e $(G_n, \Delta_n) = (\Box, \Delta)$.
Δ è detto ***insieme di abducibili calcolato*** dalla refutazione.

La definizione di risoluzione abduttiva, e conseguentemente quella di refutazione, può essere facilmente estesa al caso in cui l'insieme iniziale di ipotesi è un generico insieme $\Delta_0 \subseteq Abd$ non necessariamente vuoto.
Nel seguito indicheremo con $\langle P_A, Abd\rangle$, $\Delta_0 \stackrel{\Delta}{\vdash}_{abd} G$ l'esistenza di una refutazione abduttiva di G che calcola Δ a partire da Δ_0, cioè tale che $(G, \Delta_0) \stackrel{*}{\rightarrow}_{abd} (\Box, \Delta)$.

Il meccanismo di risoluzione descritto gode delle proprietà di correttezza e completezza rispetto alla semantica presentata nella sezione 2. La loro dimostrazione si basa sulle

[4] Con α indichiamo un generico simbolo di predicato del tipo p o $\tilde{p}$ e $\alpha(t)$ è l'atomo selezionato da R, e con $\tilde{\alpha}$ rispettivamente $\tilde{p}$ e p.

corrispondenti proprietà della risoluzione SLD rispetto alla semantica dichiarativa standard dei programmi logici definiti [Llo87]. Infatti, è possibile ricondurre una refutazione abduttiva ad una refutazione SLD rispetto ad un'opportuna estensione del programma P_A, come stabilito dal seguente:

Lemma 3.1
$\langle P_A, Abd\rangle, \Delta_0 \stackrel{\Delta}{\vdash}_{abd} G \Longrightarrow \quad P_A \cup \Delta \vdash_{SLD} G.$

Il teorema di correttezza può essere enunciato nel modo seguente [5].

Teorema 3.1 (Correttezza)
$\langle P_A, Abd\rangle, \Delta_0 \stackrel{\Delta}{\vdash}_{abd} G \Longrightarrow \Delta$ è una SAI per G in $\langle P_A, Abd\rangle$.

La dimostrazione di tale proprietà è relativa alla deducibilità del goal considerato dal programma esteso con l'insieme di ipotesi calcolato, e alla consistenza di quest'ultimo. Il primo aspetto è garantito dal lemma 3.1, mentre la consistenza dell'insieme di ipotesi generato è assicurata dall'assenza di coppie complementari, cioè del tipo $[p(t), \tilde{p}(t)]$, con $p(t) \in Abd$, che deriva direttamente dalla definizione di risoluzione abduttiva (caso abd3 della definizione 3.1).
Analizziamo adesso la proprietà di completezza del meccanismo descritto, relativa alla calcolabilità effettiva di una qualsiasi spiegazione abduttiva intensionale nel senso della definizione 2.6. Prima di enunciare il teorema di completezza, esemplifichiamo alcune situazioni problematiche.

Esempio
Consideriamo uno schema abduttivo che contiene le seguenti definizioni per il predicato p e per il suo complementare $\tilde{p}$:
$p(x) : -ab(x)$
$\tilde{p}(x) : -\widetilde{ab}(x)$
dove ab è un predicato abducibile.
Supponendo che l'universo di Herbrand contenga, tra l'altro, le costanti a, b e c, è chiaro che $\Delta = \{\ ab(a),\ \widetilde{ab}(b),\ ab(c)\ \}$ è una SAI per il goal $: -p(a)$, ma la refutazione abduttiva di quest'ultimo calcola Δ'=$\{ab(a)\}$, che è un sottinsieme proprio di Δ. La presenza in Δ di ipotesi superflue ai fini della dimostrazione abduttiva del goal fornito comporta dunque la restrizione della proprietà di completezza ad un sottinsieme effettivo dell'insieme assegnato. Proseguendo nell'esempio, se $\Delta = \{ab(a)\}$ e G è il goal non ground $: -p(x)$, si verifica che $P_A \cup \Delta \vdash_{SLD} G$ con sostituzione di risposta $\theta = \{x/a\}$, ma G non è abduttivamente dimostrabile, dal momento che il meccanismo di risoluzione abduttiva non è in grado di istanziare opportunamente l'atomo abducibile $ab(x)$ e dunque di calcolare Δ come insieme di ipotesi.

Osserviamo comunque che $\langle P_A, Abd\rangle, \Delta_0 \stackrel{\Delta'}{\vdash}_{abd} G\vartheta$, con $G\vartheta$=$: -p(a)$ e Δ'=$ab(a)$.

La restrizione effettuata sulla regola di calcolo che consente di abdurre solo ipotesi ground determina un fenomeno analogo a quello del *floundering* nella risoluzione SLDNF [Llo87]. Tuttavia, poiché i programmi trattati non contengono variabili locali, possiamo evitare tale fenomeno assumendo che il goal di partenza sia ground. Otteniamo dunque il seguente risultato di *completezza ground.*

[5] per mancanza di spazio omettiamo le dimostrazioni dei teoremi che seguono, rimandando il lettore a [Ton90].

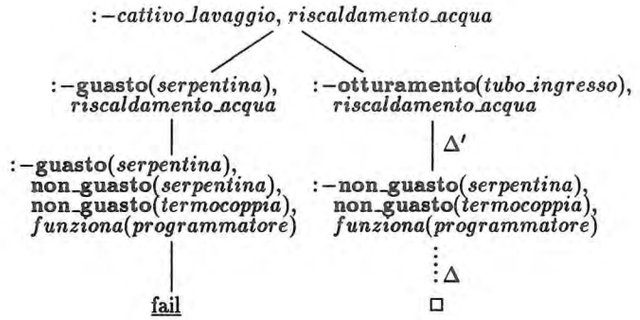
Teorema 3.2 (Completezza ground)
Dato $\Delta \subseteq Abd$ consistente, e G goal ground tale che $M_{P_A \cup \Delta} \models$ G, vale che:
$\langle P_A, Abd\rangle, \Delta_0 \stackrel{\Delta'}{\vdash}_{abd} G$, con Δ' finito e $\Delta' \subseteq \Delta$ finito.

4 Un esempio

Il meccanismo risolutivo descritto può essere usato come base per applicazioni nell'ambito dell'IA che richiedano un approccio abduttivo, compatibilmente con una assiomatizzazione dei domini inquadrabile nello schema delineato. Descriviamo un esempio nell'ambito della diagnostica, legato ad una caratterizzazione abduttiva del problema a partire da una rappresentazione logica della conoscenza [Poo88b].
L'esempio descritto riguarda l'individuazione di guasti per una lavatrice. La conoscenza del sistema è espressa mediante un programma logico, indicato con T, secondo un modello *causale* [CDT89a] in cui il dominio è formalizzato tramite implicazioni logiche che esprimono il passaggio causa-effetto.
Il programma T descrive il processo mentale del tecnico dell'apparecchio considerato, che è in grado di risalire alle cause del malfunzionamento sulla base delle segnalazioni dell'utente derivate da un'analisi della realtà contingente.
Diamo qui di seguito parte della descrizione di T, evidenziando in neretto i predicati abducibili. Si noti quanto l'uso esplicito della negazione renda di immediata interpretazione il significato del programma.

T : *cattivo_lavaggio* : $-$**guasto**(*serpentina*)
cattivo_lavaggio : $-$**otturamento**(*tubo_ingresso*)
riscaldamento_acqua : $-$
not **guasto**(*serpentina*),
not **guasto**(*termocoppia*),
funziona(*programmatore*)
generico_guasto : $-$*not* *funziona*(*tasto_accensione*)
generico_guasto : $-$*guasto_interno*
guasto_interno : $-$*not* *funziona*(*programmatore*)
guasto_interno : $-$*not* *funziona*(*elettrovalvola*)
guasto_interno : $-$*not* *funziona*(*motore*)
funziona(*programmatore*) : $-$
not *acqua_sui_contatti*(*programmatore*),
not **guasto**(*isolante*),
not **guasto**(*connettori*)
acqua_sui_contatti(x) : $-$**guasto**(*tubi_collegamento*)
⋮

L'elaborazione di una diagnosi per un malfunzionamento consiste, da un punto di vista computazionale, nella risoluzione del goal corrispondente all'insieme di osservazioni, rispetto alla teoria che rappresenta la conoscenza del tecnico, ottenuta dall'applicazione della trasformazione di negazione intensionale abduttiva al programma T. Di quest'ultima omettiamo per semplicità la descrizione. Si noti comunque che nel seguito la negazione intensionale di un predicato p sarà indicata con *non_p*.



Δ' = {**otturamento**(*tubo_ingresso*)}
Δ = {**otturamento**(*tubo_ingresso*),
non_guasto(*isolante*), **non_guasto**(*serpentina*),
non_guasto(*connettori*), **non_guasto**(*termocoppia*),
non_guasto(*tubi_collegamento*)}

Figura 1: risoluzione di G

Supponiamo adesso che le informazioni sintomatiche a disposizione del tecnico riguardino il cattivo lavaggio effettuato dall'elettrodomestico, registrato in concomitanza con il fenomeno di corretto riscaldamento dell'acqua. L'intervento del tecnico consiste allora nel risolvere il goal G

$:-$*cattivo_lavaggio*, *riscaldamento_acqua*

Parte dell'albero di risoluzione corrispondente è mostrato in figura 1. Si noti che il ramo di fallimento è dovuto alla presenza in esso della coppia complementare **non_guasto**(*serpentina*) e **guasto**(*serpentina*).
L'insieme di ipotesi Δ calcolato contiene la causa di malfunzionamento individuata, **otturamento**(*tubo_ingresso*), affiancata da abducibili "negativi" che segnalano esplicitamente l'assenza di guasti per altri elementi del sistema. Come segnalato in [CDT89a] questo è vantaggioso rispetto ad approcci in cui l'informazione negativa non è disponibile in modo esplicito, ma deve essere ottenuta interpretando come false le ipotesi non occorrenti in una spiegazione calcolata. In alcune situazioni diagnostiche è infatti necessario distinguere tra abducibili realmente ridondanti ed ipotesi scartate sulla base di osservazioni effettive, come in ambito medico, in cui si deve tener conto della presenza di controindicazioni nella scelta di una terapia appropriata.
Il meccanismo di risoluzione definito risulta dunque uno strumento flessibile, adattabile a più ambiti dell'IA con peculiarità diverse.

Riferimenti

[BMPT90] Barbuti, R., Mancarella, P., Pedreschi, D., Turini, F., *A Transformational Approach to Negation in Logic Programs.* J. Log. Progr.2 (1990) 201–228.

[CMcD85] Charniak, E., McDermott, D., *Introduction to Artificial Intelligence.* Addison-Wesley (1985)

[Cla78] Clark,K.L., *Negation as Failure.* Logic and Data Bases(Gallaire,H. & Minker,J. eds.) (1978) 293–322.

[CDT89a] Console, L., Duprè, D., Torasso, P. *Abductive Reasoning through Direct Deduction from Completed Domain Models.* Methodologies for Intelligent Systems 4 (Z. Ras ed.) North-Holland (1989) 175.

[CDT89b] Console, L., Duprè, D., Torasso, P. *A Theory for Diagnosis for Incomplete Causal Models.* Proc. 11th IJCAI-89 (1989) 1311.

[CP86] Cox, P. T., Pietrzykowskii, T., *Causes for Events: their Computation and Applications.* Proc. CADE-86 (1986) 608–621.

[EK89] Eshghi, K., Kowalski, R.A., *Abduction compared whit Negation by Failure.* Proc 6th Int. Conf. on Log. Progr. (1989) 234.

[FG85] Finger, J.J., Genesereth, M.R., *RESIDUE: A Deductive Approach to Design Synthesis.* Report no. CS-85-1035, Stanford University. (1985)

[KM89] Kakas, A. C., Mancarella, P., *Anomalous Models and Abduction.* Proc. 2nd Int. Symp. on Artif. Int., Monterrey, Mexico (1989).

[KM90] Kakas, A. C., Mancarella, P., *Generalized Stable Models: a Semantics for Abduction.* Proc, of ECAI 90 (1990) 385–391.

[Lev89] Levesque, H.J., *A knowledge-level account of abduction.* Proc. 11th IJCAI (1989) 1061.

[Llo87] Lloyd, J. W., *Foundations of Logic Programming.* seconda edizione, Springer Verlag (1987).

[Man88] Mancarella, P., *La Negazione Intensionale di Programmi Logici.* Tesi di Dottorato di Ricerca, Università di Pisa (1988).

[MPRT90] Mancarella, P., Pedreschi, D., Rondinelli, M., Tagliatti, M., *Algebraic Properties of a Class of Logic Programs.* Proc. NACLP 90, Mit Press (1990).

[Poo88a] Poole, D., *A logical Framework for Default Reasoning.* Artif. Intell. 36 (1988).

[Poo88b] Poole, D., *Representing Knowledge for Logic-based Diagnosis.* Proc. of the Int. Conf. on Fifth Generation Computer System (1988) 1282–1290.

[Rei78] Reiter, R., *On closed world data bases.* Logic and Data Bases(Gallaire,H. & Minker,J. eds.) (1978) 55–76.

[Ton90] Toni, F. *Abduzione e Negazione Intensionale in Programmazione Logica.* Tesi di laurea, Dipartimento di Informatica, Università di Pisa (1990).

THE COMPLETION APPROACH TO ABDUCTION AND ITS APPLICATION TO CLASSIFICATION IN TAXONOMIC HIERARCHIES

Luca CONSOLE*, Daniele THESEIDER DUPRE'**

* Dipartimento di Matematica e Informatica
Universita` di Udine
Via Zanon 6
33100 Udine
Telephone: (+39) 432 295716
Telefax: (+39) 432 507882
E-mail: lconsole@di.unito.it

** Dipartimento di Informatica
Universita` di Torino
Corso, Svizzera 185
10149 Torino
Telephone: (+39) 11 7712002
Telefax: (+39) 11 751603
E-mail: dtd@di.unito.it

ABSTRACT

Abduction is a form of non-monotonic reasoning that can be seen as an extension of the computational paradigm of logic programming; therefore its formalization is particularly interesting in the area of the relations between logic programming and non-monotonic reasoning. In this paper we present a deductive framework for abduction which is related to logic programming since it is based on predicate completion. The framework is then applied to the solution of classification problems in taxonomic hierarchies with exceptions.

1. INTRODUCTION

There is a growing interest in the relation between non-monotonic reasoning and logic programming [13], mainly originated from the fact that the negation as failure rule used in logic programming is a nonmonotonic rule of inference (and, as a consequence, various semantics for it have been based or can be expressed in terms of various non-monotonic formalisms [16]).

In particular, we are interested in studying the integration of *abduction* within a logic programming framework. Abduction is the form of reasoning that, given a domain theory T and a formula Q, consists in finding a set of assumptions that, in conjunction with the domain theory, entail Q, and therefore can be seen as an *explanation* of Q given T.

Abduction seems to be a basic reasoning component in many intelligent problem solving activities, such as diagnosis [7, 12] and planning [8]. Moreover, abduction seems to naturally extend the computational paradigm of logic programming, allowing for the generation of conditional answers to queries, i.e. a set of assumptions can be returned as an answer to a query [9, 17].

In this paper we present a deductive approach to abduction: in fact, we show that abduction can be reduced to deduction using predicate completion. Since predicate completion

The research described in this paper has been partially supported by grants from MPI 40% (Automated Reasoning Techniques for Intelligent Systems) and CNR.

provides a semantics for a class of logic programs which is not exhaustive but appears to be well suited for abductive problem solving, this is another interesting intersection between abduction and logic programming. The approach presented in this paper extends the basic one introduced in [4, 5] allowing for domain theories expressed as logic programs with negation and taking into account several types of knowledge that can be used to refine the answers.

We demonstrate, throughout several examples, the usefulness of this approach in solving classification problems in taxonomic hierarchies with exceptions, that is knowledge bases which represent hierarchies of classes and subclasses of entities and in which the properties of a class are inherited by the subclasses but exceptions are possible. Reasoning in such hierarchies is a relevant problem in knowledge representation. However, most of the work has been concentrated on the problem of determining, given a class C (or an individual a) and a property p, whether C (or a) has the property p. The problem is not purely deductive due to the presence of exceptions and therefore various non-monotonic reasoning approaches have been applied [10, 18].

We are interested in the opposite problem: given an individual having some properties, find those classes having (at least) these properties, i.e. those classes the individual can be assumed to belong to. This problem, which is clearly abductive in nature, has not received much attention so far (an example of this application of abductive reasoning is given in [15]) and allows us to show various features of our framework.

The problem of classification in taxonomic hierarchies without exceptions, on the other hand, has received a lot of attention in the area of hybrid knowledge representation languages [14]. In such a case, given the description of the properties of a class C, the classification task is to determine the position of C in the taxonomy, i.e. the most general classes which C subsumes and the most specific classes which subsume C. Such a task is deductive in nature since subsumption can be seen as logical implication (C_1 subsumes C_2 if and only if $C_2(x) \rightarrow C_1(x)$). This points out the fundamental difference with respect to our notion of "classification".

The paper is organized as follows: in section 2 we review the basic deductive framework for abduction we have proposed, observing that it naturally accommodates for logic programs with negation to be used as domain theories; in section 3 we extend the framework to take into account several knowledge sources that are useful to refine the answer in many domains, and, in particular, to solve classification problems in taxonomies. The required extensions are presented informally throughout several examples (section 3.1) and then formalized (section 3.2). In section 4 we sketch the relationship between our framework and other approaches to abduction (a more complete discussion is reported in [5, 6]).

2. THE BASIC FRAMEWORK

An abduction problem P is characterized by:

a domain (explanatory) theory T: a set of general clauses

$$L_1 \wedge ... \wedge L_n \rightarrow A$$

where L_i are literals and A is an atom, and the body is intended to "explain" the head (for example, the clause may correspond to a cause-effect relationship). Since it is not reasonable to explain something in terms of itself, we assume that the domain theory is hierarchical [3][1]. The set of predicates that do not appear in the head of any clause is the set of abducible predicates, i.e. those that can be accepted in explanations.

a formula Ψ to be explained: a conjunction of literals.

In [4, 5] we have proposed an "object-level" account of abduction and discussed its correspondence with more traditional "meta-level" definitions of abduction. In the object-level approach, the abductive conclusions are derived deductively at the same level of the domain theory. The idea of such an approach is the following. If the domain theory T contains the following clauses having "q" as their head:

$$p_1 \rightarrow q$$
$$...$$
$$p_n \rightarrow q$$

then, in order to explain "q", at least one of the "p_i" has to be assumed. This form of reasoning can be reduced to deduction provided that one takes the predicate completion [3] of "q", i.e. adds the formula

$$q \rightarrow p_1 \vee ... \vee p_n$$

One can then reason in the theory resulting from the union of the observations and the completion T_C of all the non-abducible predicates in the domain theory. In particular, since the completed domain theory is a set of equivalences, and the original domain theory is hierarchical, one can rewrite the observations using these equivalences until one obtains a formula containing only abducible predicates (and the equality predicate, in the first-order case), called the **explanation formula**[2]. We then have that the explanation formula F is such that

$$T_C \models \Psi \longleftrightarrow F$$

where Ψ is the conjunction of positive and negative literals describing the observations, and, in particular, F follows deductively from $T_C \cup \Psi$.

Abduction has thus been characterized in terms of deduction at the same level of the theory T; this is the reason for denoting our approach as an object-level one, as opposed to the following meta-level definition (which is similar to those proposed in the literature):

A set E of abducible atoms is a (meta-level) explanation for Ψ **iff** $T \cup E \models \Psi$

The following correspondence result is proved in [5]:

E is a meta-level explanation **iff** an interpretation satisfying exactly those abducible atoms occurring in E is a model of the explanation formula.

This result shows that each model of the explanation formula F can be considered as an explanation for the abduction problem. If one wants to give a syntactic characterization of the notion of object-level explanation, one can turn the explanation formula into disjunctive normal form and consider the different disjuncts as explanations.

