Procedure di decisione per logiche modali classiche

Armando Tacchella

Tesi presentata per il conseguimento del titolo di

Laurea in
Ingegneria Informatica

Relatore: Chiar.$^{\underline{\text{mo}}}$ Prof. Mauro Di Manzo

Correlatori: Dott. Ing. E. Giunchiglia
Prof. F. Giunchiglia

DIST
Dipartimento di Informatica Sistemistica e Telematica

Università di Genova
Genova - Italia

Così si presenta con la vita qualcosa di totalmente nuovo nel mondo, già con la singola cellula, qualcosa che prima non esisteva; problemi e tentativi attivi di soluzione; valutazioni, valori; tentativo ed errore.

Karl R. Popper "Alla ricerca di un mondo migliore"

Ringraziamenti

Intendo ringraziare il mio relatore, professor Mauro Di Manzo, che ha reso possibile questo lavoro e i miei due correlatori, Fausto ed Enrico Giunchiglia che mi hanno sempre seguito con attenzione e pazienza, senza farmi mai mancare consigli e incoraggiamenti. Personalmente sono orgoglioso del fatto che tre persone che stimo molto per qualità umane e professionali abbiano deciso di presentarmi come loro candidato.

In via ufficiosa,molte altre persone sono state da me coinvolte, a vario titolo, nella realizzazione di questo lavoro ed anche a loro sono debitore di un ringraziamento. In particolare: Roberto Sebastiani, per avermi dato più consigli e dedicato più tempo di quello che questi ringraziamenti possono ripagare e per avermi fornito materiale prezioso tra cui la sua tesi di dottorato su procedure SAT-based e il codice LISP sui cui ho basato la realizzazione delle procedure di decisione(e la mitica 127!); Marco Roveri, che ha programmato l'ambiente di editing con cui questa tesi è stata realizzata, che ha realizzato le macro LaTeX con cui è stata impaginata e che ho spesso usato come "help in linea" per tutto quello che riguardasse l'aspetto realizzativo di questo lavoro; Silvio Ranise, Adolfo Villafiorita e Alessandro Armando che mi ricordo di aver tampinato per problemi con dimostrazioni, codice e analisi sperimentali; Simone Ruffino per avermi dato "qualche dritta" su `MATLAB`; tutti i membri di MRG-DIST che hanno sopportato non poche ore di processi in background su varie macchine nonché mio fratello Alberto che ha dovuto rinunciare al PC di casa, comprato in società, perché "...c'è un test che gira!".

Ringrazio i miei amici e compagni di laboratorio Marco e Silvio. In questi ultimi anni, abbiamo formato un gruppetto compatto ed agguerrito di giovani cabarettisti: con loro ho passato alcuni dei momenti più belli e spensierati della mia vita. Anche sul lavoro, la collaborazione con entrambi è stata sempre fruttuosa e il confronto piacevole e istruttivo. Singolarmente, ringrazio Silvio per avermi fatto conoscere la realtà del gruppo MRG e per essere stato un compagno di studi affidabile e preciso e ringrazio Marco per "esserci sempre passato prima" ed essere sempre stato in grado di darmi utili e preziosi consigli. Infine, ringrazio entrambi per avermi corretto una versione iniziale dell'introduzione di questa tesi.

Voglio ringraziare Enrico, nella sua veste ufficiosa di amico, per tutto quello che in quest'anno di lavoro abbiamo costruito assieme e per avermi sempre fatto sentire parte di una "squadra" motivata e professionale.

Ringrazio tutti i membri del gruppo MRG di Genova che non ho ancora citato; in rigoroso ordine sparso ringrazio Alessandro Coglio (per spartiti, appunti, consigli e le citazioni dai "Fantozzi"), Emanuele Foresti, Claudio Castellini e Alessandro Massarotto (che ringrazio per la mitica talk alle 5 del mattino!). A tutti va il ringraziamento per aver saputo creare un'ambiente amichevole e rilassante in cui le ore di lavoro scorrono più piacevolmente. Un pensiero anche per il gruppo MRG di Trento e, in particolare, un ringraziamento a Paolo Pecchiari, Alessandro Cimatti e Giovanna Venzano.

Ringrazio il mio amico e compagno di studi Federico Scottoni, che mi ha insegnato un metodo di studio e l'amore per la pulizia formale in cambio di pochi rudimenti sui linguaggi di programmazione. I suoi consigli e il suo esempio sono stati fondamentali affinché la mia carriera universitaria finisse molto meglio di come era cominciata.