The object-level approach naturally accommodates for clauses with negation (interpreted as failure to prove) since it is based on predicate completion. Moreover, it has the following interesting properties:

---

[1] The discussion in the following can be generalized to acyclic programs (theories) and bounded queries (formulae to be explained) in the sense of [1].

[2] The same kind of evaluation is suggested in [1] for bounded queries and acyclic programs. In that case there are no abductive assumptions and the goal is to obtain a formula containing only the equality predicate.

- it clarifies the relationship between abduction and deduction (i.e. shows that abduction can be regarded as deduction in a completed theory);
- it produces a formula that provides an abstract, model-theoretic characterization of all the explanations.

3. KNOWLEDGE FOR MORE CLEVER EXPLANATIONS

In many application domains, knowledge is not only of the form "A explains B". In particular, abductive hypotheses are not independent; we are therefore interested in extending the above approach by considering several forms of knowledge about abducible predicates. In particular, in addition to an explanatory theory T in the sense of section 2, we will consider the following pieces of knowledge about abducible predicates of T:

- a set A of taxonomic relations of the form:

$$B_1 \rightarrow B_2$$

that expresses a class-subclass relation, or, in general, that B_2 is an abstraction of B_1 (A can be naturally assumed to be hierarchical since it is intended to represent a taxonomy);

- a set I of denials

$$\neg (B_1 \wedge ... \wedge B_n)$$

that can be used, in particular (with n=2), to express mutual exclusion relationships;

- a set D of ground atoms that can be useful in particular cases where some instance of abducible predicates is already known (i.e. D represents "factual knowledge" about the abducibles).

Such forms of knowledge are useful not only in the formalization of taxonomic reasoning, as will be illustrated below, but also in other domains. For example, in diagnostic reasoning, taxonomic relations can be used to represent the relation between general and specific faults (e.g., *battery_faulty* $\rightarrow$ *electrical_system_faulty*), and it can also be necessary to express that two faults are mutually exclusive (e.g., $\neg$(*rich_mixture* $\wedge$ *lean_mixture*), since they are two opposite anomalies of the mixture produced by the carburettor).

The use of the various forms of knowledge will be the following. In the generation of explanations, one should trace back from the observations to their explanations on the basic explanatory theory, but not on the taxonomic relationships. In other words, the process must halt as soon as an abducible atom is reached: an abducible atom α should not be explained in terms of another abducible atom β unless β has to be assumed for other reasons. This last case shows that taxonomic knowledge can be used to simplify the explanations. In general, taxonomic knowledge, constraints and factual knowledge can be used to simplify the explanation formula, pruning or simplifying some of the explanations, given what is already known about abducibles. In such a way a more clever answer can be provided.

Abduction can be thus characterized at the object level as a two-step process:

i) Determine the explanation formula F for the abduction problem P using the observations Ψ and the predicate completion T_C as discussed in section 2.

ii) Transform F into a formula F^* using knowledge about abducibles contained in A, I and D. In particular, the goal of the transformation is to obtain a formula F^* which is equivalent, given A $\cup$ I $\cup$ D, to F. In this way, the result will be a logical consequence of $T_C \cup$ A $\cup$ I $\cup$ D $\cup \Psi$, which is, in this case, the suitable theory from which abductive

conclusions can be derived deductively (at the object level). Moreover, the result will be equivalent to Ψ (not only a consequence of Ψ), given $T_C \cup$ A $\cup$ I $\cup$ D, to avoid any loss of information about the problem at hand.

In the following we first illustrate, throughout several examples, the importance and use of knowledge about abducible predicates for classification in taxonomic hierarchies; then we provide formal definitions for the transformations of the explanation formula based on A, I and D.

3.1. EXAMPLES OF CLASSIFICATION IN TAXONOMIC HIERARCHIES

The knowledge representation framework described above will be used in this section to represent taxonomic hierarchies with explicit exceptions in the following way:

- A is obviously used to represent class-subclass relations:

$$C_1(x) \rightarrow C(x)$$

expresses the fact that C_1 is a subclass of C.

- T will be used to represent properties of classes; "default" properties will be represented using an abnormality predicate, as it is common in non-monotonic reasoning:

$$C(x) \wedge \neg ab(x) \rightarrow p(x)$$

expresses the fact that elements of the class C have the property p unless they are exceptions. A different abnormality predicate is needed for every pair (C, p). Exceptions will be explicitly given in T:

$$C_1(x) \rightarrow ab(x)$$
$$...$$
$$C_n(x) \rightarrow ab(x)$$

mean that each C_i (which is supposed to be a subclass of C - this should be represented in A) is an exception to the rule above, i.e. an exception in C with respect to the property p. Abductive reasoning will be performed assuming that all the exceptions have been explicitly given: if something does not have the property "p", the explanations that can be given are that it does not belong to C or it belongs to one of the C_i (see the examples below). An alternative is to treat abnormality predicates as abducible, and therefore put the exception clauses in A.

- I can be used to represent that two subclasses of the same class are mutually exclusive.
- D can be used in case it is already known that some object belongs to a given class (see examples 2 and 3 below).

Example 1. As a first example, let us consider the following theory about flying birds:

$$T_1 = \{bird(x) \wedge \neg ab(x) \rightarrow flies(x),$$
$$ostrich(x) \rightarrow ab(x),$$
$$bird(x) \rightarrow feathered(x),$$
$$canary(x) \rightarrow yellow(x)\}$$

$$A_1 = \{ostrich(x) \rightarrow bird(x),$$
$$canary(x) \rightarrow bird(x)\}$$

$$I_1 = \{\neg[ostrich(x) \wedge canary(x)]\}$$

The clauses in T_1 specify that normal birds fly, ostriches are abnormal birds (with respect to

the property of flying), birds have feathers and canaries are yellow. A_1 specifies the taxonomic relationships (i.e. that ostriches and canaries are birds). Finally, I_1 specifies that an individual cannot be both an ostrich and a canary.
Consider the observation

$$flies(Tweety)$$

Using the framework presented previous section 2, it can be rewritten as

$$bird(Tweety) \wedge \neg ostrich(Tweety)$$

which says that, up to the knowledge in T_1, the fact that Tweety flies can be explained only by assuming that Tweety is a bird, but not an ostrich (i.e. a bird which is not abnormal with respect to the property of flying). However, if the observation is

$$flies(Tweety) \wedge yellow(Tweety)$$

it is rewritten as

$$bird(Tweety) \wedge \neg ostrich(Tweety) \wedge canary(Tweety)$$

which can be simplified to

$$canary(Tweety)$$

given A_1 and I_1: the fact that Tweety is a bird is already implicit, given A_1, in its being a canary; similarly, the fact that Tweety is not an ostrich is already implicit, given I_1, in its being a canary; i.e. we have:

$$A_1 \models canary(Tweety) \rightarrow bird(Tweety)$$
$$I_1 \models canary(Tweety) \rightarrow \neg ostrich(Tweety)$$

and therefore the simplified formula is equivalent, given $A_1 \cup I_1$, to the previous one (we will not repeat this observation for the following examples).

On the other hand, consider the problem of explaining

$$feathered(Fred)$$

This is rewritten as

$$bird(Fred)$$

and there is no need to further trace back on the taxonomy, making unnecessary assumptions on what type of bird Fred is (this corresponds to looking for explanations called *least presumptive* in [15]).

Negative observations can be also dealt with. The observation

$$\neg flies(Sam)$$

is rewritten as

$$\neg bird(Sam) \vee ostrich(Sam)$$

and the observation

$$\neg flies(Sam) \wedge feathered(Sam)$$

is rewritten as

$$[\neg bird(Sam) \vee ostrich(Sam)] \wedge bird(Sam) \equiv$$
$$\equiv ostrich(Sam) \wedge bird(Sam)$$

which can be simplified to

$$ostrich(Sam)$$

given A_1.

Notice that in our framework we can also explain non-ground formulae (including negative non-ground literals). For example, given

$$\neg flies(x)$$

we get the (non-ground) explanation

$$\neg bird(x) \vee ostrich(x)$$

and similarly in other cases.

Example 2. In this example we consider a case where we have factual knowledge about an individual (in particular, we know that the individual John is an ostrich).

$$T_2 = \{bird(x) \wedge \neg ab(x) \rightarrow flies(x),$$
$$ostrich(x) \rightarrow ab(x),$$
$$ostrich(x) \rightarrow walks(x)\}$$

$$A_2 = \{ostrich(x) \rightarrow bird(x)\}$$

$$D_2 = \{ostrich(John)\}$$

The observation

$$flies(John)$$

is rewritten as

$$bird(John) \wedge \neg ostrich(John)$$

which can be simplified to

$$false$$

given D_2. This means that there is no explanation for $flies(John)$. This should be clear from the fact that the explanations are in correspondence with the models of the explanation formula.

The observation

$$walks(John)$$

is rewritten as $ostrich(John)$ which can be simplified to

$$true$$

given D_2; i.e. no assumption is necessary to explain the observation which already follows from $T_2 \cup D_2$. Less trivial cases in which factual knowledge is useful are discussed in the next example.

Example 3. Let us consider a theory in which we know about birds and bats; in particular, we know that both normal birds and normal bats fly:

$$T_3 = \{bird(x) \wedge \neg ab_bird(x) \rightarrow flies(x),$$
$$bird(x) \rightarrow walks(x),$$
$$ostrich(x) \rightarrow ab_bird(x),$$
$$ostrich(x) \rightarrow has_long_neck(x),$$
$$bat(x) \wedge \neg ab_bat(x) \rightarrow flies(x),$$
$$bat(x) \rightarrow walks(x)\}$$

$$A_3 = \{ostrich(x) \rightarrow bird(x)\}$$

$$I_3 = \{\neg[bat(x) \wedge bird(x)]\}$$

$$D_3 = \{ostrich(Bob), bird(Sam)\}$$

The observation

$$has_long_neck(John) \wedge flies(John)$$

is rewritten as

$$ostrich(John) \wedge \{[bird(John) \wedge \neg ostrich(John)] \vee bat(John)\} \equiv$$
$$\equiv ostrich(John) \wedge bat(John)$$

which can be further simplified to

$$false$$

(i.e. there is no explanation) given A_3 and I_3 (they entail $\neg[bat(John) \wedge ostrich(John)]$).

The observation

$$walks(Fred) \wedge has_long_neck(Fred)$$

is rewritten as

$$[bird(Fred) \wedge ostrich(Fred)] \vee [bat(Fred) \wedge ostrich(Fred)]$$

which can be simplified to

$$bird(Fred) \wedge ostrich(Fred)$$

since A_3 and I_3 entail $\neg[bat(Fred) \wedge ostrich(Fred)]$, and then further simplified to

$$ostrich(Fred)$$

since from A_3 we know $ostrich(Fred) \rightarrow bird(Fred)$.

The observation

$$walks(Bob)$$

is rewritten as

$$bird(Bob) \vee bat(Bob)$$

which can be transformed into

$$[bird(Bob) \wedge ostrich(Bob)] \vee [bat(Bob) \wedge ostrich(Bob)]$$

given D_3, then simplified to

$$ostrich(Bob)$$

given A_3 and I_3 (as above) and then to

$$true$$

given D_3.

Finally, let us consider the problem of explaining

$$flies(Sam)$$

This can be rewritten into

$$[bird(Sam) \wedge \neg ostrich(Sam)] \vee bat(Sam)$$

However, since from D_3 we know that Sam is a bird and then from I_3 we know that Sam is not a bat, we can thus give as an answer

$$\neg ostrich(Sam)$$

In other words, in order to explain the observation, since we already know that Sam is bird, it is sufficient to assume that it is not an ostrich.

Example 4. Let us consider now an example (adapted from [11]) in which there are exceptions to exceptions. In particular, we know that normal molluscs bear a shell, cephalopods (which are a subclass of molluscs) are an exception, but nautiluses (which are cephalopods) do bear a shell.

$$T_4 = \{mollusc(x) \wedge \neg ab(x) \rightarrow shell_bearer(x),$$
$$cephalopod(x) \rightarrow ab(x),$$
$$nautilus(x) \rightarrow shell_bearer(x)\}$$

$$A_4 = \{cephalopod(x) \rightarrow mollusc(x),$$
$$nautilus(x) \rightarrow cephalopod(x)\}$$

The observation

$$shell_bearer(Fred)$$

is rewritten as

$$[mollusc(Fred) \wedge \neg cephalopod(Fred)] \vee nautilus(Fred)$$

that is, Fred is either a nautilus or a mollusc other than cephalopod.

The observation

$$\neg shell_bearer(John)$$

is rewritten as

$$\neg [mollusc(John) \wedge \neg cephalopod(John)] \wedge \neg nautilus(John)$$

which can be simplified to

$$\neg mollusc(John) \vee [cephalopod(John) \wedge \neg nautilus(John)]$$

given A_4 (it entails $\neg mollusc(John) \rightarrow \neg nautilus(John)$).

Notice that this example would also work if there existed a further exception level such as a class of "naked-nautiluses", i.e. nautiluses with no shell. In that case the answers would change to

$$[mollusc(Fred) \wedge \neg cephalopod(Fred)] \vee [nautilus(Fred) \wedge \neg naked_nautilus(Fred)]$$

in the first case, and to

$$\neg mollusc(John) \vee [cephalopod(John) \wedge \neg nautilus(John)] \vee naked_nautilus(John)$$

in the second case.

3.2. FORMAL DEFINITION OF THE TRANSFORMATIONS

In the following we present the general form of the transformations used in the above examples. For the sake of simplicity, we shall limit ourselves to the case where F is ground (see the comments below).

1) Take $F \wedge [\bigwedge D_F]$, i.e. the conjunction of F and all the atoms in D_F, where D_F is the subset of D constituted by the atoms in which occur constants occurring in F (D_F contains the fact relevant to the problem under examination). Turn into disjunctive normal form.

2) Simplify the result using the constraints I (possibly in conjunction with the taxonomy A); in particular,

2.1) if $I \models \neg(\alpha_1 \wedge ... \wedge \alpha_n)$ and $A \models \beta_i \rightarrow \alpha_i$ (i=1,..,n), then eliminate disjuncts containing $\beta_1 \wedge ... \wedge \beta_n$, i.e. transform

(F'): $(\beta_1 \wedge ... \wedge \beta_n \wedge E_1') \vee E_2 \vee ... \vee E_n$

into the equivalent

(F"): $E_2 \vee ... \vee E_n$

Notice that F"$\models$F' is trivial, while the opposite requires A$\cup$I (i.e. F'$\cup$A$\cup$I $\models$ F"). For example, this simplification is used in example 3 in the case about Fred.

2.2) If $I \models \neg(\alpha_1 \wedge ... \wedge \alpha_n \wedge \gamma)$ and $A \models \beta_i \rightarrow \alpha_i$ and $A \models \delta \rightarrow \gamma$, then transform

(F'): $(\beta_1 \wedge ... \wedge \beta_n \wedge \neg\delta \wedge E_1') \vee E_2 \vee ... \vee E_n$

into the equivalent

(F"): $(\beta_1 \wedge ... \wedge \beta_n \wedge E_1') \vee E_2 \vee ... \vee E_n$

Notice that F'$\models$F" is trivial, while the opposite requires A $\cup$ I (i.e. F"$\cup$A$\cup$I $\models$ F'). For example, this simplification is used in example 1 in the generation of the explanation for *flies*(*Tweety*) $\wedge$ *yellow*(*Tweety*).

3) Simplify the explanation formula using the taxonomy A; in particular, if $A \models \alpha \rightarrow \beta$, then:

3.1) Transform

(F'): $(\alpha \wedge \beta \wedge E_1') \vee E_2 \vee ... \vee E_n$

into the equivalent

(F"): $(\alpha \wedge E_1') \vee E_2 \vee ... \vee E_n$

Notice that F'$\models$F" is trivial, while the opposite requires A (i.e. F"$\cup$A $\models$ F'). For example, this simplification is used in example 1 in the generation of the explanation for *flies*(*Tweety*) $\wedge$ *yellow*(*Tweety*).

3.2) Transform

(F'): $(\alpha \wedge E_1') \vee (\beta \wedge E_1') \vee E_2 \vee ... \vee E_n$

into the equivalent

(F"): $(\beta \wedge E_1') \vee E_2 \vee ... \vee E_n$

Notice that F"$\models$F' is trivial, while the opposite requires A (i.e. F'$\cup$A $\models$ F"). This is the case where both a general hypothesis β and a more specific hypothesis α explain the same observation (and there is no specific evidence for α)[3]: in such a case the most general hypothesis should be preferred.

3.3) Eliminate disjuncts ($\alpha \wedge \neg\beta \wedge$ E'), i.e. transform

[3] This happens when the model contains a redundant link between the specific hypothesis α and some of the observations. The same explanation (F") could be obtained without such a redundant link.

(F'): $(\alpha \wedge \neg\beta \wedge E_1') \vee E_2 \vee ... \vee E_n$

into the equivalent

(F"): $E_2 \vee ... \vee E_n$

Notice that F"$\models$F' is trivial, while the opposite requires A (i.e. F'$\cup$A $\models$ F"). This transformation corresponds to removing a disjunct which is inconsistent with the taxonomy.

4) Remove the atoms in D.

We have the following property, which extends the property that $T_C \models \Psi \longleftrightarrow$ F (where F is the explanation formula) to the case of abduction with knowledge about abducibles.

Property. Let F^* be the result of the process above for the abduction problem characterized by T, A, I, D, Ψ; we have:

$$T_C \cup A \cup I \cup D \models \Psi \longleftrightarrow F^*.$$

The proof is a consequence of the already mentioned property ($T_C \models \Psi \longleftrightarrow$ F) and of the equivalence of F and F^* given A $\cup$ I $\cup$ D (see the comments in the definitions of the transformations).

In the first order case more general transformations are needed to deal with non-ground formulae (this is a further complication, other than the fact that the equality theory associated with the completion has to be considered in order to simplify the explanation formula or formulae obtained at intermediate steps of rewriting).

Let us consider, for example, transformation 2.1 and suppose that $\neg(\alpha(x) \wedge \beta(x)) \in I$; in such a case if a disjunct contains $\alpha(t_1) \wedge \beta(t_2)$, then, in general, the condition $t_1 \neq t_2$ must be added to the disjunct. In the particular case where t_1 and t_2 are identical, the disjunct can be removed, since the inequality is unsatisfiable in the equality theory associated with the completion; in case t_1 and t_2 are not unifiable, the inequality can be left out, since it is valid in the equality theory [3].

As we noticed before, the only important criterion that must be taken into account is that the original explanation formula F and the transformed formula F^* must be equivalent given A $\cup$ D $\cup$ I (and the equality theory associated with the completion).

4. CORRESPONDENCE RESULT

The correspondence result proved in [5] and sketched in section 2 is limited to the case where the theory T is a set of propositional definite clauses. A similar correspondence result can be proved also in the more general case discussed in this paper. In order to prove such a result, we must extend the definition of meta-level explanation in the following way:

Given the abduction problem characterized by T, A, I, D, Ψ, a set E of abducible atoms is a **meta-level explanation** for Ψ if

1) $T \cup A \cup D \cup E \vdash_{NF} \Psi$,
where $\vdash_{NF}$ is the query evaluation procedure introduced in [3];

2) $E \cup I \cup A \cup D$ is consistent.

In order to define the correspondence, it is necessary to characterize how a meta-level explanation and a model can be seen as correspondent. In particular:

1) We are only interested in models that satisfy I; however, we do not want to make explicit, in the formula characterizing the explanations, all the constraints and facts, including those that are not relevant to solve the specific problem under examination (i.e. we do not want to take $F^* \wedge I$ instead of F^*). Therefore we shall consider models of F^* and I.

2) The same interpretation may correspond to different meta-level explanations which are equivalent given $A \cup D$. In particular, we shall consider a model M and a meta-level explanation E as correspondent in case for every ground abducible atom α:

$$M \models \alpha \text{ iff } A \cup D \cup E \models \alpha$$

Notice that this implies $M \models A \cup D$ and thus we will not require explicitly that $M \models A \cup D$.

The correspondence result is then the following (the proof is reported in [6]).