Ringrazio il mio amico e compagno di studi Marco De Luca per essersi sempre preoccupato della mia salute mentale e i miei compagni di studi Giuseppe Guagliardo, Giulio Pasotti e Stefano Ricciardi con i quali ho riso, scherzato e tra le altre cose, preparato esami.

Ringrazio tutta la "banda dei chiavaresi" e cioè Massimo Corradi, Claudio Lavezzo, Attilio Rocca e, per estensione, Roberto Luscardo (visto che lui è di Moneglia!). A loro devo esempi, consigli, incoraggiamenti e anche qualche passaggio in macchina...

Ringrazio le "ragazze del DIST" e cioè, sempre in ordine sparso, Silvia, Raffaella, Lorella, Alessandra, Maria, Simona, Tiziana, Daniela, Anna, Bice, ecc. Un ringraziamento particolare a Daniela, per i tanti favori, e a Silvia, per avermi aiutato a spedire le lettere per l'eccezionalità. Un grazie a Marco Fato (la lettera per l'eccezionalità... e non solo!) e agli studenti del BioLab per la simpatia e i piacevoli intermezzi nelle giornate lavorative.

Ringrazio Mirko Alessi e sua madre Anna per avermi sempre fatto sentire "uno di casa" e per avermi insegnato con il loro esempio, più di tutto quello che io riuscirò mai a trasmettere con le mie parole. Voglio anche ringraziare tutti gli "amici di Mirko" con cui ho condiviso momenti di svago e di lavoro e, in particolare Lorenza e Daniela, i ragazzi di progetto 80 e Roberto Sperindio.

Un ringraziamento speciale per il mio amico Francesco Macciò. Anche se non riesco a esprimerlo compiutamente, io so che dentro questo lavoro c'è una parte che è anche merito suo e questa parte è l'amicizia che mi lega a lui.

Ringrazio tutti i miei amici (che nel periodo della tesi hanno sopportato qualche "pacco" più del solito) e, in particolare, Francesca Pozzi, Marco Leale e Nicola Pegazzano. Ringrazio anche i "compagni di scompartimento", Marco, Cristina, Paolo, Gabriella e Patrizia per la piacevole compagnia.

Ringrazio la professoressa Maria Cigliuti per un sacco di cose, non ultima quella di avermi sempre insegnato a vivere facendomi credere di essere io a insegnarle qualcosa. Un pensiero e un ringraziamento vanno anche al professor Letterio Rando che oggi non è più con noi ma il cui esempio è stato determinante per la strada che ho intrapreso. In onore del professor Claudio Mignone ho fatto sforzi incredibili per evitare il termine "implementare" nel corso di tutta questa tesi, come segno del rispettto e della stima che gli porto.

Ringrazio tutte le persone che mi hanno aiutato e hanno creduto in me e che non nomino esplicitamente affinché i ringraziamenti non diventino la tesi vera e propria!

Un ringraziamento sentito va a mio fratello Alberto per essere stato sempre paziente nei miei confronti e per aver perdonato qualche mia posizione da "fratello maggiore". Il suo aiuto è stato importante per la preparazione orale di molti esami e più di una volta ho usufruito dei suoi consigli tecnici e del suo appoggio morale in momenti difficili. La sua presenza e il suo spirito sono stati fondamentali per superare i momenti in cui la posizione di "figlio unico" sarebbe stata insostenibile. Grazie Alberto!

Se la vita è un dono, allora è sicuramente il dono più bello. Questa tesi è dedicata ai miei genitori Francesco e Giovanna che non mi hanno mai fatto mancare il sostegno materiale e morale necessario per completare questi cinque anni di studio. Spero che questo risultato possa compensarli, almeno in parte, per tutto quello che mi hanno dato.

Indice

1 Introduzione **1**

1.1 Motivazioni . . . 1

1.2 Stato dell'arte . . . 3

1.3 Obiettivi . . . 4

1.4 Struttura della tesi . . . 5

2 Logiche modali classiche **8**

2.1 Il problema dell'onniscienza logica . . . 8

2.2 Sintassi . . . 11

2.3 Semantica . . . 14

2.3.1 Le strutture $\mathcal{MS}$ . . . 14

2.3.2 Una semantica per le logiche modali . . . 17

2.3.3 Assegnamenti e loro proprietà . . . 20

2.4 Sistemi deduttivi modali . . . 22

3 Algoritmi di decisione SAT-based **28**

3.1 Schema di un algoritmo SAT-based . . . 28

3.2 Gli algoritmi di base . . . 30

3.2.1 Logica E . . . 30

3.2.2 Logica EM . . . 35

3.2.3 Logica EC . . . 37

3.3 Combinazione di algoritmi . . . 41

3.3.1 Logiche EN, EMN ed ECN . . . 41

3.3.2 Logica R (EMC) . . . 44

3.3.3 Logica K ($EMNC$) . . . 46

4 Realizzazione dei decisori **48**

4.1 Il decisore proposizionale DPLL . . . 48

4.2 Schema dei decisori modali DPLL-based . . . 51

4.3 Inferenza modale . . . 54

4.3.1 La logica E . . . 54

4.3.2 La logica EM . . . 56

4.3.3 La logica EN . . . 58

4.3.4 La logica EMN 59
4.3.5 La logica EC 61
4.3.6 La logica ECN 63
4.3.7 La logica R (EMC) 64
4.3.8 La logica K ($EMCN$) 65
4.4 Dimostrazione di correttezza e completezza 66
4.5 La costruzione di decisori efficienti 71
4.5.1 Le procedure di decisione per E, EM, EN EMN . . 72
4.5.2 Le procedure di decisione per EC, ECN, R 82
4.5.3 Un decisore per K 95

5 Risultati sperimentali 100
5.1 L'approccio sperimentale 100
5.2 Metodologia di testing 102
5.2.1 Generazione di formule 104
5.2.2 Ordinamento e semplificazione 107
5.2.3 Scelta dei parametri 111
5.2.4 Valutazione delle prestazioni 117
5.3 Presentazione e analisi dei risultati 121
5.3.1 La logica E 122
5.3.2 La logica K 132

6 Conclusioni e futuri sviluppi 139

A I decisori in LISP 140
A.1 La strutturazione del codice 140
A.2 La descrizione del linguaggio modale 144
A.3 Etichettatura di atomi modali 146
A.4 Il decisore `L-TAUT-DP` 148
A.5 Valutazione parziale di formule 150
A.6 Memoizing e inferenza modale per E 152
A.7 Gestione di insiemi 154
A.8 Memoizing e inferenza modale per $EMCN$ 156

Bibliografia 162

Capitolo 1

Introduzione

The Physical Symbol System Hypothesis.
“A physical symbol system has the necessary and sufficient means for general intelligent action”
[A. Newell and H.A. Simon, “Computer Science as Empirical Inquiry”]

1.1 Motivazioni

I contenuti di questa tesi hanno rilevanza nel contesto dei *sistemi multiagente* e, in particolare, nell'ambito della deduzione meccanizzata per sistemi di agenti *formalmente specificati*.

L'interesse nei sistemi multiagente, detti anche MAS (acronimo per *Multi Agent System*), deriva dalla loro possibilità di catturare un grande numero di situazioni che si presentano nello sviluppo di sistemi ed applicazioni per l'informatica. In generale, i sistemi MAS sono particolarmente adatti a rappresentare le architetture software/hardware costituite da moduli indipendenti e connessi tra di loro al fine di raggiungere un obiettivo comune. Si pensi, ad esempio, ai processi di uno stesso programma dedicati a compiti specifici diversi, oppure ai calcolatori connessi in una rete di telecomunicazione. Questi sono solo alcuni dei casi, peraltro molto generali, in cui è possibile fornire una descrizione del sistema in termini di MAS.

Nell'ambito più specifico della robotica e dell'intelligenza artificiale si hanno numerosi esempi in cui la cooperazione tra entità indipendenti assume un ruolo fondamentale. Si pensi, ad esempio, ai problemi di *task planning* in cui più robot autonomi si coordinano al fine di eseguire una particolare attività, quale il montaggio di un pezzo complesso o il trasporto di oggetti in particolari configurazioni, oppure ai problemi di *scheduling* in cui i limiti imposti eccedono le capacità operative di un singolo agente, che deve pertanto coordinarsi con altri al fine di rispettare i tempi di esecuzione prefissati.

La specifica formale dei sistemi multiagente consente di dedurre proprietà riguardanti il sistema oggetto della specifica utilizzando opportune tecniche logico-formali. La meccanizzazione di tali tecniche può essere applicata, ad esempio, alla *verifica automatica* di architetture hardware o software che possono essere specificate con sistemi multiagente.

Una possibile specifica formale per i MAS è la semantica *context-based* (descritta, ad esempio, in [Giunchiglia e Giunchiglia, 1996] e in [Benerecetti *et al.*, 1996]). In questo approccio, per agente (*reasoner*) si intende qualsiasi ente che sia in grado di credere in fatti riguardanti se stesso, altri agenti o il mondo esterno. L'importanza di questo approccio risiede nel fatto che un agente, per sua stessa natura, è in grado di ragionare non solo sulla sua condizione interna, ma può utilizzare anche quella che *crede* essere la condizione degli altri agenti. In altre parole, nelle architetture basate su contesti, un agente compie inferenze basandosi su una rappresentazione *opaca* dello stato interno degli altri agenti, cioè il modello è tale per cui nessun agente conosce esplicitamente lo stato interno degli altri agenti. Un modello come quello descritto cattura situazioni reali in cui gli agenti sono molto più "intelligenti" (*smart*) rispetto al caso in cui il lo stato interno di ogni agente sia noto: in quest'ultimo caso, infatti, l'intero modello sarebbe riconducibile ad un'unico agente avente come stato l'unione degli stati di tutti gli agenti.