Property. Let us consider the abduction problem characterized by T, I, A, D, Ψ; let F^* be the result of the transformations above on the explanation formula. Let E be a set of abducible atoms of T and M an interpretation such that for every abducible atom α:

$$M \models \alpha \text{ iff } A \cup D \cup E \models \alpha$$

If Ψ does not flounder in $T \cup A \cup D \cup E$, then

E is a meta-level explanation **iff** ($M \models F^*$, $M \models I$).

Notice that the correspondence is restricted to the case where there is no floundering. A more general notion of meta-level explanation should make use of some form of constructive negation [2].

CONCLUSIONS

In this paper we have presented an approach that shows how abduction can be reduced to deduction using predicate completion. This is an interesting intersection with logic programming. Another nice characteristic of the approach is that it provides a very general form of answer that characterizes all the abductive explanations; negative information and variables are naturally dealt with, and, in particular, variables and negation can appear in the answer. The approach can be successfully applied to the solution of classification problems in taxonomic hierarchies with exceptions, taking properly into account various knowledge sources different from purely "explanatory" knowledge.

REFERENCES

[1] Apt, K.R. and Bezem, M., "Acyclic programs," pp. 617-633 in *Proc. 7th Int. Conf. on Logic Programming*, MIT Press, Jerusalem (1990).

[2] Chan, D., "Constructive Negation Based on the Completed Database," pp. 111-125 in *Proc. 5th Int. Conf. and Symposium on Logic Programming*, MIT Press, Seattle (1988).

[3] Clark, K., "Negation as failure," pp. 293-322 in *Logic and Data Bases*, ed. H. Gallaire, J. Minker,Plenum Press, New York (1978).

[4] Console, L., Theseider Dupre´, D., and Torasso, P., "Abductive Reasoning through Direct Deduction from Completed Domain Models," pp. 175-182 in *Methodologies for Intelligent Systems 4*, ed. Z. Ras,North Holland (1989).

[5] Console, L., Theseider Dupre´, D., and Torasso, P., "A Completion Semantics for Object-level Abduction," pp. 72-76 in *Proc. AAAI Symposium on Automated Abduction*, Stanford (1990).

[6] Console, L., Theseider Dupre´, D., and Torasso, P., "On the Relationship between Abduction and Deduction," Techn. Rept. Universita' di Torino (February 1991).

[7] Cox, P.T. and Pietrzykowski, T., "General Diagnosis by Abductive Inference," pp. 183-189 in *Proc. IEEE Symposium on Logic Programming*, San Francisco (1987).

[8] Eshghi, K., "Abductive Planning with Event Calculus," pp. 562-579 in *Proc. 5th Int. Conf. and Symposium on Logic Programming*, MIT Press, Seattle (1988).

[9] Eshghi, K. and Kowalski, R., "Abduction Compared with Negation as Failure," pp. 234-254 in *Proc. 6th Int. Conf. on Logic Programming*, MIT Press, Lisbon (1989).

[10] Etherington, D., "Formalizing Nonmonotonic Reasoning Systems," *Artificial Intelligence* **31** pp. 41-85 (1987).

[11] Fahlman, S., Touretzky, D., and van Roggen, W., "Cancellation in a Parallel Semantic Network," pp. 257-263 in *Proc. 7th IJCAI*, , Vancouver (1981).

[12] Goebel, R., Furukawa, K., and Poole, D., "Using Definite Clauses and Integrity Constraints as the basis for a Theory Formation Approach to Diagnostic Reasoning," pp. 211-222 in *Proc. 3rd Int. Conf. on Logic Programming*, Springer Verlag Lecture Notes in Computer Science 225 (1986).

[13] Marek, W., Nerode, A., Pedreschi, D., and Subrahmanian, V.S. (eds.), *Proc. of the Workshop on Logic Programming and Non-Monotonic Logic,* Austin (November 1990).

[14] Nebel, B., *Reasoning and Revision in a Hybrid Knowledge Representation System,* Springer Verlag, LNCS 422 (1990).

[15] Poole, D., "Explanation and Prediction: An Architecture for Default and Abductive Reasoning," *Computational Intelligence* **5**(1) pp. 97-110 (1989).

[16] Przymusinski, T., "Non-monotonic formalisms and logic programming," pp. 655-674 in *Proc. 6th Int. Conf. on Logic Programming*, MIT Press, Lisbon (1989).

[17] Sergot, M., "A Query-the-user facility for logic programming," pp. 27-41 in *Integrated Interactive Computer Systems*, ed. P. Degano, E. Sandewall,North Holland (1983).

[18] Touretzky, D., Horty, J., and Thomason, R., "A Clash of Intuitions: The Current State of Nonmonotonic Multiple Inheritance Systems," pp. 476-482 in *Proc 10th IJCAI*, Milano (1987).

APPLICAZIONI

(*Applications*)

Uso di conoscenza lessicale e grammaticale per la risoluzione dell'anafora in italiano.

Dario Bianchi Paolo Frugoni
Dipartimento di Fisica, Università di Parma
Viale delle Scienze 43100 Parma

Rodolfo Delmonte
Istituto di Linguistica e Didattica delle Lingue
Università di Venezia
Ca' Garzoni Moro S.Marco 3417- 30124 Venezia

March 17, 1991

Abstract

In questo lavoro vengono presentati due moduli per la risoluzione dell'anafora per la lingua italiana. L'analisi delle frasi realizzata mediante il formalismo delle LFG (Lexical Functional Grammar) si mostra particolarmente adatta a descrivere le regole che legano antecedente e pronominale. A livello di frase si introduce la nozione strutturale di f-comando oltre a regole grammaticali specifiche per le varie categorie pronominali. Per risolvere l'anafora nel discorso è stato sviluppato un modello a stati basato sulla nozione di centro del discorso. I moduli, così come il parser, sono stati realizzati in Prolog.

1 Introduzione

Con il termine *anafora* si definisce il mezzo per fare all'interno del discorso un riferimento abbreviato ad una o più entità, presupponendo che il ricevente del discorso sia in grado di disabbreviare tale riferimento. Il riferimento è detto *anaforico*: risolvere l'anafora significa pertanto trovare l'*antecedente* di un riferimento anaforico, l'entità cioè al quale esso riferisce.

Il problema dell'anafora è di grande rilevanza in quasi tutti i settori del NLP: natural language understanding (NLU), generazione di frasi in linguaggio naturale, traduzione automatica.

Per una trattazione più dettagliata del problema, si rimanda a [Hirst 81] ed a [Frugoni 90], dove, tra l'altro, è presente un'ampia rassegna dei principali approcci al problema dell'anafora, qui omessa per ragioni di spazio. In particolare emerge che, nonostante gli approcci possano essere di varia natura (grammaticali, semantici, euristici) nessun sistema può essere soddisfacente se si limita ad analizzare la singola frase. Inoltre la maggior parte dei lavori sono sviluppati solo a livello teorico da linguisti: al momento non si hanno notizie di altri sistemi operanti sulla lingua italiana.

Il risolutore delle anafore qui presentato si compone di due moduli: il primo cerca l'antecedente di un riferimento anaforico all'interno della frase. Dove questa ricerca non

abbia successo interviene il secondo modulo, il risolutore delle anafore nel discorso, che cerca un antecedente fuori della frase. Si estraggono i principali argomenti di un discorso mano a mano che esso si evolve, e parallelamente si riconduce la risoluzione degli eventuali riferimenti anaforici ad un *matching* su tali argomenti.

Il presente lavoro si colloca nell'ambito di un progetto tuttora in fase di sviluppo presso il Dipartimento di Fisica dell'Università di Parma in collaborazione con il Dipartimento di Linguistica dell'Università di Venezia riguardante un sistema per l'analisi di testi in lingua italiana.

I componenti del sistema sono realizzati in Prolog, sia per la facilità di realizzazione del parser che per mantenere uno stile dichiarativo nella stesura delle specifiche regole per la ricerca degli antecedenti.

2 Parser e rappresentazione delle frasi

Il sistema utilizza un parser che si basa sulle Definite Clause Grammar (DCG). La grammatica include un meccanismo di estraposizione sinistra [Pereira 83] per il trattamento di frasi relative, interrogative e a struttura topicalizzata.

Il riferimento linguistico utilizzato è quello delle grammatiche Lessico Funzionali (LFG) [Bresnan 80]. Vi è quindi un lessico che fornisce per nomi, verbi ed aggettivi la forma predicativa. Questa include per ogni argomento del predicato la categoria grammaticale, il ruolo tematico e la categoria semantica.

Sempre su base lessicale è possibile assegnare i soggetti di frasi infinitive (*Pro*) e di altri complementi predicativi.

Il parser fornisce in output strutture funzionali, dette f-strutture, che, in accordo a quelle presentate in [Bresnan 80], possono essere così definite: una f-struttura è un set ordinato di coppie attributo-valore. Un attributo è il nome di una funzione grammaticale o di una caratteristica (ad esempio soggetto, predicato, oggetto, numero, caso...). I valori possono essere:

1. simboli
2. f-strutture sussidiarie
3. set di simboli e f-strutture come nel caso di modificatori ed aggiunti.

In fig. 1 è illustrato un esempio di f-struttura per la frase:

La mamma regala un giocattolo al bambino.

Una f-struttura può essere applicata ad un attributo per ottenere un valore:

$$f_2\ num = sing$$

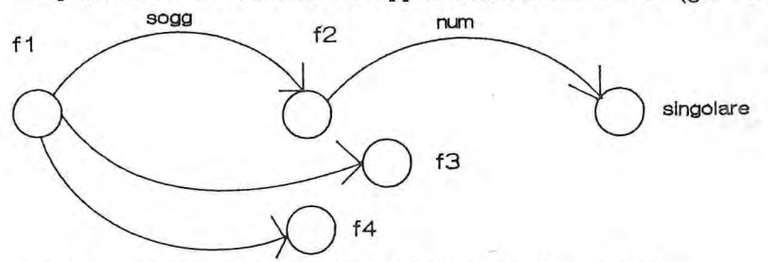
Applicando una f-struttura ad un attributo è possibile ottenere ancora una f-struttura che può essere applicata ad un secondo attributo (in modo simile alla composizione di funzioni).

$$f_1\ sogg/agente = f_2$$
$$(f_1\ sogg/agente)\ num = sing$$

$$\left[\begin{array}{lll}
pred & regalare & \\
modo & ind & \\
tempo & pres/semp & \\
pers & 3 & \\
num & sing & \\
cat & operazione & \\
sogg/agente & \left[\begin{array}{ll} cat & [umano] \\ pred & mamma \\ gen & fem \\ num & sing \\ det & det \end{array}\right]_{f_3} & \\
ogg/tema & \left[\begin{array}{ll} cat & [oggetto] \\ pred & giocattolo \\ gen & mas \\ num & sing \\ det & indef \end{array}\right]_{f_5} & \\
ogg2/paziente & \left[\begin{array}{ll} cat & [umano] \\ pred & bambino \\ gen & mas \\ num & sing \\ det & det \end{array}\right]_{f_4} &
\end{array}\right]_{f_1}$$

Figure 1: f-struttura di *La mamma regala un giocattolo al bambino.*

Una f-struttura può essere convenientemente rappresentata come un DAG (grafo aciclico orientato)



in cui ciascun nodo rappresenta un valore (un simbolo o una f-struttura).

Ciascun arco orientato rappresenta un attributo. L'applicazione di funzioni può così essere vista come la definizione di un cammino nel grafo.

Il grafo è rappresentato da una serie di fatti prolog:

```
arco(DaNodo,ANodo,Attributo,Rete).
nodo(Nodo,Valore,Rete).
```

Le funzioni di accesso sulla rappresentazione a DAG possono essere ridotte alla ricerca di cammini: pertanto in Prolog è possibile esprimere l'applicazione di f-strutture come:

f_1 : `sogg/agente : numero : Valore`

ma è anche possibile non istanziare altri elementi del cammino:

`F : sogg/agente : numero : singolare`

(onde trovare la f-struttura che ha un soggetto di numero singolare), oppure trovare un cammino tra due valori:

f_i : `path(X)` : f_j

ed in questo caso X verrà istanziato al cammino tra f_i ed f_j.

3 Il risolutore dei riferimenti interni alla frase

È possibile individuare un certo numero di regole linguistiche che fissano quale deve essere la posizione strutturale relativa fra antecedente e pronome. Queste regole variano a seconda del tipo di pronome, come illustrato dai seguenti esempi:

Pronomi riflessivi Devono trovare un antecedente nella frase in cui si trovano. I riflessivi lunghi (*sè*) devono avere per antecedente un soggetto, mentre quelli corti (*se stesso*) no.

1. *Mia madre crede che Mario$_j$ porterà con sè$_j$ il suo amico.*
2. *Solamente io potevo riconciliare mio fratello$_i$ con sestesso$_i$.*

Il pronome anaforico *proprio* è simile al riflessivo corto, ma può salire nelle frasi a livello superiore:

1. *Lui ritiene che Mario$_j$ ami la propria$_j$ sorella.*
2. *Lui$_i$ ritiene che la propria$_i$ sorella ami Gino.*

Pronomi liberi, clitici e possessivi Non possono trovare un antecedente nel loro livello di frase, ma devono cercarlo ad un livello superiore o nel discorso.

1. [*Mario$_j$ ha detto* [*che lui lo$_j$ aspetta*]].

Qui *lui* sarà riferito a livello di discorso.

I soggetti inespressi di frasi di modo finito (*pro*) o di modo non finito (*Pro*) e i possessivi seguono regole analoghe.

Un altro problema è quello di determinare quali tipi di teste nominali possano essere considerate come antecedenti. Per poter trattare con uniformità i vari tipi di pronomi e di antecedenti, nella rappresentazione della frase fornita in output dal parser ogni sintagma nominale viene dotato di una lista *dei tratti di referenzialità*. Infatti nelle f-strutture non compaiono le categorie grammaticali. I tratti di referenzialità permettono di caratterizzare univocamente i vari tipi di sintagma nominale. L'algoritmo di risoluzione delle anafore utilizza unicamente questi tratti.

Nella tab. 1 sono riportate le liste dei tratti attribute a seconda della categoria lessicale. Il segno + posto davanti ad una marca attribuisce ad essa il valore vero, mentre il segno − funge da negazione.

Per quanto riguarda poi il significato delle singole marche abbiamo:

ref significa referenziale, cioè che può essere oggetto di un riferimento indipendente nel discorso: un elemento $+ref$ può essere un potenziale antecedente.

pro significa pronominale, cioè che deve essere riferito o a livello di frase (fuori della frase minima) o nel discorso.

ana significa anaforico, cioè che deve essere legato ad un antecedente o localmente (nella stessa frase minima in cui si trova), o in un dominio più ampio (in questo caso deve essere anche $+pro$). In certe condizioni può avere interpretazione arbitraria.

Categoria Lessicale	*Lista dei Tratti*
nome comune	$[+ref, -pro, -ana, +class]$
nome proprio	$[+ref, -pro, -ana, -class]$
possessivo	$[+ref, +pro, -ana, +me]$
riflessivo lungo	$[-ref, +pro, +ana, +me, +subj]$
riflessivo corto	$[-ref, -pro, +ana, +me, -subj]$
clitico	$[+ref, +pro, +ana, +me]$
pronome anaforico	$[-ref, +pro, +ana, +me, -subj]$
pronome libero	$[+ref, +pro, -ana, +me]$
pronome quantificato	$[+ref, -pro, -ana, +me]$
Pro	$[-ref, +pro, +ana, -me]$
pro	$[+ref, +pro, -ana, -me]$

Table 1: Tratti di referenzialità per le categorie lessicali

class significa classe, e serve a distinguere i nomi comuni dai nomi propri: i primi individuano una classe di elementi, i secondi una singola entità.

subj significa subjective, ed è una sottoclassificazione degli elementi anaforici. Se un elemento $+ana$ può essere legato con un antecedente che si trova in un contesto più ampio dalla frase minima in cui esso è situato, il tratto *subj* indica se l'antecedente deve essere un soggetto.

me significa *morfologicamente espresso* e indica se un pronome è lessicalmente espresso oppure è implicito.

Dalla tabella 1 si deduce che gli elementi pronominali da risolvere saranno quelli aventi nella relativa matrice di referenzialità i tratti $+pro$ o/e $+ana$, mentre i possibili antecedenti di un riferimento anaforico saranno quelli etichettati $+ref$.

Il ruolo fondamentale delle matrici dei tratti è quello di far scattare la regola di risoluzione adeguata per ogni tipo di riferimento anaforico.

Le regole del risolutore sono molto dettagliate, in quanto si devono coprire un alto numero di casi particolari presenti nella lingua italiana; esse incorporano vincoli di varia natura. Volendo schematizzare troviamo:

- vincoli strutturali
- vincoli sintattici
- vincoli semantici

3.1 Vincoli strutturali

Uno dei classici problemi della linguistica è la definizione esatta della configurazione strutturale entro la quale un SN lessicale può avere il ruolo di antecedente di un elemento anaforico. Cercare di dare una risposta a tale quesito significa individuare delle regole generali che stabiliscano in quale posizione all'interno di una frase possa essere situato l'antecedente di un elemento anaforico.

Perciò abbiamo realizzato una regola applicabile alla descrizione funzionale della frase, denominata **f-comando** in contrapposizione al *c-command*[1] presentato in [Reinhart 76], che qui di seguito enunciamo:

F-comando Per ogni occorrenza delle funzioni A e B in una *F*-struttura F, A f-comanda B se e soltanto se A non contiene B ed ogni *F*-struttura di F che contiene A contiene anche B.

Pertanto il primo passo nella ricerca dell'antecedente di un pronominale è la ricerca dell'insieme degli elementi $+ref$ che lo *f*-comandano. Questo insieme verrà filtrato da regole specifiche in base ai tratti di referenzialità del pronominale.

L'implementazione dell'*f*-comando si rivela particolarmente efficace e sintetica grazie alla struttura dati adottata. La regola seguente indica che la *f*-struttura `Alpha` *f*-comanda la *f*-struttura `Beta`(dove ":" è un operatore infisso associativo a destra):

```
f_comando(Alpha,Beta,Livello):-
        Fs : Funzione : Beta,
        f_comando_aux(Fs,Alpha,0,Livello).
```

dove `Fs` è la *f*-struttura che domina immediatamente `Beta`. Avremo quindi:

```
f_comando_aux(Fs,Alpha,Liv,Liv):-
        Fs : Funzione : Alpha.
```

che avrà successo quando esiste una *f*-struttura `Alpha` anch'essa dominata direttamente da `Fs`, mentre

```
f_comando_aux(Fs,Alpha,Liv,Liv1):-
        L1 is Liv +1,
        Fs1 : Funzione1 : Fs,
        f_comando_aux(Fs1,Alpha,L1,Liv1).
```

sarà verificata quando l'*f*-struttura `Alpha` è dominata da una *f*-struttura `Fs1` che a sua volta domina `Fs` in cui è inclusa `Beta`.

L'argomento `Livello` è la distanza in termini di livelli di *f*-strutture che intercorre tra il riferimento e il nodo che *f*-comanda. Perciò risulta immediato l'imporre condizioni sui livelli di grafo che debbono essere scandagliati dall'*f*-comando per i vari tipi di pronome.

3.2 Vincoli sintattici e semantici

Dalla sintassi possono essere ricavate diverse regole sulla pronominalizzazione. Tali regole possono essere usate per eliminare antecedenti che altrimenti potrebbero essere considerati accettabili.

Oltre alla semplice concordanza del genere e del numero della coppia antecedente-pronome, ben più utili si rivelano le funzioni grammaticali dei costituenti che, come abbiamo visto precedentemente, sono anch'esse informazioni direttamente disponibili in una f-struttura. Un riflessivo deve avere come antecedente un soggetto (se quest'ultimo si trova nello stesso livello di frase o se il riflessivo è $+subj$). Inoltre se il pronominale si trova in posizione di soggetto l'*f*-comando deve scattare dal livello di frase immediatamente superiore.