In letteratura sono stati proposte diverse strutture per modellare la credenza (*belief*) degli agenti. Storicamente, il primo approccio formale in tal senso, viene fatto risalire all'opera [Hintikka, 1962] e alle strutture note come *semantiche a mondi possibili.* Dal punto di vista deduttivo, i sistemi formali che gestiscono operatori modali quali la credenza o la conoscenza di un agente sono detti logiche *epistemiche* o *modali.* Tali sistemi deduttivi sono un'estensione *decidibile* del linguaggio e delle regole di inferenza della logica proposizionale.

In particolare, le logiche corrette e complete rispetto alle semantiche a mondi possibili sono note come *normali* o *standard.* Il problema fondamentale nell'applicazione di queste logiche nel contesto dei MAS è il fatto che questo ci porta ad avere agenti *onniscienti* cioè agenti che credono in tutte le possibili conseguenze del loro ragionamento.

Dal nostro punto di vista, siamo interessati a sistemi di agenti *non onniscienti*, cioè agenti che non necessariamente credono in tutte le conseguenze del loro ragionamento. Il motivo di questa scelta (affrontato anche nella sezione 2.1) parte dall'osservazione che, in pratica, le entità rappresentate dagli agenti non hanno risorse sufficienti per permettersi di dedurre tutte le conseguenze dei fatti in cui credono.

In particolare, visto il tipo di inferenza che siamo interessati a meccanizzare, questa tesi è rivolta alle logiche modali *classiche*, dette anche *non normali*, che sono corrette e complete rispetto a semantiche con agenti non onniscienti. Dal punto di vista formale, i modelli delle logiche classiche sono un sovrainsieme dei modelli a mondi possibili per le logiche normali. Il nostro approccio risulta quindi molto generale in quanto ci consente di catturare diversi tipi di inferenza epistemica (anche onnisciente) semplicemente aggiungendo particolari vincoli sulle strutture (assiomi nelle logiche).

Nel senso che abbiamo chiarito, lo sviluppo di procedure di decisione per logiche modali classiche è strettamente legato alla necessità di meccanizzare l'inferenza in sistemi di agenti non onniscienti.

1.2 Stato dell'arte

Dal punto di vista teorico, le logiche modali classiche sono descritte e formalizzate in numerosi testi tra cui, ad esempio, [Chellas, 1980] e [Fagin *et al.*, 1995].

La descrizione del linguaggio logico, dei sistemi deduttivi in senso classico e l'introduzione di una semantica formale per il ragionamento non onnisciente sono contenute, relativamente al caso *monoagente*, in [Chellas, 1980]. Sempre nello stesso testo, sono riportate le dimostrazioni relative alla correttezza e completezza dei sistemi deduttivi e i risultati che attestano come il problema della soddisfacibilità sia decidibile in tali logiche.

In [Fagin *et al.*, 1995] viene indagata la connessione tra sistemi multiagente che realizzano inferenze non onniscienti e vengono presentati estensioni delle logiche classiche monoagente per la formalizzazione della deduzione. Complementari a questo testo, esistono altri lavori che generalizzano i risultati di [Chellas, 1980] alle logiche multiagente e, in particolare, [Vardi, 1986] e [Vardi, 1989] in cui, tra l'altro, viene investigata la complessità teorica del problema decisionale nelle logiche classiche.

Sempre dal punto di vista teorico, altri risultati rilevanti ai nostri fini, sono contenuti in [Giunchiglia *et al.*, 1993] dove viene proposta una semantica basata su contesti per il ragionamento in sistemi di agenti non onniscienti. L'idea fondamentale che mutuiamo da questo lavoro è che l'inferenza modale può essere vista come un'appropriata composizione di ragionamento proposizionale in contesti (teorie) distinti.

Algoritmi di decisione per alcune fra le logiche da noi prese in esame sono disponibili in [Fitting, 1983] mentre, ed esiste una traduzione presentata in [Gasquet e Herzig, 1996] da sistemi classici a sistemi normali che preserva la soddisfacibilità. Per quanto ne sappiamo, nessuna delle due tecniche è stata utilizzata per realizzare procedure di decisione che automatizzano l'inferenza per logiche modali classiche.