[1] *C-command* è l'abbreviazione di *constituent-command*. I domini definiti in termini di c-comando fanno capo infatti ai costituenti grammaticali F, SN, SP, SA, SV

Nonostante che una f-struttura non possa essere considerata come una rappresentazione semantica della frase, essa ingloba informazioni di carattere semantico che risultano anch'esse utili nella scelta di un antecedente. Tali informazioni sono:

- tratti semantici
- ruoli tematici

Il matching sui tratti semantici per la coppia antecedente-pronome viene posto per scartare antecedenti che altrimenti risulterebbero corretti.

I ruoli tematici permettono di trattare casi particolari, come nella frase seguente:

La salute della *propria* moglie preoccupa Mario.

in cui compare un verbo psicologico. In questo caso il soggetto, avente il ruolo di *tema*, non è un valido antecedente, ma va selezionato l'oggetto (*Mario*) poichè ha il ruolo di *esperiente*.

Spesso l'algoritmo propone per ciascun pronome più di un possibile antecedente. A questi vengono associati dei pesi calcolati sulla base del ruolo tematico, della funzione grammaticale e della referenzialità della categoria lessicale degli antecedenti stessi, che permettono di individuare l'antecedente più probabile.

Vediamo i risultati forniti dal risolutore dei riferimenti interni applicato alla frase:

(Pro) Parlando di Avveduti, Trabucchi ha detto che lo aspetta.

la cui *f*-struttura è illustrata in fig. 2.

```
[parlando,di,avveduti,',',trabucchi,ha,detto,che,lo,aspetta]
Indice della rete: ex36
PRONOMINALI: sn5(Pro)   sn17(lo)   sn16(pro)
CONTROLLATI: controllato(sn16,sn6)
RISOLVO : [sn17/[+ref,+pro,+ana,+me],sn5/[-ref,+pro,+ana,-me]]
Nodi che f-comandano sn 17:
     sn6/trabucchi/1
     Antecedente gia' utilizzato
     f5/1
     f3/2
sn1/avveduti/2
bene
esterno(ex36,sn17)
Possibili Antecedenti di sn17:   [sn1/avveduti/2,nessuno]

Nodi che f-comandano sn5:
     sn6/trabucchi/2
     bene
     f5/2
     f3/2
esterno(ex36,sn5)
Possibili Antecedenti di sn5:    [sn6/trabucchi/0,nessuno]
```

```
Rete ex36
indice:f1
principale:indice:f3
           pred:dire
           modo:ind
           tempo:pass/comp/avere
           cat:operazione
           sogg/agente:ref_tab:[+ref,-pro,-ana,-class]
                       indice:sn6
                       cat:[umano]
                       pred:trabucchi
                       gen:mas
                       num:sing
           ogg2/paziente:pred:indefinito
           ogg/prop:indice:f5
                    pred:aspettare
                    modo:ind
                    tempo:pres/semp
                    cat:operazione
                    sogg/agente:ref_tab:[+ref,+pro,-ana,-me]
                                indice:sn16
                                cat:[umano]
                                pred:pro
                                gen:sing
                                num:3
                                caso:[nom]
                                controllore:sn6
                    ogg/tema:ref_tab:[+ref,+pro,+ana,+me]
                             indice:sn17
                             cat:[umano]
                             pred:lo
                             pers:3
                             gen:mas
                             num:sing
                             caso:[acc]
subordinata:gerundiva:indice:fgerund12
                      pred:parlare
                      modo:ger
                      tempo:pres/semp
                      cat:operazione
                      sogg/agente:ref_tab:[-ref,+pro,+ana,-me]
                                  indice:sn5
                                  cat:[umano]
                                  pred:pPro
                                  controllore:indefinito
                      obl/tema:ref_tab:[+ref,-pro,-ana,-class]
                               indice:sn1
                               cat:[umano]
                               pred:avveduti
                               gen:mas
                               num:sing
```

Figure 2: *f*-struttura di *Parlando di Avveduti, Trabucchi ha detto che lo aspetta*

4 Il risolutore dei riferimenti nel discorso

Risolvere un riferimento anaforico avente il relativo antecedente al di fuori della frase in cui si trova comporta che vi sia la capacità da parte del risolutore di elaborare e di gestire un modello di discorso.

Secondo le più moderne teorie [Grosz 86] la struttuta di un discorso è composta da tre componenti separate ma correlate tra loro: la struttura della sequenza delle frasi (detta *struttura linguistica*), la struttura delle finalità (detta *struttura intenzionale*), e lo stato del centro (*focus*) dell'attenzione (detto *stato attenzionale*). La distinzione tra queste componenti è essenziale per fornire un'adeguata spiegazione di alcuni fenomeni legati al discorso come ad esempio le espressioni che introducono un cambiamento dei centri del discorso (*cue phrases*), i riferimenti anaforici e le interruzioni temporanee.

In particolare lo stato attenzionale è un'astrazione del centro dell'attenzione dei partecipanti durante l'evolversi del discorso. Essendo dinamico, lo stato attenzionale tiene conto degli oggetti, delle proprietà e delle relazioni che sono salienti ad ogni punto del discorso.

Alla base del modulo per la risoluzione dei riferimenti esterni quindi abbiamo sviluppato un algoritmo per la costruzione di un modello della struttura attenzionale del discorso. Tale approccio si rifa al modello *centering*, ovvero basato sui centri del discorso, presentato in [Brennan 86].

Il lavoro compiuto da tale modulo per ogni frase che gli viene fornita in input può essere suddiviso in tre differenti fasi:

1. Estrazione dei centri del discorso.
2. Risoluzione dei riferimenti esterni.
3. Gestione degli stati del discorso.

Di seguito esaminiamo in dettaglio queste tre fasi, tenendo presente che le ultime due sono strettamente legate.

4.1 Estrazione dei centri del discorso

Prima di tutto si estrae dal grafo della frase corrente una lista di tutti i sintagmi nominali presenti.

Quindi vengono sostituite le teste di quei sintagmi nominali contenenti riferimenti che in un qualche modo siano già stati risolti, o a livello grammaticale (i riferimenti controllati lessicalmente), o dal modulo precedente, con le teste dei relativi antecedenti.

A questo punto tramite una funzione peso, che attribuisce un punteggio a tutti i sintagmi nominali presenti nella frase, viene introdotta una gerarchia tra i vari possibili argomenti; i fattori di cui questa funzione tiene conto sono:

- distanza del sintagma nominale dalla radice del grafo della frase
- ruolo tematico e funzione grammaticale di tale sintagma
- categoria semantica

La distanza è importante poichè intendiamo privilegiare come possibili argomenti quei sintagmi nominali che si trovano nei livelli più alti della frase.

I punteggi relativi alla funzione grammaticale, al ruolo tematico e ai tratti semantici vengono assegnati tramite tabelle presenti nel programma.

Una volta pesati i sintagmi nominali vengono ordinati in ordine crescente rispetto al punteggio attribuito loro; a questo punto un'apposita regola crea la lista degli argomenti di frase estraendo tutti i sintagmi nominali aventi il tratto *+ref* nella relativa lista dei tratti di referenzialità e quindi sottoponendoli a due filtri. Innanzi tutto se vi sono più occorrenze dello stesso sintagma verrà estratto solo quello col punteggio migliore. Inoltre saranno scartati quelli aventi un punteggio maggiore di un certo valore di soglia, poichè si presuppone che questi non possano rappresentare argomenti a cui possano riferirsi eventuali forme pronominali di frasi successive. I sintagmi nominali "sopravvissuti" a questi due filtri vengono inseriti in una lista, detta *Lista Pesata.*

4.2 Risoluzione dei riferimenti esterni e gestione degli stati del discorso

Dunque gli argomenti di una frase a questo punto risultano asseriti in memoria in una lista ordinata durante la fase precedente: questo ordinamento costituisce in un certo qual modo la probabilità che ogni elemento ha di diventare un centro del discorso. In questa lista individuiamo quattro tipi di argomento:

Argomento Principale è il primo della lista e costituisce l'argomento principale del discorso.

Argomento Secondario è il secondo nella lista e costituisce l'elemento avente maggior probabilità di divenire argomento principale nella frase seguente ogniqualvolta vi sia un cambiamento del centro del discorso ed un conseguente cambiamento di stato.

Argomento Proposto è il primo elemento della lista quando lo stato attuale non permette di stabilire con certezza un argomento principale ma solo di formulare delle ipotesi (ad esempio nella prima frase di un discorso).

Altri Argomenti tutti gli elementi della lista ordinata che non siano il principale, il secondario o il proposto.

Gli stati del discorso da noi individuati sono quattro, e precisamente:

CONTINUING indica che nella frase corrente gli argomenti corrispondono a quelli fissati per la frase precedente.

RETAINING è lo stato che prelude ad un possibile cambiamento dei centri del discorso, quando nella frase corrente siano stati introdotti nuovi argomenti.

SHIFTING indica un cambiamento dei centri del discorso, ma solo se in presenza di una frase a costruzione marcata o topicalizzata.

CHANGING indica un normale cambiamento dei centri del discorso.

Vediamo ora di capire come vengono gestiti al lato pratico i passaggi da uno stato all'altro.

Consideriamo quindi un brano tratto da un articolo di giornale. Nella prima frase,


```
Gli piaceva parlare del suocero come di una facile occasione mancata,
che chiunque altro avrebbe sfruttato, ma che lui, Avveduti,
preferiva lasciar perdere.
```

il risolutore fornisce:

```
Lista Pesata: [avveduti/22,suocero/33,chiunque/60,occasione/100]
Arg. Prop.: avveduti  Arg. Princ.: nil  Arg. Sec.: nil
Altri Argomenti: [suocero,chiunque,occasione]
Stato: CHANGING
```

Poichè si tratta della prima frase si ha solo l'argomento proposto e lo stato viene fissato a *changing.*

A questo punto l'analizzatore passa ad esaminare la seconda frase:

```
Solo verso il 1950 (pro) si rassegnò ad accettare un posto nell'organizza-
zione della fiera di Verona.
```

```
Lista Pesata: [pro/10,organizzazione/50,posto/1025,fiera/1040,verona/2050]
Arg. Prop.: nil  Arg. Princ.: avveduti  Arg. Sec.: nil
Altri Argomenti: [organizzazione,posto,fiera,verona]
Esterni: pro ---> avveduti
Stato: CONTINUING
```

Il pronome *si* è risolto dall'algoritmo interno alla frase. Poichè lo stato della frase precedente è *changing* si può riferire il *pro* all'argomento proposto. In questo modo *avveduti* diventa *argomento principale* e lo stato è *continuing.*

Siamo giunti alla terza frase:

```
Lo (pro) nominarono delegato, cioè una specie di funzionario...
```

```
Lista Pesata: [pro/10,lo/14,funzionario/50]
Arg. Prop.: pro  Arg. Princ.: avveduti  Arg. Sec.: nil
Altri Argomenti: [funzionario]
Esterni: lo ---> avveduti
Stato: RETAINING
```

Il *pro* (primo elemento della lista), essendo plurale, non coincide con l'argomento principale, quindi lo stato non può essere continuing. Tuttavia il clitico *lo* può essere riferito all'argomento principale della frase precedente. Lo stato individuato è quindi *retaining* e l'argomento principale rimane invariato.

Siamo giunti ora alla quarta frase:

```
Un compito che corrispondeva bene alla sua vocazione ...
```

```
Lista Pesata: [avveduti/30,pPro/1030,suo/1041,vocazione/1043]
Arg. Prop.: nil  Arg. Princ.: avveduti  Arg. Sec.: nil
Altri Argomenti: [pPro,avveduti,vocazione]
Esterni: suo ---> avveduti
Stato: CONTINUING
```

Poichè *suo* può essere riferito all'argomento principale della frase precedente, lo stato ritorna ad essere *continuing*. L'argomento proposto della terza frase viene scartato.

Nella quinta frase:

```
Quando il suocero morì, egli non perse il posto.

Lista Pesata: [egli/20,suocero/25,posto/1025]
Arg. Prop.: nil  Arg. Princ.: avveduti  Arg. Sec.: nil
Altri Argomenti: [suocero,posto]
Esterni: egli ---> avveduti
Stato: CONTINUING
```

Egli può essere riferito ad *Avveduti*, quindi si rimane in *continuing*.

Osserviamo ora la sesta frase del nostro brano:

```
A Verona il collegio di Alberti lo aveva ereditato Trabucchi,...

Lista Pesata: [trabucchi/20,collegio/30,alberti/41,verona/2050]
Arg. Prop.: nil  Arg. Princ.: trabucchi  Arg. Sec.: avveduti
Altri Argomenti: [collegio,verona]
Stato: SHIFTING
```

Poichè abbiamo un **topic** (*collegio*) e un **focus** (*Trabucchi*) lo stato diviene *shifting*.

Per la settima frase:

```
Trabucchi continuo' a valersi della collaborazione di Avveduti.

Lista Pesata: [trabucchi/10,avveduti/30,collaborazione/103]
Arg. Prop.: nil  Arg. Princ.: trabucchi  Arg. Sec.: avveduti
Altri Argomenti: [avveduti,collaborazione]
Stato: CONTINUING
```

il primo elemento della lista pesata (*Trabucchi*) coincide con l'argomento principale della frase precedente, quindi si passa in *continuing*.

Mentre per l'ottava frase:

```
L'ex ufficiale del Novara Cavalleria gli era simpatico.

Lista Pesata: [gli/30,ex-ufficiale/1010,novara-cavalleria/1020]
Arg. Prop.: nil  Arg. Princ.: trabucchi  Arg. Sec.: nil
Altri Argomenti: [avveduti,ex-ufficiale,novara-cavalleria]
Esterni:  gli ---> trabucchi
Stato: RETAINING
```

il risolutore non risolve la parafrasi (*l'ex ufficiale* dal testo risulta essere *Avveduti*) quindi riferisce *gli* all'argomento principale (*Trabucchi*) e passa in *retaining*.

5 Conclusioni

Il risolutore qui presentato è stato testato su un centinaio di frasi della lingua italiana comprendenti i vari tipi di pronome in diversi contesti strutturali e su un brano tratto da un giornale.

È da sottolineare il fatto che nonostante l'uso dei pronominali vari da lingua a lingua, regole come L'f-comando o il modello di discorso a stati sono di validità generale e prescindono dal linguaggio al quale vengono applicate: è attualmente in fase di sviluppo una versione del risolutore per la lingua tedesca.

Va inoltre sottolineato come la LFG fornisca una rappresentazione della frase particolarmente adeguata: essa contiene infatti tutte le informazioni necessarie al risolutore. Pertanto la risoluzione dei riferimenti interni alla frase può essere effettuata basandosi principalmente su regole grammaticali e su limitati elementi semantici derivati dal lessico.

Solo alcuni casi non vengono trattati correttamente dal risolutore dei riferimenti interni, e ciò si verifica in special modo in presenza di ambiguità, come nel seguente esempio:

L'insegnante promosse lo studente poichè $_{pro}$ *era preparato.*
L'insegnante promosse lo studente poichè $_{pro}$ *era di buon umore.*

In questo caso l'ordinamento dei possibili antecedenti del *pro* seleziona il soggetto della frase principale, dando un risultato corretto per la seconda frase, ma non per la prima. Le due frasi infatti hanno la stessa struttura e solo conoscenze esterne potrebbero differenziare il comportamento del risolutore.

Anche l'analisi a livello di discorso presuppone un'analisi dettagliata a livello grammaticale: essa infatti permette di individuare con sicurezza i cambiamenti dei centri del discorso, anche nei casi più particolari, come i costrutti topicalizzati tipici della lingua italiana (stato di *shifting*).

Solo in un numero limitato di casi si rivela necessario ricorrere a conoscenze esterne. Ad esempio nell'ultima frase del brano analizzato, dove è presente un riferimento anaforico tramite parafrasi. Qui l'impossibilità di rimanere in uno stato di *continuing* dovrebbe far scattare un meccanismo onde ricorrere ad una base di conoscenze e vedere se *l'ex ufficiale* può essere riferito ad *Avveduti*. I futuri sviluppi del risolutore saranno quindi dettati da questa esigenza: far sì che i due moduli possano interagire tra loro e soprattutto con una base di conoscenze specifiche del dominio cui il testo riferisce.

6 Ringraziamenti

Desideriamo ringraziare il prof. G.A. Lanzarone del Dipartimento di Scienze dell'Informazione dell'Università di Milano per le utili discussioni e l'incoraggiamento a svolgere il lavoro.

References

[Brennan 86] S. E. Brennan, M. W. Friedman, C. J, Pollard *A centering approach to pronouns*. Proc. 25th annual meeting of Association for Computational Linguistics, (1986).

[Bresnan 80] J. W. Bresnan, *The Mental representation of grammatical relations.* MIT Press, Cambridge, MA (1980).

[Bullwin 77] C. Bullwinkle, *Levels of complexity in discourse for Anaphora Disambiguation and Speech Act Interpretation.* Proc. 5th International Joint Conference of Artificial Intelligence, (1977).

[Dalrymp 90] M. E. Dalrymple, *Syntactic constraints on anaphoric binding.* Tesi di Ph.D., Stanford University, Stanford (June 1990).

[Ehrlich 81] K. Ehrlich, *Search and inference strategies in pronoun resolution: an experimental study.* Proc. 19th Annl. Meeting Assn. Computational Linguistics, Stanford, CA (1981).

[Enç 89] M. Enç, *Pronouns, Licensing, and Binding, Natural Language and Linguistic Theory*

[Frugoni 90] P. Frugoni, *Studio e realizzazione di programmi logici per la risoluzione dell'anafora in italiano.* Tesi di Laurea, Università di Milano, Facoltà di Scienze dell'Informazione (Ottobre 1990).

[Grosz 86] B. J. Grosz, C. L. Sidner, *Attention, Intentions and the Structure of Discourse.* da Computational Linguistics, Volume 12, Num. 3, July-Sept 1986.

[Hirst 81] Graeme Hirst, *Anaphora in Natural Language Understanding: A Survey.* Springer-Verlag, Berlin (1981).

[Pereira 83] F. Pereira, *Logic for Natural Language Analysis.* SRI International, Menlo Park, California (1983), Technical Note 275.

[Reinhart 76] T. Reinhart, *The Syntactic Domains of Anaphora.* Tesi di PhD, MIT Press, Cambridge, MA (1976).

[Sidner 83] C. L. Sidner, *Focusing in the comprehension of definite anaphora.* da Computational Models of Discourse, MIT Press, Cambridge, MA (1983).

CONFIGURAZIONE DI L.A.N.: UNO STRUMENTO INTELLIGENTE DI SUPPORTO ALLE DECISIONI

F.Medelin, F.Rota, L.Tanca, A.Zarattini

Dipartimento di Elettronica - Politecnico di Milano

ABSTRACT

ELAND, (Expert LAN Designer), is a Prolog tool to assist the design of a local area network of personal computers. The expert system solves the main aspects of the configuration: node-connectivity problems, the choice of the computers to be linked by the network (both workstations and servers are considered), and the kind of software which better meets the application and service requirements of the network users. The main topic in this paper is the design of both the Knowledge Base and the Inference Engine.

1. Introduzione

Questo articolo presenta ELAND, un sistema esperto Prolog di ausilio alla configurazione di una rete locale di personal computer. Al suo interlocutore, cioe' al progettista della rete, ELAND indica gli elementi utili per la scelta dei prodotti che meglio soddisfano le esigenze applicative dell' organizzazione che ne ha deciso l'installazione. Questo avviene in modo interattivo, come durante un colloquio con un esperto del settore.

1.1 Natura del problema

L'offerta di reti locali (Local Area Networks: LAN) sul mercato si compone di prodotti con caratteristiche e prestazioni differenti. La scelta di un prodotto si rivela quindi impegnativa e complessa per il progettista della rete, che puo' essere un consulente o il responsabile EDP di una organizzazione. Egli in genere segue una metodologia di progetto personale, che comporta pero' un certo numero di passi abbastanza standard. Tenendo presenti questi passi, abbiamo ipotizzato il procedimento che segue.
In primo luogo si determinano le caratteristiche fondamentali della rete e dei nodi da collegare (topologia, metodo d'accesso, microprocessore, memoria di massa, software di rete...) in base ai vincoli ed ai requisiti piu'

critici del caso in esame (disponibilita' di budget, grado di affidabilita' e di sicurezza, quantita' di dati da gestire, ambiente software, servizi di rete, ...). In tal modo viene creato un modello della rete.
A questo punto si e' in grado di confrontare questa soluzione "teorica" con le caratteristiche dei prodotti presenti sul mercato, riducendone notevolmente lo spettro. Proprio su questo numero limitato di prodotti ci si puo' concentrare per effettuare, in un secondo tempo, un'analisi piu' approfondita che tenga conto degli altri vincoli, di esigenze e preferenze che determinano in realta' la scelta finale della configurazione (ad esempio prestazioni in particolari condizioni critiche, garanzie sulle condizioni di vendita e di assistenza, preferenze verso determinati produttori o concessionari ...).
La fase di scelta definitiva dei prodotti si puo' ritenere molto soggettiva, in quanto spesso compaiono elementi valutativi difficilmente esprimibili. La prima fase, di determinazione della configurazione "ideale", e quella della scelta dei prodotti ad essa piu' fedeli, sono indipendenti invece dalla tipologia dell'offerta del mercato e costituiscono un processo che puo' essere automatizzato o quantomeno assistito dal calcolatore. E' proprio in questa fase che ELAND supporta l'attivita' decisionale del progettista.

1.2 Perche' un sistema esperto Prolog

L'approccio al problema mediante un sistema esperto Prolog ci e' sembrato appropriato per una serie di motivi:

- Il problema per sua natura richiede una manipolazione simbolica.
- Il dominio di conoscenza relativo al problema e' sufficentemente circoscritto e chiaro: si e' infatti deciso di considerare solo reti locali aventi personal computer come stazioni di lavoro. Questa scelta e' motivata anche dall' interesse che tale soluzione suscita nell'ambito dell'automazione del lavoro d'ufficio e, piu' in generale, dell'informatica dipartimentale. In corrispondenza di casi particolari, cioe' al limite del dominio, il sistema fornisce soluzioni parziali, ma sempre consistenti.

- Poiche' gli esperti del settore sono pochi e non esistono procedure standard per configurare una LAN, l'uso di euristiche e "backtracking" sembra il modo di procedere piu' opportuno.

- Un aspetto molto importante e' l'elevato grado di interattivita' che un sistema esperto garantisce all'utente nella guida alla soluzione finale, fornendo

eventualmente la spiegazione delle scelte fatte e dei risultati conseguiti.

- Nel progetto della configurazione di una LAN gli esperti sono guidati soprattutto dall'esperienza accumulata nella risoluzione di casi pratici: questo tipo di conoscenza e' esprimibile in modo immediato sotto forma di fatti e regole Prolog.

Questo articolo e' organizzato come segue: nel Capitolo 2 e' illustrato l'approccio al problema, nel Capitolo 3 e' descritta l'architettura del sistema esperto, il Capitolo 4 descrive come e' stata strutturata la conoscenza e il formalismo utilizzato per la sua rappresentazione, infine il Capitolo 5 e' dedicato al motore inferenziale e ai due moduli che lo compongono.

2. Caratteristiche generali

Il problema generale della configurazione e' molto complesso, ed e' percio' opportuno affrontarlo scomponendolo in sottoproblemi piu' elementari. Seguendo un approccio alla scomposizione di tipo funzionale, in ELAND sono stati individuati tre sottoproblemi principali:

- configurazione della rete fisica, cioe' scelta del tipo di connessione dei nodi della rete;
- scelta degli elaboratori, che costituiscono le stazioni di lavoro ed i server della rete;
- scelta del software di rete, che fornisce le applicazioni ed i servizi di rete.

Ognuno di questi viene a sua volta ulteriormente scomposto, ottenendo una gerarchia di sottoproblemi, la cui soluzione porta alla determinazione della configurazione del sistema.
La scomposizione in sottoproblemi e' concettualmente rappresentabile mediante una struttura tipo albero (in realta' e' un grafo orientato aciclico). Ogni passo di tale scomposizione puo' introdurre una SEQUENZA di sottoproblemi da affrontare, oppure una SCELTA tra modi di procedere alternativi, o infine puo' rappresentare la SOLUZIONE di un sottoproblema "elementare" di progettazione. Esistono pertanto tre tipi di nodi:
- nodi di Sequenza (AND): i nodi figli costituiscono un insieme di sottoproblemi piu' elementari che devono essere tutti esaminati;
- nodi di Scelta (OR): relativi a una scelta tra due o piu' alternative progettuali, rappresentate dai figli del nodo stesso;
- nodi Terminali: rappresentano l'ultimo livello di scomposizione dei problemi e ne rappresentano la soluzione elementare.

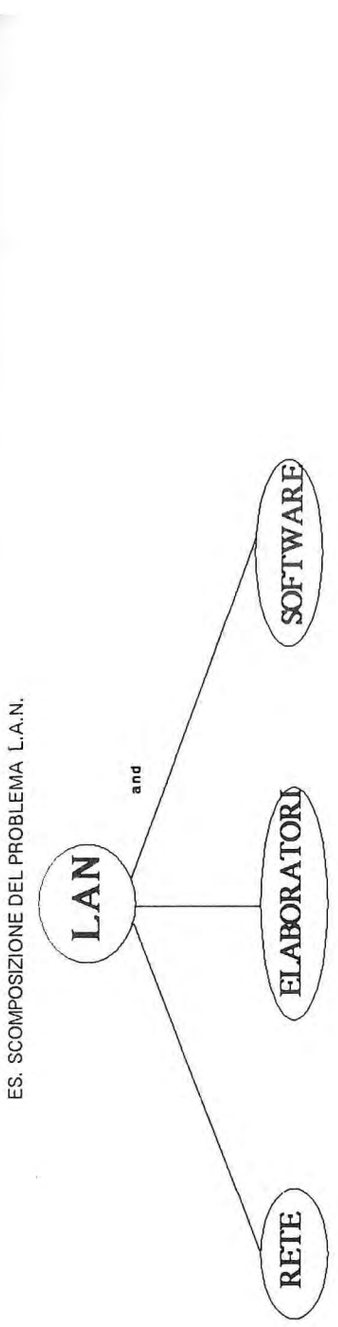
Nella figura 1 osserviamo che la configurazione del sistema e' determinata dall'insieme delle soluzioni dei tre sottoproblemi principali (RETE FISICA, SISTEMI e SOFTWARE); essi sono figli del nodo LAN che e' un nodo AND e rappresenta il problema iniziale. La figura 2 mostra l'esplosione del sottoproblema di primo livello RETE FISICA.
Si puo notare dalla figura che la struttura risultante e' in effetti piu' generale di un albero, e precisamente un grafo orientato aciclico (DAG).

ELAND e' stato realizzato in Prolog. La doppia semantica che caratterizza questo linguaggio consente sia all'ingegnere della conoscenza che al programmatore di disporre di uno strumento di sviluppo adatto alle proprie esigenze. Infatti la natura dichiarativa del linguaggio ha facilitato la creazione della base della conoscenza, mentre nella implementazione della strategia di "problem solving" si e' visto Prolog come linguaggio procedurale, con l'utilizzo dei cosiddetti predicati "impuri" che permettono al programmatore di variare il controllo dell'inferenza dove ritenuto necessario.
L'interprete Prolog fornisce un controllo dell'inferenza di tipo concatenamento all'indietro ("backward chaining") ed una strategia di ricerca in profondita' ("depth first"): la risoluzione dei conflitti di regole e' basata sull'ordine di queste nella base della conoscenza. Tuttavia il problema in esame richiede per sua natura di controllare dinamicamente l'uso delle regole. La strategia di ricerca fornita da Prolog e' stata percio' arricchita con alcune regole euristiche, dette metaregole: esse possono modificare l'attraversamento del grafo, cioe' l'ordine con cui i sottoproblemi vengono affrontati. Per effetto di cio', le decisioni piu' importanti vengono prese per prime e quindi possono influenzarne altre meno critiche. Viene cosi' ridotta la necessita' di ritorno sulle decisioni prese ("backtracking"), con un sensibile aumento dell'efficienza del sistema.

3. Architettura

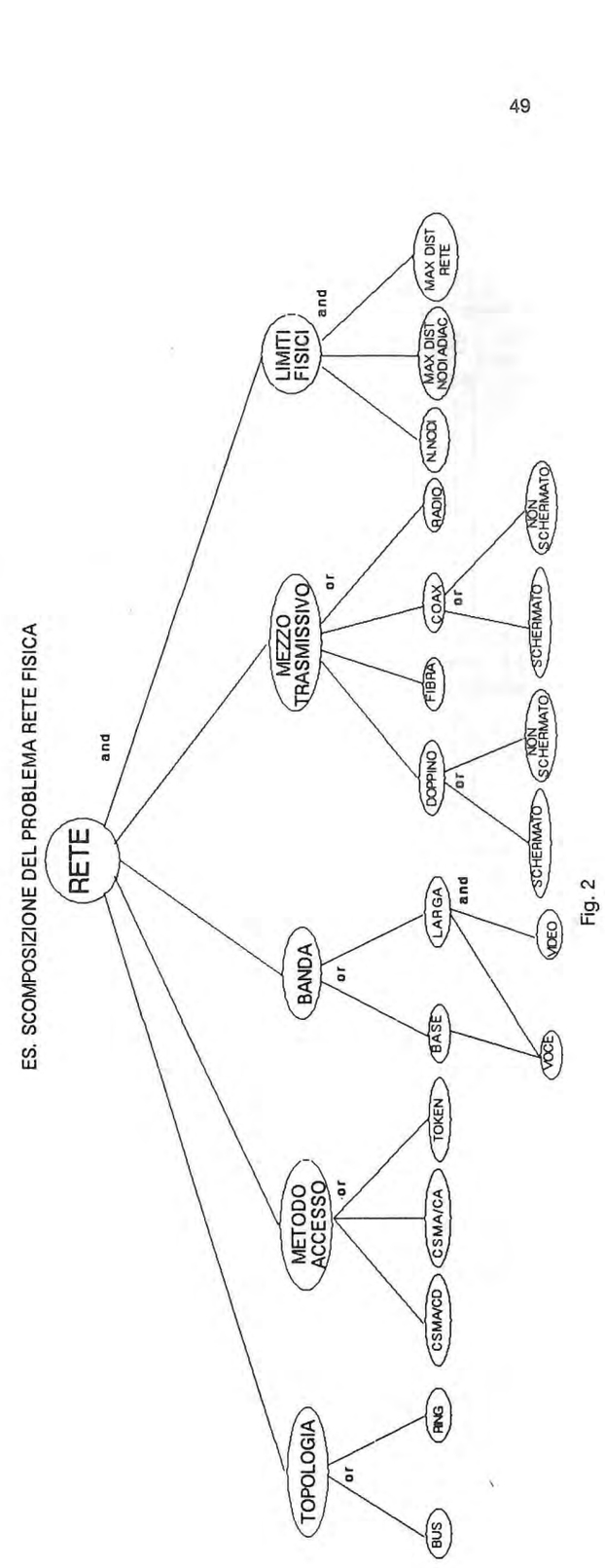
I moduli che costituiscono il sistema esperto con le loro interazioni sono illustrati nella figura 3. Si e' cercato di dare loro la massima indipendenza; in particolare lo sforzo maggiore e' stato rivolto alla separazione del motore inferenziale generico dalla conoscenza riguardante il nostro specifico problema. I principali vantaggi di questa scelta sono:

- la facilita' con cui si puo' incrementare la base della conoscenza;
- la possibilita' di ottenere facilmente spiegazioni del comportamento del sistema;



ES. SCOMPOSIZIONE DEL PROBLEMA L.A.N.

Fig. 1



ES. SCOMPOSIZIONE DEL PROBLEMA RETE FISICA

Fig. 2

ARCHITETTURA DEL SISTEMA

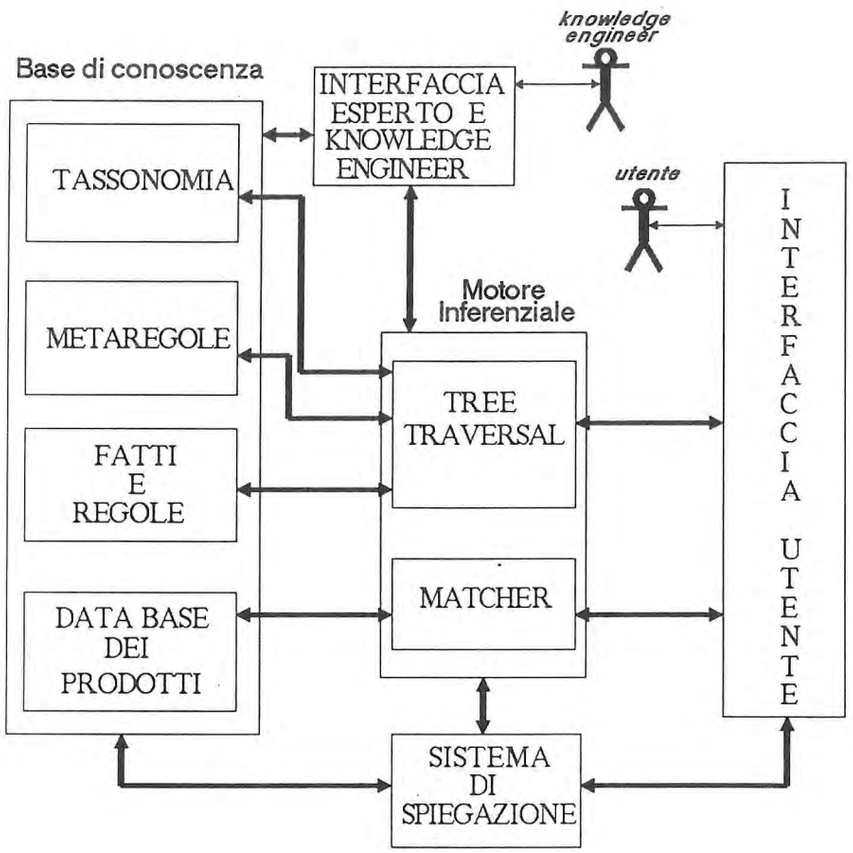


Fig. 3

- la possibilita' di utilizzare il motore inferenziale per altri problemi dello stesso tipo, cioe' problemi la cui base di conoscenza si rappresenta bene con una struttura a grafo orientato aciclico.

In questo articolo esaminiamo principalmente l'organizzazione della base della conoscenza e le funzionalita' del motore inferenziale. Una descrizione piu' dettagliata e completa del sistema puo' essere trovata in [16].

4. Base della conoscenza

4.1 Metaconoscenza

La metaconoscenza consiste nella descrizione del modello concettuale del problema e le euristiche che determinano il percorrimento dinamico dell'albero, entrambe rappresentate tramite regole di produzione.

4.1.1 Modello concettuale

Il modello concettuale del problema e' realizzato rappresentando ogni (sotto)problema mediante un nodo del grafo. Tale nodo si compone di una struttura dati con la quale si formalizza il grafo e di un insieme di informazioni che descrivono la semantica del (sotto)problema corrispondente.
La struttura dati contiene l'identificatore del (sotto)problema, i parametri che lo caratterizzano, gli identificatori dei sottoproblemi nei quali puo' essere ulteriormente scomposto.
Le informazioni semantiche sono chiamate "proprieta'" e sono di quattro tipi: strutturali, di ordinamento, di ciclicita', di confronto. Le proprieta' strutturali distinguono nodi AND, OR, TERMINAL (le foglie dell'albero); l'ordinamento riguarda l'ordine di attraversamento dell'albero; la ciclicita' identifica un (sotto)problema che deve essere risolto piu' volte (si noti che la proprieta' di ciclicita' di un nodo non influisce sulla aciclicita' del grafo, ma indica solo che l'operazione di analisi del nodo deve essere ripetuta piu' volte); il confronto richiede che la soluzione del corrente (sotto)problema venga verificata consultando la base dati dei prodotti.
Le proprieta' dei nodi verranno esaminate piu' dettagliatamente nella sezione relativa all'attraversamento dell'albero (sez. 5.1).
Un nodo del grafo e' implementato in Prolog con il seguente formato:

```
node(<nome_nodo>(<parametri,[<sottoproblemi>])).

<nome_proprieta'1>(<nome_nodo>[(parametri)]).
        .
        .
        .
<nome_proprieta'n>(<nome_nodo>)[(parametri)])
```

In fig. 4 sono riportati due esempi:

```
      Metodo di accesso:

node(access_met(Value,Topol,Cable,Voice),
            [csma_ca,csma_cd,token]).
ornode(access_met).
sortnode(access_met).
comparenode(access_met(Value,Topol,Cable,Voice)).

      Workstation:

node(workstation (<parametri>,
              [<sottoproblemi>])).
andnode(workstation).
sortnode(workstation).
cyclicnode(workstation).
comparenode(workstation(<parametri>).
```

fig. 4

4.1.2 Metaregole

Ad ogni figlio del nodo considerato, e quindi in corrispondenza di ogni possibile soluzione del sottoproblema, viene associato un numero, da 0 a 100 in ordine di importanza, che ne indica la rilevanza nella situazione espressa dal corpo della regola.
In fig. 5 e' illustrato un esempio in linguaggio naturale che riguarda ls scelta della topologia. La configurazione a "BUS" costituisce la migliore soluzione nella maggioranza dei casi: ad essa corrisponde quindi il valore di "default" maggiore (metaregole MR6 e MR7). Tuttavia vi sono casi in cui e' meglio prendere in considerazione la configurazione a "RING" (metaregola MR3).
Si noti che a questo punto l'ordine secondo cui sono disposte le metaregole e' significativo dato il metodo di ricerca sequenziale dell'interprete Prolog.


```
MR1     importanza  di (topologia = bus) = 85
          IF    affidabilita' = molto alta
            OR  tempo di risposta = basso
            OR  espandibilita' rete = alta / molto alta
        .
MR2     importanza  di (topologia = bus) = 70
          IF    affidabilita' = alta
            OR  espandibilita = alta

MR3     importanza  di (topologia = anello) = 90
          IF  distribuzione del traffico = a picchi

MR4     importanza  di (topologia = anello) = 80
          IF    numero di stazioni = basso / medio
           AND  tempo di risposta = basso
           AND  traffico = elevato

MR5     importanza  di (topologia = anello) = 65
          IF  traffico = alto

MR6     importanza  di (topologia = bus) = 50
          negli altri casi

MR7     importanza  di (topologia = anello) = 30
          negli altri casi
```

Fig. 5

4.2 La base di conoscenza

La conoscenza si compone di tre moduli: il primo contiene i requisiti dell'utente (CONOSCENZA RELATIVA ALL'UTENTE); il secondo le caratteristiche dei principali prodotti esistenti sul mercato (CONOSCENZA RELATIVA AI PRODOTTI). I due moduli rappresentano un tipo di conoscenza non strutturata, costituita da semplici asserzioni: esse vengono espresse in Prolog mediante fatti, che compongono la base di dati gestita dal linguaggio stesso. Il terzo modulo (CONOSCENZA RELATIVA AL PROCESSO) comprende l'insieme di regole che descrivono il processo di configurazione di una LAN.

4.2.1 Conoscenza relativa all'utente

All'inizio della sessione di lavoro l'interfaccia utente chiede al progettista di specificare le esigenze ed i vincoli di cui il progetto deve tener conto. Queste informazioni sono espresse in Prolog da fatti che associano al requisito in esame il suo valore qualitativo o quantitativo: alcuni esempi sono riportati in fig. 6.
Alcuni di questi requisiti, ritenuti fondamentali per

descrivere il caso in esame, devono essere specificati obbligatoriamente dall'utente; altri invece, se non indicati, vengono dedotti dal sistema mediante opportune regole che si basano su quanto espresso dall'utente. Nell'esempio della figure 6, questi ultimi sono evidenziati con asterisco.

```
disponibilita'_spesa(medio).          estensione_rete(500m).
elasticita'_spesa(no).                numero_stazioni_lavoro(12).
affidabilita'(media).                 espandibilita'(media)
flessibilita'(media).                 tempo_di_risposta(medio).
attivita'(ufficio).                   traffico(basso).
attivita'(elab_testi).                colleg_rete_geograf(x25)
distribuzione_traffico(picchi).       facilita'_uso(si).
rumore_elettrico_ambiente(basso).     posta_elettronica(si).
costo_staz_lavoro(basso).*            ritardo_trasmissione
                                           (probabilistico).*
affidabilita'_server(medio).*         affidabilita'_staz_lavoro
                                           (medio).*
costo_server(medio).*                 num_staz_lavoro(medio).*
velocita'_rete(media).*               mass_storage_serv(media).*
```

Fig. 6

4.2.2 Conoscenza relativa ai prodotti

La scomposizione del problema dal punto di vista funzionale in tre sottoproblemi principali (configurazione della rete fisica, dei nodi da collegare e del software), viene riproposta come classificazione dei prodotti, che quindi risultano di tre tipi: hw di rete, elaboratori, sw di rete. Ciascun tipo possiede le caratteristiche che identificano i nodi terminali del corrispondente sottoalbero e le variabili decisionali definite nel sottoalbero stesso: cio' permette di effettuare il confronto tra le soluzioni determinate dal sistema ed i prodotti esistenti nel database.
Ad esempio il prodotto hw di rete 3COM viene cosi' rappresentato in Prolog:

```
tipo(rete,'3COM').

product(topol,bus,'3COM').
product(metodo_accessso,csma_cd,'3COM').
product(banda,bandabase,'3COM').
product(mezzo_trasmissivo,coassiale,'3COM').
product(max_num_nodi,256,'3COM').
```

A ciascun componente di ogni prodotto e' associato anche il costo: ELAND puo' fornire cosi' all'utente il costo approssimativo della soluzione proposta. Naturalmente la conoscenza relativa ai costi e' la piu' suscettibile di

modifica durante la vita del sistema.

Ecco un esempio riferito ancora a 3COM (i costi sono espressi in milioni di lire it.):

```
cost(configurazione_minima,4.6,'3COM').
cost(per_connessione,1,2,'3COM').
cost(mezzo_trasmissivo,0.025,'3COM').
```

4.2.3 Conoscenza relativa al processo

Tale conoscenza e' costituita da regole mediante le quali vengono assegnati i valori alle variabili che rappresentano le soluzioni dei sottoproblemi.
Le regole sono applicate ai nodi che implicano una decisione progettuale, cioe' ai figli di nodi OR ed alcuni nodi terminali: in pratica enunciano le condizioni di applicabilita' di ogni soluzione e costituiscono la rappresentazione piu' visibile della competenza specifica dell'esperto umano.
Nella risoluzione di un (sotto)problema le condizioni di applicabilita' delle regole possono sovrapporsi: le metaregole determinano un ordine di preferenza nella scelta delle soluzioni stesse che e' valido nella situazione contingente e non in generale. Tuttavia questo non implica che le altre soluzioni vengano scartate; infatti se la soluzione che e' stata esaminata si rivelasse non valida, il sistema passerebbe ad esaminare la prossima nell'ordine dettato dalle metaregole.

5. Motore Inferenziale

Il motore inferenziale si compone di due moduli funzionalmente distinti:

- "TREE TRAVERSAL"(attraversamento dell'albero):
  interviene nella fase di sintesi costruendo un modello teorico che soddisfi le esigenze dell'utente;

- "MATCHER"(modulo di confronto):
  confronta i parametri del modello con le soluzioni offerte dal mercato.

5.1 Tree Traversal

Tree Traversal e' il modulo principale del motore inferenziale ed ha come compito l'interpretazione del modello concettuale del problema alla luce dei requisiti dell'utente. Tale compito viene svolto in due fasi distinte: una ricerca dei sottoproblemi da risolvere che si traduce in una discesa ricorsiva nello spazio delle soluzioni strutturato ad albero ed una risoluzione vera e propria dei sottoproblemi che genera la descrizione dei requisiti che la soluzione dovra' possedere.

Il modulo svolge le seguenti operazioni:

- traduce le proprieta' associate ad ogni nodo dell'albero in corrispondenti azioni di percorrimento della struttura

- applica le metaregole per modificare il cammino lungo l'albero

- applica le regole contenute nella base della conoscenza per risolvere i singoli sottoproblemi

- fornisce le soluzioni dei sottoproblemi e le integra nella soluzione complessiva

Nel suo funzionamento generale il "Tree Traversal" vede ogni nodo dell'albero come una tesi da dimostrare.
Questa tesi si considera dimostrata e il sottoproblema corrispondente completamente risolto solo quando a tutti i parametri che lo caratterizzano e' stato assegnato un valore. Le proprieta' del nodo determinano la modalita' di risoluzione del (sotto)problema corrispondente.
La prima proprieta' esaminata e' quella legata alla struttura dell'albero, in base alla quale il Tree Traversal conosce se e come un problema e' suddiviso ulteriormente e quindi quando terminare o continuare il cammino. Nel primo caso il motore inferenziale applica le regole contenute nella base della conoscenza secondo i meccanismi dell'interprete Prolog (sequenzialita' e backtracking). Nel secondo caso invece si tratta di risolvere i conflitti derivanti dalla scelta del cammino da percorrere. A tal scopo e' esaminata la proprieta' di ordinamento che puo' modificare la risoluzione dei conflitti sulla base di una classifica di priorita' costruita applicando le metaregole.

Fondamentale e' anche la proprieta' di ciclicita'. Un nodo

ciclico viene attraversato piu' volte (il numero fa parte dei requisiti dell'utente). Ogni volta la soluzione appena determinata viene memorizzata in una memoria di lavoro e vengono annullati tutti i valori temporanei assunti dai parametri durante l'attraversamento del sottoalbero. Al prossimo attraversamento l'utente ha la possibilita' di introdurre nuovi requisiti che porteranno ad una soluzione in generale diversa dalla precedente. Si realizza cosi' la possibilita' di ripercorrere ciclicamente alcuni sottografi senza rendere ciclico il grafo. Un esempio di nodo ciclico e' fornito dal nodo di primo livello "workstation". Ogni attraversamento completo del sottoalbero che lo ha per radice comporta la decisione riguardante il tipo di computer da collegare in rete dove i diversi tipi sono distinti in base alle differenti funzionalita' dei nodi della rete: ad es. uno stesso elaboratore dovra' essere configurato diversamente a seconda del suo ruolo di "server" o di "client".

La proprieta' di confronto invece richiede l'intervento del modulo Matcher (v. sez. 5.2).
Durante la fase di risoluzione dei (sotto)problemi il motore inferenziale agisce essenzialmente assegnando valori a (istanziando) i parametri che li caratterizzano, rappresentati da variabili Prolog. Le variabili vengono inserite nella struttura ad albero in modo tale che ad ogni nodo genitore siano associate le variabili corrispondenti ai parametri decisionali dei figli, e ad ogni figlio siano associati i parametri che caratterizzano il singolo sottoproblema corrispondente. In questo modo si viene a creare un meccanismo per cui i nodi genitori "ereditano" dai figli i valori dei parametri.
I vari sottoproblemi non sono sempre indipendenti, ma possono influenzarsi reciprocamente. Il legame fra essi e' realizzato tramite condivisione di variabili. Un parametro a cui sia stato gia' assegnato un valore puo' intervenire in una regola che assegna il valore a un altro parametro. Le variabili associate ad un nodo rappresentano percio' due tipi distinti di parametri:

- parametri che caratterizzano la soluzione

- parametri che influenzano la soluzione

I parametri che i figli trasmettono ai genitori sono del primo tipo.
Per il secondo tipo di parametri si crea invece un meccanismo di eredita', opposto a quello precedentemente descritto, che procede questa volta nella direzione genitori--->figli.

5.2 Matcher

Questo modulo da la possibilita' all'utente, durante il percorrimento dell'albero,di verificare la significativita' delle decisioni prese dal sistema operando un confronto con i prodotti forniti dal mercato. I valori assunti dalle variabili che rappresentano le caratteristiche della rete verranno confrontati con i valori che i corrispondenti parametri assumono nel prodotto specifico e che sono contenuti nella base di conoscenza.

I nodi dell'albero, dal punto di vista del Matcher, sono suddivisi in due tipi: quelli dotati della proprieta' di confronto e quelli che non hanno associata tale proprieta'. I primi rappresentano i sottoproblemi principali, che implicano il dimensionamento di parti importanti, e spesso comportano un grande numero di decisioni. Giunto in questi nodi il motore inferenziale opera sempre dei confronti con le caratteristiche dei prodotti rappresentati nella base della conoscenza, per vedere se esistono prodotti che soddisfano le caratteristiche dettate dalle decisioni prese finora. Il secondo tipo di nodi rappresenta invece dei sottoproblemi meno critici ed il confronto con quanto esistente sul mercato viene effettuato solo quando e' l'utente a farne esplicita richiesta.
Come risultato finale saranno proposti uno o piu' prodotti commerciali che possono soddisfare tutti o solo alcuni dei requisiti utente: i parametri del modello costruito dal modulo Tree Traversal vengono confrontati in ordine decrescente di importanza e, nel caso solo alcuni requisiti siano soddisfatti, il Matcher propone i prodotti che soddisfano comunque i requisiti di maggiore importanza. Di fronte ad un prodotto che non soddisfa alcune caratteristiche del modello l'utente ha tre possibilita':

- accettare il prodotto in considerazione del fatto che i requisiti non soddisfatti sono in ogni caso i meno vincolanti

- rimettere in discussione le ultime scelte operate dal Tree Traversal e quindi richiedere al motore inferenziale di modificare il modello

- modificare alcuni requisiti iniziali obbligando il

motore inferenziale a ricostruire in parte od anche completamente il modello.

Tuttavia, il confronto dei parametri non sempre e' sufficiente. Infatti, le componenti che costituiscono la soluzione del problema possono appartenere a fornitori diversi e quindi possono creare problemi di incompatibilita'. Percio' ogni prodotto della base di conoscenza viene rappresentato non solo con le proprie caratteristiche ma anche con l'informazione relativa a tutti i prodotti(o famiglie di prodotti) con i quali e' compatibile. Prima di considerare un prodotto come una delle soluzioni, il Matcher controlla tale informazione ed elimina gli insiemi di prodotti non compatibili, anche se in teoria soddisfano tutti i requisiti. Questo controllo permette quindi di proporre solo le soluzioni effettivamente realizzabili.
Un esempio di risposta all'utente relativa ai requisiti della sez 4.2.1 e' in Fig. 7

6. Conclusioni

Questo articolo ha presentato ELAND, un sistema esperto in Prolog di supporto alle decisioni per la configurazione di una rete locale di personal computer. ELAND guida il progettista alla scelta dei prodotti piu' adatti al caso in esame. La decisione di ELAND e' condizionata sia da requisiti precisi, sia da fattori soggettivi non definibili in modo rigoroso, che il sistema provvede a precisare. La decisione finale dell'utente sara' poi basata sulle indicazioni del sistema e anche su considerazioni piu' "esterne", come le condizioni di vendita e di assistenza.

La strutturazione della conoscenza secondo un grafo AND/OR permette una decomposizione in moduli con interfacce ben definite e una limitazione precisa dello "scope" di ogni regola.
E' pertanto facile inserire nuova conoscenza sia orizzontale (n. di sottoproblemi) sia verticale (profondita' dei sottoproblemi). Inoltre la mancanza di struttura della rappresentazione della conoscenza relativa ai prodotti facilita il suo aggiornamento.
Il sistema e' stato realizzato in Prolog presso il Politecnico di Milano, utilizzando l'interprete C-prolog su Vax 780 e Prolog2 su PC IBM.

```
Soluzione globale numero 1 del problema " lan " .
Esiste una rete locale composta dai seguenti prodotti compatibili:
    systype   PCclone
    systype   PC-AT
    softtype   3+
    nettype   3COM

Viene ora determinato il costo della soluzione proposta.
Tale costo e' espresso in milioni di lire italiane      ed e' basato
sui prezzi di listino.

Calcolo del costo della connesssione degli elaboratori.
La rete 3COM ha i seguenti costi unitari:
  Costo base della rete:   0.26
  Costo per connessione:   1.5
  Costo per metro del mezzo trasmissivo:   0.01
     (comprende anche l'installazione)

  Costo totale della connessione:   21.26

Calcolo del costo delle stazioni di lavoro.
Il personal computer PCclone ha i seguenti costi unitari:
  Costo base del sistema:   1
  Costo espansione memoria a 512 KB:   0.2
  Costo drive:   0.4
  Costo mouse:   0.6

  Costo totale delle 12 workstation:   26.4

Calcolo del costo del/i server.
L'elaboratore PC-AT ha i seguenti costi unitari:
  Costo base del sistema:   6
  Costo espansione memoria a 1 MB:   0
  Costo drive:   0.5
  Il costo disco rigido e' stato scomposto in due valori:
  Costo dei primi 10 MB:   0
  Costo dei successivi 10 MB:   0.3
  Costo della generica stampante:   1
  Costo del generico protocollo di comunicazione:   2
  Costo del sistema di backup a nastro:   3.2

  Costo totale del/i server:   14.8

Si noti che il costo del software di rete e' compreso nel costo per
connessione.
Costo totale della soluzione proposta:   62.46  milioni.


Soluzione globale numero 2 del problema " lan " .
Esiste una rete locale composta dai seguenti prodotti compatibili:
    systype   PCclone
    systype   PC-AT
    softtype   ADV-NETWARE
    nettype   PC-NET

Viene ora determinato il costo della soluzione proposta.
Tale costo e' espresso in milioni di lire italiane 1987 ed e' basato
sui prezzi di listino.

Calcolo del costo della connesssione degli elaboratori.
La rete PC-NET ha i seguenti costi unitari:
  Costo base della rete:   0.08
  Costo per connessione:   1.2
  Costo per metro del mezzo trasmissivo:   0.01
     (comprende anche l'installazione)

  Costo totale della connessione:   17.18
OMISSIS
  Costo totale della soluzione proposta:   58.38  milioni.

  yes
  | ?-
```


BIBLIOGRAFIA

[1] Aiello L., Cecchi C., Sartini D., "Representation and use of MetaKnowledge", Proceedings of the IEEE, vol.70, n.10, 1986.

[2] Bracchi G., L'automazione del lavoro d'ufficio, ETAS, Milano, 1984.

[3] Bratko I., Prolog Programming for Artificial Intelligence, Addison-Wesley, Reading, MA, 1986.

[4] Ceri S., Tanca L., ELAND: an expert system for the configuration of local area network applications, IEEE Expert.

[5] Chorafas D., Designing and Implementing Local Area Networks, McGraw-Hill, New York, 1984.

[6] Clocksin W., Mellish C., Programming in Prolog, Springer-Verlag, Berlino, 1981.

[7] Datapro Research Corp., "Users rate their LANs", Data Communications, vol.15, n.7, 1986.

[8] Goldberg D., Moore R., Siegman J., "Microcomputer networks", Stanford University Academic Computing and Information Systems, aprile 1984.

[9] Guida G., Tasso C., Fum D., "Forward and Backward Reasoning in Automatic Abstracting", 9th COLIG Proceedings, North-Holland, Amsterdam, 5-10 luglio 1982.

[10] Harmon P., King D., Expert Systems / Artificial Intelligence in Business, John Wiley & Sons, New York, 1985.

[11] Hayes-Roth F., Waterman D., Lenat D., Building Expert Systems, Addison-Wesley, Reading, MA, 1983.

[12] Hayes-Roth F., "Rule-based Expert Systems", Communications of the ACM, vol.28, n.9, 1985.

[13] Hutchison , "Local Area Network Architectures", Addison Wesley Reading, MA, 1988

[14] Kowalski R., Logic for Problem Solving, Elsevier North-Holland, New York, 1979.

[15] Le Moli G., Telematica: architetture, protocolli e

servizi, ISEDI, Milano, 1983.

[16] Medelin F., Rota F., Zarattini A., Un sistema esperto per la configurazione di reti locali, Tesi di Laurea in Ingegneria Elettronica, Politecnico di Milano, 1987.

[17] Moore C., McAuliffe R., "An effective way to design and implement a local area network", Data Communications, vol.15, n.4, 1986.

[18] Stallings W., "Local Area Networks", ACM Computer Surveys, vol.16, n.1, 1984.

[19] Stallings W., "Master plan for LANs", Datamation, 15, Nov. 1986.

[20] Tanenbaum A., Computer Networks, Pretince-Hall, London, 1981.

Time Granularity in Logical Specifications

Edoardo Corsetti, Angelo Montanari, Elena Ratto

CISE - Tecnologie Innovative, SPA

Via Reggio Emilia, 39 - 20090 Segrate (Milano), Italy

Abstract

The paper extends the specification language TRIO to deal with time granularity. TRIO is a first-order logic formalism, augmented with temporal operators and a metric on time. To overcome its original temporal flatness, we first replace the notion of temporal domain by the one of temporal universe. Then we introduce multisorted temporal terms to relate the truth value of the subformulae of a specification to the different domains of the temporal universe. Finally we define the semantics and the executability of the extended language by translating it into TRIO. Such an ability of dealing with different time granularities within a single specification enhances both the expressive and the computational power of the specification language. With respect to the expressive power it introduces temporal modularity and abstraction. With respect to the computational power it supports local temporal reasoning at different granularity. It also allows one to alternate among different time granularities during the execution of a specification. Finally, it enables one to execute a specification at a time granularity coarser than the required one to cope with the complexity of temporal reasoning.

1 Introduction

A joint research project on the definition of an *executable formalism* for *specifying real-time systems* was started in 1987 by CISE and Politecnico di Milano[1]. During the first phase of the research we defined the specification language TRIO. It is a first-order logical formalism, augmented with temporal operators and a metric on time [Mor89], [MRRZ89], [GMM90]. It has a rigorous *model theoretic semantics* that provides a sound basis for the executability of specifications. It is performed by using the *proof method of semantic tableaux* [BPM83], [BeMa77], [MMW84], [Wol83]. This method has the advantage of being more natural than other proof theories [JRvH89]. For example, formulae do not need to be rewritten in conjunctive normal form as in the resolution-based theories, but can be used in whatever syntactic form. Moreover, it is equivalent to the natural deduction and the resolution proof methods with respect to decidability, but differently from them it supports constructive proofs. That is, it

[1]The research is funded by the Automatica Research Center (CRA) of the Electricity Board of Italy (ENEL) and, from June of 1989, partially by the National Research Council of Italy(CNR), within the National Project on Parallel Machines and Computer Science

proves the finite satisfiability of a formula by "constructing" a finite model of it. We also implemented a prototype TRIO-based specification environment that supports the activity of verifying and validating specifications [MMRR89][2].

In this paper we show how to embed time granularity in TRIO. There exists a widespread recognition of the relevance of the notion of granularity in formal languages. In particular, time granularity enhances both the expressive and the computational power of a specification formalism. With respect to the *expressive power* it allows one to maintain the specification of the dynamics of different processes that evolve according to different time constants as separate as possible [CMR90b]. It also allows one to model the dynamics of a process with respect to different time scales [CMR90a]. Finally, it increases both the temporal distinctions that the language can make and the distinctions that it can leave unspecified. This means that considering two "events" as simultaneous or temporally distinct, or two "properties" as temporally overlapped or disjoint depends on the time granularity we refer to. With respect to the *computational power* it supports different grains of temporal reasoning. In such a way it allows one to deal with incompleteness and uncertainty of information [All83], [FGMT89]. It also allows one to alternate among different time granularities during the execution of a specification in order to perform each reasoning step at a time granularity as coarse as possible. Finally, it allows one to execute a specification at a time granularity coarser than the required one to cope with the complexity of temporal reasoning. Such a simplification speeds up the execution, but it implies a relaxation of the precision of the results (partial validation). The ratio between the time granularities provides a measurement of their approximation.
Nevertheless, there is a lack of a systematic framework for time granularity. The main references are the paper of Hobbs on the general concept of granularity [Hob85] and the works of Plaisted [Pla81], Giunchiglia and Walsh [GiWa89] on abstract theorem proving. Hobbs defines a concept of granularity that supports the construction of simple theories out of more complex ones. He formally introduces the basic notions of relevant predicate set, indistinguishability relations, simplification, idealization and articulation. With respect to time granularity, he only sketches out a rather restrictive mapping of continuous time into discrete times using the situation calculus formalism. The notions defined by Hobbs are extended and refined in [IIaAl87] and [GrMC89]. In particular, Greer and McCalla identify two orthogonal dimensions along which granularity may be interpreted, namely abstraction and aggregation. However, both reserve little or no attention to temporal granularity. Conversely, a set-theoretic formalization of temporal granularity is provided by Clifford and Rao, but they do not attempt to relate the truth value of assertions to temporal granularity [ClRa88]. Finally, Shoham gives a significant categorization of assertions based on their temporal properties, but he does not take into account any notion of temporal granularity [Sho88].

TRIO does not take into account time granularity. It constrains the specifier to describe the whole system with respect to the finest time scale. It also prevents the user to incrementally build up a specification moving from coarser time scales to finer ones through successive refinements. The rationale of the introduction of time granularity in real-time systems specifications together with the identification of the main representational requirements it imposes are presented in detail in [CMR91]. A preliminary definition of time granularity in TRIO is given in [MRCM91]. We first introduced the weakest semantics of time granularity, then we identified a number of possible specializations of such a semantics taking into account both common-sense and domain-specific knowledge. They resulted in a taxonomic classification of predicates that makes the formalism more expressive and easier to use.

In this paper, we extensively illustrate the formalization of time granularity in TRIO. Each proposed extension to the basic formalism is presented in detail and contrasted with alternative possibilities. First of all, we replace the original notion of temporal domain by the notion of temporal universe to overcome the temporal flatness of TRIO. Then we introduce multisorted temporal terms to qualify predications with respect to time granularity. Finally, we define the semantics and the executability of the extended language by translating it into TRIO.

[2]The close relationships between logic specification and logic programming is widely recognized, e.g. [Kow85]. First of all, there is no syntactic distinction between complete specification and program. Moreover, the same mechanism that is used to execute logic programs (automated deduction) can also be used to execute logic specifications. The only difference between a complete specification and a program is one of efficiency: a program is more efficient than a specification. Logic also provides formal methods to prove the partial correctness and/or the completeness of logic programs with respect to logic specifications. Finally, logic supports transformations of logic programs as well as derivations of logic programs from logic specifications.

2 The Basic Logical Language TRIO

TRIO is a typed first order language, augmented with temporal operators and a metric on time. According to the Linear Temporal Logic approach, TRIO is endowed with a totally ordered temporal domain and its temporal operators allow one to talk about the truth and the falsity of predications at time instants different from the current one that is left implicit [Pnu81], [ReUr71]. The principal temporal operators of Temporal Logic expressing necessity and possibility in time can be easily derived from these TRIO basic temporal operators. However, differently from Temporal Logic TRIO allows the specifier to explicitly quantify over time. Moreover, it is endowed with a metric on time that enables the specifier to constrain the width of time intervals according to both discrete and dense structures of time[3]. Finally, TRIO differs from other specification formalisms for real-time systems based on logic as [KVdR83]. [KdR85], [AuKe86] and [JaMo86], because it is much closer to first order predicate calculus, thus retaining its simplicity and expressiveness.

2.1 TRIO Syntax

TRIO syntax looks like first order logic one. Its alphabet consists of the usual symbols of a first order alphabet plus the temporal operators $Futr$ and $Past$. Moreover, the variables and the predicates of the language are partitioned in time dependent and time independent ones[4]. The terms of the language are the first order terms, while TRIO formulae are the first order ones plus the formulae obtained by applying the temporal operators. That is, if F is a TRIO formula and t is a temporal term, then $Futr(F,t)$ and $Past(F,t)$ are TRIO formulae too.
The intuitive semantics of these formulae is that they hold at the current instant if and only if F holds at a temporal distance t in the future and in the past, respectively.
From TRIO temporal operators we can derive a number of other ones, including:

$$
\begin{aligned}
AlwFutr(F) &\equiv \forall t(t > 0 \rightarrow Futr(F,t)) \\
SomFutr(F) &\equiv \neg AlwFutr(\neg F) \\
Lasts_{e,e}(F,t) &\equiv \forall t'(0 < t' < t \rightarrow Futr(F,t')) \\
Lasted_{e,e}(F,t) &\equiv \forall t'(0 < t' < t \rightarrow Past(F,t')) \\
Sometimes(F) &\equiv SomPast(F) \vee F \vee SomFutr(F)
\end{aligned}
$$

[3]On the contrary, the *next* operator of Temporal Logics is obviously defined for discrete time only.

[4]Time dependent variables identify quantities that may change their value in time, while time independent ones do not change it. Time dependent predicates may or may not hold at a given time instant, while time independent ones do not change their truth value in time.

$Always(F) \equiv AlwPast(F) \wedge F \wedge AlwFutr(F)$
$Until(F_1, F_2) \equiv \exists t\ (Futr(F_2, t) \wedge Lasts_{e,e}(F_1, t))$

where F, F_1 and F_2 are TRIO formulae and t and t' are TRIO terms denoting temporal distances. $AlwFutr(F)$ says that F will be true in any future instant; $SomFutr(F)$ says that F will be true in at least one instant in the future; $Lasts_{e,e}(F, t)$ says that F will be true in each instant in the future at a temporal distance lower than t from the current one; $Lasted_{e,e}(F, t)$ says that F was true in each instant in the past at a temporal distance lower than t from the current one; $Sometimes(F)$ says that there exists at least one instant over which F holds; $Always(F)$ says that F holds in every instant of the temporal domain; $Until(F_1, F_2)$ says that F_2 will be true in the future and till then F_1 will be true.

A *specification* Σ of a real-time system is a classically and temporally closed TRIO formula. Let us consider, as an example, the TRIO specification of a communication channel that delays the output of a time interval of width t with respect to the input and that neither produces nor loses messages:

$$Always(\forall msg(in(msg) \leftrightarrow Futr(out(msg), t)))$$

Beside the notion of specification, we introduce the one of relevant history H.
Formally, a *history* H is a TRIO formula of the form:

$$Sometimes(\wedge_\imath Futr(F_\imath, t_\imath) \wedge \wedge_\jmath Past(P_\jmath, t_\jmath))$$

where, for each $\imath$ and $\jmath$, $F_\imath$ and $P_\jmath$ are atomic formulae of the type: (a) time dependent predicates applied to ground terms; (b) time independent predicates of the form $y = t$, where y is a time dependent variable and t is a ground term; $t_\imath$ and $t_\jmath$ are ground terms.
A history describes a possible temporal evolution of the specified system by constraining the temporal relations between predications that represent occurring events or system states.

2.2 TRIO Semantics and Executability

A model theoretic semantics is provided for TRIO. It is founded on the concept of temporal structure, over which the syntactical components of TRIO are interpreted. Such a structure S is defined as 4-tuple (I, T, G, L), where G is the time independent part of TRIO interpretation that evaluates time independent variables, functions and predicates; L is the time dependent part of TRIO interpretation that, for each instant belonging to the temporal domain T, evaluates time dependent variables and predicates; I is the set of sorts over which TRIO terms are interpreted.

A temporal structure defines a set of interpretations that differ from each other for the time instant of the temporal domain they assign to the implicit current time. The TRIO formal semantics allows the definition of the notions of temporal satisfiability and temporal validity of formulae with respect to a temporal structure. A TRIO formula is said to be *temporally satisfiable* with respect to a temporal structure if it evaluates to true in at least one instant of the temporal domain. In such a case, we say that the temporal structure provides a *model* for the formula. A formula is said to be *temporally valid* with respect to a temporal structure if it is true in every instant of the temporal domain. Finally, a TRIO formula is said to be *time invariant* if it is either temporally valid or unsatisfiable with respect to every suitable temporal structure[5].

The TRIO's rigorous model theoretic semantics provides a sound basis for the executability of TRIO specifications. It is performed by using the proof method of semantic tableaux. By

[5]It is possible to prove that a specification is a time invariant formula.

applying such a method we can perform a number of formal proofs about TRIO specifications, TRIO histories and the relations between them.
First of all, the *consistency* of a TRIO specification Σ or history H is proved by showing that they have a model. The relation between a specification Σ and a relevant history H can be also expressed in a model theoretic way. We say that a given history H is *compatible* with a specification Σ if the logical conjunction of them $H_\Sigma = \Sigma \wedge H$ has a model, otherwise it is *incompatible*. In particular, if such a model is also a model of the history taken in isolation then we say that the history is *complete* with respect to the specification. Otherwise it is compatible, but *incomplete*. In the case of incomplete histories, the process of model construction implements a sort of system simulation as *history generation*. That is, it completes a given incomplete history according to the specification requirements. In general, such a process produces a set of complete histories, each one including the initial incomplete history that provides the system simulation process with a sort of initial conditions. Finally, the proof method of tableaux used in its full generality allows the specifier to formally prove any property of the system that can be deduced from its specification. To prove that a system has a given property, in fact, it is sufficient to show that the system specification Σ logically implies the TRIO formula P expressing the property, i.e. that the TRIO formula $\Sigma \rightarrow P$ is valid. The method of tableaux proves the validity of this formula by showing that the logical conjunction of Σ and the negation of P has no model, i.e. that the TRIO formula $\Sigma \wedge \neg P$ is unsatisfiable.

TRIO executability provides an implementation of the general concepts of verification and validation of real-time system specifications [CMR91]. *Verifying a specification* means proving its internal *consistency*. It consists of showing that the specification does not hold contradictions. *Validating a specification* means proving its *adequacy*. It consists of showing the completeness and the soundness of the specification with respect to the specifier expectations. We say that a specification is *complete* if it recognizes every history and satisfies every property which describe the system behaviour expected by the specifier. We say that a specification is *sound* if every history it generates is accepted by the specifier.

3 A Logic Treatment of Time Granularity

3.1 Temporal Universe and Multisorted Temporal Terms

For each specification a totally ordered set of disjoint temporal domains is given. These are denoted as $T_1, T_2, .., T_n, ..$ and constitute the *temporal universe* of the specification.

Figure 1: Differently Grained Temporal Domains.

For each i, T_{i+1} is said of a finer granularity than T_i. Each T_i is discrete with the possible exception of the finest temporal domain that may be dense. The temporal universe includes at most one dense temporal domain because each dense domain is already at the finest level

of granularity. Figure 1 pictorially represents two temporal domains T_i and T_j, with T_i coarser than T_j.

For each pair of *discrete* temporal domains T_i and T_{i+1} an elementary correspondence rule (*decomposition function*) is defined that maps time instants of T_i into time intervals of T_{i+1}[6]:

$$\exists \psi_{i,i+1} : T_i \mapsto 2^{T_{i+1}} (\forall t_i \in T_i, t_{i+1} \in T_{i+1}$$
$$(first_{i+1}(\psi_{i,i+1}(t_i)) \leq t_{i+1} \leq last_{i+1}(\psi_{i,i+1}(t_i)) \Rightarrow t_{i+1} \in \psi_{i,i+1}(t_i)))$$

$\psi_{i,i+1}$ is defined for each element of T_i (*totality*):

$$\forall t \in T_i \; \exists I \in 2^{T_{i+1}} \; (\psi_{i,i+1}(t) = I)$$

and maps contiguous elements into contiguous, disjoint intervals preserving the ordering of the domains (*contiguity*):

$$\forall t \in T_i \; (succ_{i+1}(last_{i+1}(\psi_{i,i+1}(t))) = first_{i+1}(\psi_{i,i+1}(succ_i(t))))$$

Moreover, the union set of the time intervals of T_{i+1} belonging to its range is equal to T_{i+1} (*coverage*):

$$\bigcup_{t \in T_i} \psi_{i,i+1}(t) = T_{i+1}$$

Finally, the length of the intervals is usually constant (*homogeneity*):

$$\forall t \in T_i (\sharp \psi_{i,i+1}(t) = \Delta_{i,i+1})$$

where $\Delta_{i,i+1}$ is a constant that defines the *conversion factor* between T_i and T_{i+1}. It provides a relative measurement of the granularity of T_i and T_{i+1} with respect to each other.
A general mapping between T_i and T_j, with T_i coarser than T_j, can be obtained by a suitable composition of a number of elementary decomposition functions. For each pair T_i, T_j of domains, with T_i coarser than T_j, the decomposition function $\psi_{i,j}$ is recursively defined as follows:

$$if(j = i+1) then(\forall t \in T_i(\psi_{i,j}(t) = \psi_{i,i+1}(t))) else(\forall t \in T_i(\psi_{i,j} = \bigcup_{t_{j-1} \in \psi_{i,j-1}(t)} \psi_{j-1,j}(t_{j-1})))$$

It is easy to show that the functions $\psi_{i,j}$ also have the properties of totality, contiguity, homogeneity and coverage.

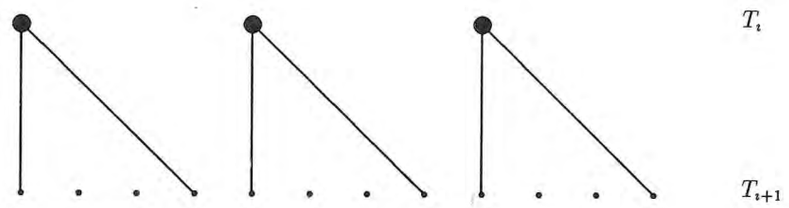


Figure 2: A Possible Mapping between domains.

[6]In the next mathematical expressions we use the functions *succ*, *first* and *last* borrowed from the theory of natural numbers. For each domain T_i, $succ_i(t)$ denotes the successor of t in T_i, $first_i(t_1, .., t_n)$ denotes the least element of the set $\{ t_1, .., t_n \}$ and last, $last_i(t_1, .., t_n)$ denotes the greatest element of the set $\{ t_1, .., t_n \}$.

In general, there are several ways to define the mappings $\psi_{i,i+1}$ each one satisfying the required properties. According to the intended meaning of the mappings as decomposition functions, each time instant of T_i is mapped into the set of time instants of T_{i+1} that compose it. Nevertheless, we are faced by a number of alternative possibilities in settling the reference time instant of each domain of the temporal universe. Choosing the one or the other is merely a matter of convention, but it determines the actual form of the mappings $\psi_{i,i+1}$.
Figure 2 shows a possible mapping $\psi_{i,i+1}$ between two temporal domains T_i and T_{i+1} whose $\Delta_{i,i+1}$ is equal to 4.

Then, we allow the specifier to associate a different temporal domain with the different subformulae of a specification. The resulting formula consists of the composition of a set of differently grained subformulae. Formally, each subformula is qualified with respect to temporal granularity by using *multisorted temporal terms*. That is, a sort is assigned to each of its temporal terms (variables and functions) that denotes the temporal domains over which they take value. Notationally, we allow the user to label the subformula with the identifier of a temporal domain. The scope of the labeling is marked by square brackets.

Let's define $TRIO \neq$ the resulting language. In the simplest case, a $TRIO \neq$ formula consists of the logical composition of a number of temporally closed TRIO formulae, each one referring to a different temporal domain.
With respect to the temporal domains Y of the years, M_o of the months and D of the days, an example of a well formed $TRIO \neq$ formula is:

$$man(x) \rightarrow [Always(work(x))]_{M_o} \wedge [Always(eat(x))]_D$$

$TRIO \neq$ also supports more complex combination of differently grained subformulae. For instance, it allows the specifier to algebraically add differently grained temporal distance. As an example, let' s consider the sentence: "In twenty seconds will be passed five minutes from the occurrence of the fault". Given the domains M_i of minutes and S of seconds, it is represented by the $TRIO \neq$ formula:

$$[Futr([Past(fault, 5)]_{M_i}, 20)]_S$$

Moreover, $TRIO \neq$ allows the nesting of temporally quantified differently grained subformulae. As an example, let's consider the sentence: "There exist some days during which shops remain open every hour". Given the domains D of days and H of hours, it is represented in $TRIO \neq$ by the formula:

$$shop(x) \rightarrow [Sometimes([Always(open(x))]_{T_{H(D)}})]_{T_D}$$

Notice that the domain of the inner quantified variable is given by the decomposition of the elements of the domain of the outer one.

3.2 The Semantics of Time Granularity

$TRIO \neq$ semantics are defined by downward projecting its formulae into TRIO formulae referring to the finest domain of the temporal universe. The projection of a $TRIO \neq$ formula that holds at a time instant $t \in T_i$ on T_j, with T_j finer than T_i, is given in a compositional way on the basis of the projection of its literals.
Given a literal L its projection $\tau_{i,j}(L)$ is defined on the bases of the assumption that if L holds at a given $t \in T_i$ then it holds over a bounded sequence of intervals of T_j. Let us denote such a sequence with $INTSEC_{i,j}(L)$. It is depicted in Figure 3.

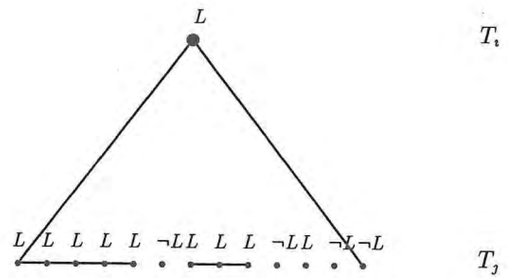


Figure 3: The Projection of the Literal L.

$INTSEC_{i,j}(L)$ is formally defined as follows:

$$INTSEC_{i,j}(L) \stackrel{\text{def}}{=} [\exists n(n > 0 \wedge \exists < t_1, t_2, ..., t_{2n} > (0 \leq t_1 \wedge t_{2n} < \Delta_{i,j} \wedge \forall k((1 \leq k < 2n \rightarrow t_k < t_{k+1}) \wedge (1 \leq k \leq n \rightarrow Futr(Lasts_{i,i}(L, t_{2k} - t_{2k-1}), t_{2k-1})) \wedge (1 \leq k \leq n-1 \rightarrow Futr(Lasts_{e,e}(\neg L, t_{2k+1} - t_{2k}), t_{2k}))) \wedge Lasts_{i,e}(\neg L, t_1) \wedge Futr(Lasts_{e,e}(\neg L, \Delta_{i,j} - t_{2n}), t_{2n})))]_{T_j}$$

where $Lasts_{i,e}(L,t)$ and $Lasts_{i,i}(L,t)$ are derived temporal operators[7].
$INTSEC_{i,j}(L)$ provides the *weakest semantics* that can be attached to L in the finer domain T_j. In fact, it is semantically equivalent to the following one:

$$INTSEC_{i,j}(L) \stackrel{\text{def}}{=} [\exists t(0 \leq t < \Delta_{i,j} \wedge Futr(L,t))]_{T_j}$$

Most often it is too weak so that user specializations of $INTSEC_{i,j}(L)$ are needed. They can be obtained by constraining the number and/or the value of the time instants of the sequence $< t_1, t_2, ..., t_{2n} >$. The resulting specialized formula always implies $INTSEC_{i,j}(L)$ [MRCM91].

The localization of the sequence in T_j depends on the synchronous or asynchronous interpretation of the formula. Let us consider, as an example, the assertion: "the answer must arrive in one month" with respect to the current time. The temporal qualifier "in one month" is susceptible of a twofold synchronous and asynchronous semantic interpretation. In the first case, it is interpreted as "within the next month"; in the other one as "within thirty days starting from today". That is, the *synchronous* interpretation constrains the "arrive of the answer" to occur at one of the thirty days belonging to the month following the current one, while the *asynchronous* one constrains it to occur at one of the thirty days following the current day. If we denote with t_1 the current day, with t_2 the day at which the "answer arrives" and with d_1 the number of days that intervene between the current day and the last day of the current month, then we can differentiate synchronous and asynchronous interpretations on the basis of

[7]They are defined as follows:

$$Lasts_{i,e}(P,t) \stackrel{\text{def}}{=} t > 0 \rightarrow (P \wedge Lasts_{e,e}(P,t))$$
$$Lasts_{i,i}(P,t) \stackrel{\text{def}}{=} P \wedge Lasts_{e,e}(P,t) \wedge Futr(P,t)$$

the different constraint that they impose to the temporal interval $d_2 = t_2 - t_1$. According to the first one $d_1 < d_2 \leq d_1 + 30$ (Figure 4):

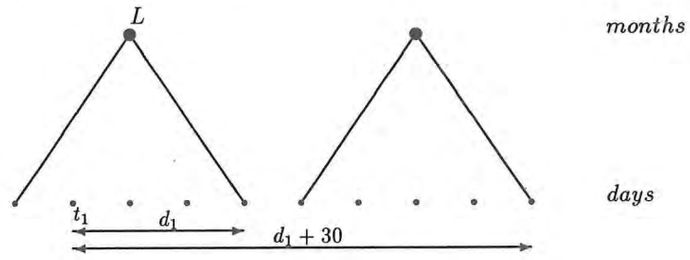


Figure 4: Synchronous Interpretation.

while the second one requires that $0 < d_2 \leq 30$ (Figure 5):

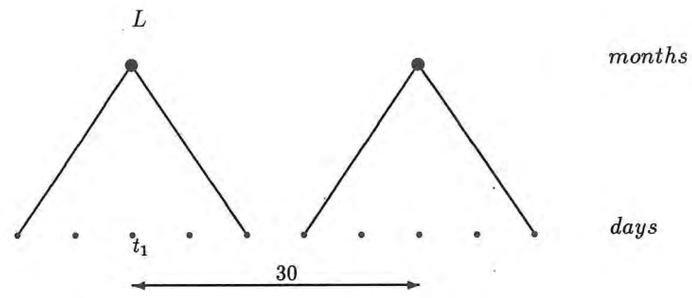


Figure 5: Asynchronous Interpretation.

As a general rule, when switching from a given domain to a finer one synchronous and asynchronous interpretations give rise to different temporal projections of literals that differ from each other in the bounds they impose to the sequence of intervals of the finer domain over which the literals hold. In this paper we only deal with the *synchronous case.*
First of all we assume that there are no translations of the domain with respect to each other, i.e. for each pair of domain T_i, T_j, with T_i coarser than T_j, the current instant of the finer one i_j belongs the temporal decomposition of the current instant of the coarser one i_i. Formally:

$$i_j \in \psi_{i,j}(i_i)$$

Then we distinguish two alternative modalities of defining the downward projection $\tau_{i,j}(L)$ of a literal L according to the synchronous interpretation:

- by introducing a *synchronization* predicate $SYNC_{i,j}$ for which the following axiom is stated a priori[8]

[8]The temporal operator $NextTime$ is defined in the following way:

$$NextTime(A,t) \equiv Futr(A,t) \wedge Lasts_{e,e}(\neg A, t)$$

$$Sometimes(SYNC_{i,j}) \wedge Always(SYNC_{i,j} \leftrightarrow NextTime(SYNC_{i,j}, \Delta_{i,j})$$

together with the truth of $SYNC_{i,j}$ at the first time instant of a time interval of T_j belonging to $\psi_{i,j}(T_i)$[9].

From the given axiom we can derive the truth of $SYNC_{i,j}$ at the first time instant of each time interval of T_j belonging to $\psi_{i,j}(T_i)$. In such a case, $\tau_{i,j}(L)$ is defined as follows:

$$\tau_{i,j}(L) \stackrel{\text{def}}{=} [\exists t(0 \leq t < \Delta_{i,j} \wedge Past(SYNC_{i,j}, t) \wedge Past(INTSEC_{i,j}(L), t))]_{T_j}$$

It is graphically depicted in Figure 6.

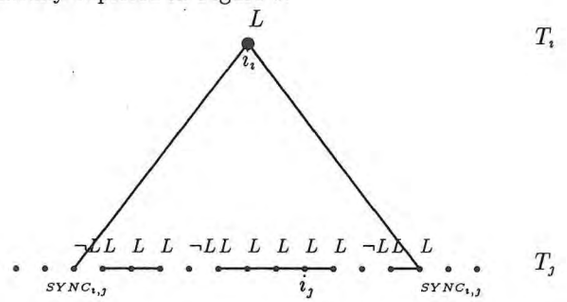


Figure 6. Downward Projection using the predicate $SYNC_{i,j}$.

- by making the projection parametric with respect to the current time instant i_j of the finer domain T_j using a parameter k that denotes the distance between i_j and the end of the projection interval at which it belongs.

  In such a case, $\tau_{i,j}(L)$ is defined as follows:

$$\tau_{i,j}(L) \stackrel{\text{def}}{=} [\exists m, n, k(m + n > 0 \wedge 0 \leq k < \Delta_{i,j} \wedge$$
$$PSEQUENCE_{i,j}(L, m, k) \wedge FSEQUENCE_{i,j}(L, n, k))]_{T_j}$$

  where:

$$PSEQUENCE_{i,j}(L, m, k) \stackrel{\text{def}}{=} [\exists t_1, .., t_{2m}$$
$$(0 < t_1 \wedge t_{2m} \leq \Delta_{i,j} - k - 1 \wedge \forall i((1 \leq i < 2m \rightarrow t_i < t_{i+1}) \wedge$$
$$\wedge(1 \leq i \leq m \rightarrow Past(Lasted_{i,i}(L, t_{2i} - t_{2i-1}), t_{2i-1})) \wedge$$
$$\wedge(1 \leq i \leq m - 1 \rightarrow Past(Lasted_{e,e}(\neg L, t_{2i+1} - t_{2i}), t_{2i}))) \wedge$$
$$\wedge Past(Lasted_{e,e}(\neg L, t_1), t_1) \wedge Past(Lasted_{e,i}(\neg L, \Delta_{i,j} - k - 1 - t_{2m}), t_{2m}))]_{T_j}$$

  and:

[9]Notice that the choice of a different time instant of the projection interval as the instant of synchronization is in principle admissible, but it involves the redefinition of the actual form of $INTSEC_{i,j}$.

$$FSEQUENCE_{i,j}(L, m, k) \stackrel{\text{def}}{=} [\exists t_1, .., t_{2n}$$
$$(0 \leq t_1 \wedge t_{2n} \leq k \wedge \forall j((1 \leq j < 2n \rightarrow t_j < t_{j+1}) \wedge$$
$$\wedge(1 \leq j \leq n \rightarrow Futr(Lasts_{i,i}(A, t_{2j} - t_{2j-1}), t_{2j-1})) \wedge$$
$$\wedge(1 \leq j \leq n - 1 \rightarrow Futr(Lasts_{e,e}(\neg L, t_{2j+1} - t_{2j}), t_{2j}))) \wedge$$
$$\wedge Lasts_{i,e}(\neg L, t_1) \wedge Futr(Lasts_{e,i}(\neg L, k - t_{2n}), t_{2n}))]_{T_j}$$

where $Lasts_{e,i}(L, t)$ is a derived temporal operator[10].

$INTSEC_{i,j}$ can be seen as the conjunction of $PSEQUENCE_{i,j}$ and $FSEQUENCE_{i,j}$ as shown in Figure 7.

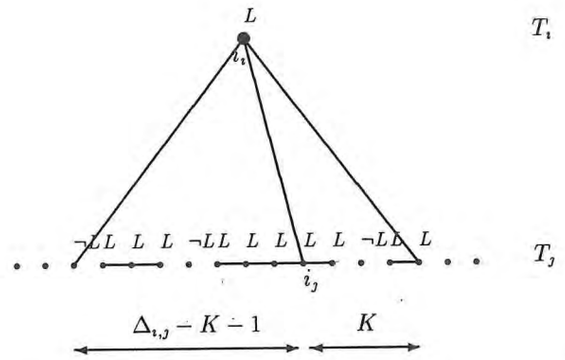


Figure 7. Downward Projection using the parameter K.

The semantic interpretation $S^{(GL)}_{i_i,i_j}(k)$ of the parameter k precisely states the relationship between i_i and i_j; it is defined as follows:

$$S^{(GL)}_{i_i,i_j}(k) \stackrel{\text{def}}{=} last(\psi_{i,j}(i_i)) - i_j$$

It is easy to show that the two modalities of "implementing" downward projection are equivalent.

Before defining the downward projection of a generic $TRIO \neq$ formula, notice that the projection of $TRIO \neq$ literals as defined by $INTSEC_{i,j}$ involves a *weak* interpretation of negation [Sho88]. It states that if $\neg A$ holds at a time instant t of T_i then there exists a sequence of time instants of T_j belonging to $\psi_{i,j}(t)$ over which $\neg A$ holds.

But we might have chosen a strong interpretation of negation, that is, if $\neg A$ holds at time instant t of T_i then it holds over the whole interval $\psi_{i,j}$. In such a case, the previous rule should have been applied to the atomic formulae and the downward projection of the negation given in a compositional way:

$$\tau_{i,j}(\neg A) \stackrel{\text{def}}{=} \neg\tau_{i,j}(A)$$

[10]It is defined as follows:

$$Lasts_{e,i}(P, t) \stackrel{\text{def}}{=} t > 0 \rightarrow (Lasts_{e,e}(P, t) \wedge Futr(P, t))$$

The weak interpretation of negation is depicted in Figure 8:

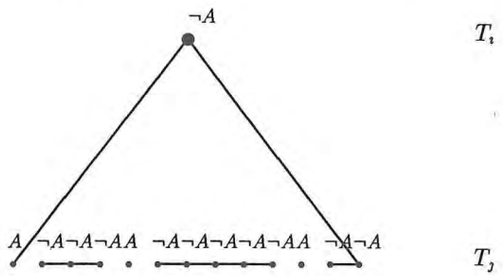


Figure 8: Weak Interpretation of Negation.

While the strong interpretation of negation is depicted in Figure 9:

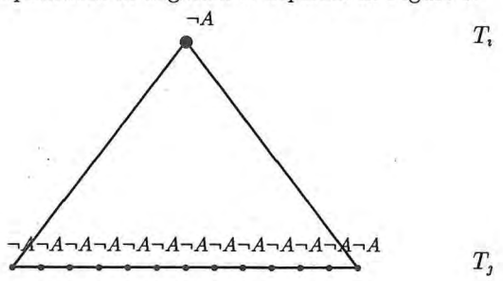


Figure 9: Strong Interpretation of Negation.

We choose the weak interpretation of negation because the strong one is a particular case of it as stated in the following expression:

$$\neg INTSEC_{i,j}(A) \equiv [Lasts_{i,i}(\neg A, \Delta_{i,j} - 1)]_{T2}$$

where:

$$Lasts_{i,i}(\neg A, \Delta_{i,j} - 1)]_{T2} \equiv$$
$$[\exists n(n > 0 \wedge \exists < t_1, t_2, ..., t_{2n} >$$
$$(0 \leq t_1 \wedge t_{2n} < \Delta_{i,j} \wedge \forall k((1 \leq k < 2n \rightarrow t_k < t_{k+1}) \wedge$$
$$\wedge (1 \leq k \leq n \rightarrow Futr(Lasts_{i,i}(L, t_{2k} - t_{2k-1}), t_{2k-1})) \wedge$$
$$\wedge (1 \leq k \leq n-1 \rightarrow Futr(Lasts_{e,e}(\neg L, t_{2k+1} - t_{2k}), t_{2k}))) \wedge$$
$$\wedge Lasts_{i,e}(\neg L, t_1) \wedge \wedge Futr(Lasts_{e,e}(\neg L, \Delta_{i,j} - t_{2n}), t_{2n})) \wedge$$
$$\wedge (n = 1 \wedge t_1 = 0 \wedge t_2 = \Delta_{i,j} - 1))]_{T_j}$$

The downward projection of a formula F is defined in an inductive way on the basis of the downward projection rule for literals as follows:

- $\tau_{i,j}(Futr(A,t)) \stackrel{\text{def}}{=} Futr(\tau_{i,j}(A), t * \Delta_{i,j})$
- $\tau_{i,j}(Past(A,t)) \stackrel{\text{def}}{=} Past(\tau_{i,j}(A), t * \Delta_{i,j})$
- $\tau_{i,j}(A \wedge B) \stackrel{\text{def}}{=} \tau_{i,j}(A) \wedge \tau_{i,j}(B)$
- $\tau_{i,j}(A \vee B) \stackrel{\text{def}}{=} \tau_{i,j}(A) \vee \tau_{i,j}(B)$
- $\tau_{i,j}(\forall x A) \stackrel{\text{def}}{=} \forall x \tau_{i,j}(A)$

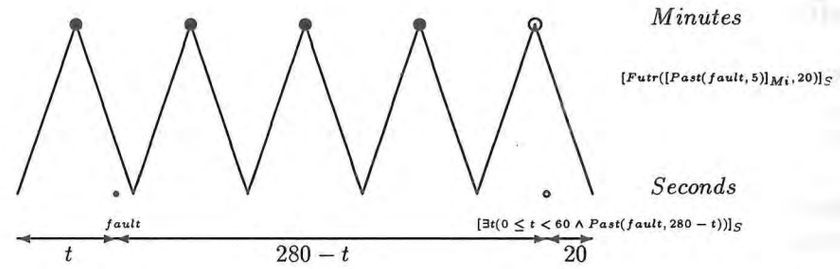


Figure 10: The visualisation of the "fault" example.

As an example of interpretation of a TRIO$_{\neq}$ formula let's reconsider the $TRIO \neq$ formula for the "fault" example (Figure 10):

$$[Futr([Past(fault, 5)]_{Mi}, 20)]_S$$

The interpretation starts with the selection of the subformula which refers to the coarsest domain. In the example, the subformula: $[Past(fault, 5)]_{Mi}$ is isolated.

Then it is projected on the domain of *Seconds*. The resulting formula is:

$$[Futr(Past(\tau_{Mi,S}(fault), 5 * 60), 20)]_S$$

It is a TRIO formula whose temporal terms refer to domain of *Seconds*. That is obviously equivalent to the formula:

$$[Futr(Past(\tau_{Mi,S}(fault), 300), 20)]_S$$

which can be farther simplified according to the algebra of temporal distances:

$$[Past(\tau_{Mi,S}(fault), 280)]_S$$

Because of $\tau_{Mi,S}(fault) \equiv \exists t(0 \leq t < 60 \wedge Futr(fault, t))$ the formula becomes:

$$[Past(\exists t(0 \leq t < 60 \wedge Futr(fault, t)), 280)]_S$$

which is equivalent to:

$$[\exists t(0 \leq t < 60 \wedge Past(fault, 280 - t))]_S$$

4 Conclusion

In the paper we show how to embed time granularity in TRIO. First we introduce the notion of temporal universe of a specification and provide the syntactic extensions to TRIO that allow us to represent time granularity, then we formally define the semantics of the extended language. We confine ourselves to the synchronous case. We are also investigating the asynchronous one which presents more technical intricacies. More precisely, we are defining the modifications in the structure of the temporal universe and in the semantic interpretation of $SYNC$ and k that it requires.

References

[Allen,83] *Maintaining Knowledge about Temporal Intervals*, Allen, J., Communications of the ACM, 26, 11, 1983.

[AuKe86] *RT-ASLAN: A Specification Language for Real-Time Systems*, Auernheimer, B., Kemmerer, R., IEEE TSE, vol. SE-12, no.9, September 1986.

[BeMa77] *A Course in Mathematical Logic*, Bell, J., Machover, M., North-Holland, 1977.

[BPM83] *The Temporal Logic of Branching Time*, Ben-Ari, M., Pnueli, A., Manna, Z., Acta Informatica, 1983.

[ClRa88] *A Simple, General Structure for Temporal Domain*, Clifford, J., Rao, A., in *Temporal Aspects in Information Systems*, Rolland, C., Bodart, F., and Leonard, M. (Editors), Elsevier Science Publishers B.V. (North-Holland), IFIP 1988.

[CMR90a] *A Methodology for an Incremental, Logical Specification of Real-Time Systems*, Corsetti, E., Montanari, A., Ratto, E., Proc. 2nd IEEE Euromicro Workshop on Real-Time Systems, Horsholm, Denmark, May 1990.

[CMR90b] *A Methodology for Real-Time System Specifications based on Knowledge Representation*, Corsetti, E., Montanari, A., Ratto, E., Proc. Computational Intelligence 90, Milano, Italy, September 1990.

[CMR91] *Dealing with Different Time Granularities in Formal Specification of Real-Time Systems*, Corsetti, E., Montanari, A., Ratto, E., REAL-TIME SYSTEMS, Vol. III, Issue 2, June 1991.

[FGMT,89] *Using Levels and Viewpoints in Text Representation*, Fum, D., Guida, G., Montanari, A., Tasso, C., AIICSR, Strbske Pleso, Czechoslovakia, 1989.

[GiWa89] *Abstract Theorem Proving*, Giunchiglia, F., Walsh, T., Proc. 11th IJCAI, Detroit, USA, 1989.

[GMM90] *TRIO, a Logic Language for Executable Specifications of Real-Time Systems*, Ghezzi, C., Mandrioli, D., Morzenti, A., The Journal of Systems and Software, June-July, 1990.

[GrMC89] *A Computational Framework for Granularity and its Application to Educational Diagnosis*, Greer, J., McCalla, G., Proc. 11th IJCAI, Detroit, USA 1989.

[HaAl87] *Short Time Periods*, Hayes, P. J., Allen, J. F., Proc. 10th IJCAI, Milano, 1987.

[Hob85] *Granularity*, Hobbs, J. R., Proc. 9th IJCAI, Los Angeles, 1985.

[JaMo86] *Safety Analysis of Timing Properties of Real-Time Systems*, Jahanian, F., Mok, A.,K., IEEE TSE, vol. SE-12, no.9, September 1986.

[JRvH89] *Logic-Based Knowledge Representation*, Jackson, P., Reichgelt, H., van Harmelen, F., The MIT Press, 1989.

[KdR85] *Examples of Real-Time Temporal Logic Specification*, Koymans, R., de Roever, W. P., in LNCS 207, Springer Verlag, 1985.

[Kow85] *The relation between logic programming and logic specification*, Kowalski, R., in Mathematical Logic and Programming Languages, Hoare, C.A.R. and Shepherdson, J.C. (Eds.), Prentice/Hall International, 1985.

[KVdR83] *Real-Time Programming and Asynchronous Message Passing*, Koymans, R., Vytopil, de Roever, W. P., Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada, 1983.

[MRCM91] *Embedding Time Granularity in Logical Specification of Real-Time Systems*, Montanari, A., Ratto, E., Corsetti, E., Morzenti, A., Proc. 3rd IEEE Euromicro Workshop on Real-Time Systems, Paris, France, June 1991.

[MMRR89] *Un Ambiente per la Specifica di Sistemi in Tempo Reale basato su un Formalismo Logico*, Meda, R., Morzenti, A., Ratto, E., Roncato, M., Proc. 4th Italian Conference on Logic Programming, Bologna, June 1989.

[MMW84] *Tablog: The deductive tableau programming language*; Malachi, Y., Manna, Z. and Waldinger, R., Proc. ACM LISP and Functional Programming Conference, 1984.

[Mor89] *The Specification of Real-Time Systems: Proposal of a Logical Formalism*, Morzenti, A., PhD Thesis, Dipartimento di Elettronica, Politecnico di Milano, 1989.

[MRRZ89] *TRIO, a Logic Formalism for the Specification of Real-Time Systems*, Morzenti, A., Ratto, E., Roncato, M., Zoccolante, L., 1st IEEE Euromicro Workshop on Real-Time, Como, Italy, May 1989.

[Pla81] Theorem Proving with Abstraction, Plaisted, D., Artificial Intelligence, 16, 1981.

[Pnu81] *The Temporal Semantics of Concurrent Programs*, Pnueli, A., Theoretical Computer Science, 13, North-Holland, 1981.

[ReUr71] *Temporal Logic*, Rescher, N., Urquhart, A., Springer Verlag, Wien-New York, 1981.

[Sho88] *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence* Shohan, Y., The MIT Press, 1988.

[Wol83] *Temporal Logic Can Be More Expressive*, Wolper, P., Information and Control 56, 1983.