Dal punto di vista realizzativo, particolarmente rilevanti per la nostra indagine, risultano i lavori [Giunchiglia e Sebastiani, 1996a], [Giunchiglia e Sebastiani, 1996b]. In particolare in [Giunchiglia e Sebastiani, 1996b] viene presentata per la prima volta una procedura di decisione per la logica K (la più semplice logica normale) basata su un decisore proposizionale (la procedura Davis-Putnam-Longemann-Loveland descritta in [Davis e Putnam, 1960], [Davis *et al.*, 1962] e in [Armando e Giunchiglia, 1993]). I decisori modali realizzati con questa tecnologia, definita dagli autori "SAT-based", sono dimostrati essere superiori ad altri decisori allo stato dell'arte basati sui formalismi tableau (si veda [Fitting, 1983]). In [Giunchiglia e Sebastiani, 1996b] si giustificano i risultati ottenuti su base sperimentale e si dimostra come la tecnologia sia estendibile alla logica termininologica $\mathcal{ALC}$.

Molte delle conclusioni dei precedenti lavori sono raccolte e sistematizzate in [Sebastiani, 1997] dove, tra l'altro, si dimostra formalmente l'applicabilità del formalismo SAT-based a tutte le logiche modali normali e si descrive la metodologia di testing utilizzata per mostrare la superiorità in termini quantitativi e qualitativi delle realizzazioni SAT-based sui sistemi di deduzione automatica noti in letteratura (vedi anche [Giunchiglia *et al.*, 1996]).

Per quanto ne sappiamo, non esistono realizzazioni di sistemi che meccanizzano la deduzione in logiche modali classiche. Anche se considerati i risultati di [Sebastiani, 1997] si potrebbe pensare di realizzare procedure di decisione che sfruttino le traduzioni presentate in [Gasquet e Herzig, 1996], non siamo a conoscenza di sistemi descritti in letteratura che sfruttano tale paradigma. Ovviamente, all'assenza di procedure di decisione effettivamente realizzate, si accompagna l'assoluta mancanza di risultati sperimentali riguardanti la deduzione nelle logiche modali classiche.

1.3 Obiettivi

Dato il corrente stato dell'arte e le motivazioni che abbiamo esposto, questo lavoro si prefigge i seguenti obiettivi:

- elaborazione di algoritmi decisionali, provati corretti e completi rispetto alle logiche modali classiche;
- sviluppo di moduli software che realizzano tali algoritmi e quindi che meccanizzano la deduzione in sistemi di agenti non onniscienti;
- elaborazione di un metodo sperimentale per la raccolta e l'analisi dei dati relativi al testing dei decisori;
- sviluppo di un ambiente software atto a condurre indagini sperimentali articolate in:
  - controllo delle scelte realizzative e *performance tuning*;
  - confronto della procedura realizzata per logiche normali con il sistema descritto in [Sebastiani, 1997];
  - analisi della reale differenza in termini di prestazioni tra algoritmi PSPACE-hard e algoritmi NP-hard;

Con l'elaborazione degli algoritmi sistematizziamo l'approccio al problema della decisone nelle logiche modali classiche. I contributi forniti sono per molti aspetti innovativi, sia perché un formalismo SAT-based viene per la prima volta dimostrato capace di catturare il problema della decisione in tali logiche, sia perchè la formalizzazione proposta è alternativa a quelle comunemente presentate in letteratura.

La realizzazione delle procedure di decisione è la dimostrazione dell'applicabilità pratica dei formalismi SAT-based. Data l'assenza di sistemi che meccanizzano la deduzione in logiche modali classiche, anche la realizzazione delle procedure di decisione rappresenta un passo in avanti in tal senso.

L'elaborazione di un metodo sperimentale è un prerequisito essenziale per poter verificare l'efficacia della tecnologia da noi utilizzata. In particolare, la metodologia da noi proposta estende e completa quella utilizzata in [Giunchiglia *et al.*, 1996] per le logiche modali normali e anche in questo caso rappresenta il primo passo verso la definizione di un metodo sperimentale per l'analisi di decisori nelle logiche modali classiche.

Particolarmente interessante è, a nostro avviso, la comparazione del nostro decisore per la logica K, ottenuto impiegando le tecniche realizzative perfezionate con i decisori per le logiche non onniscienti, e il decisore allo stato dell'arte KSAT.

Infine, un'ulteriore contributo innovativo è dato dall'analisi empirica della complessità del ragionamento epistemico, effettuata comparando due procedure di decisione, entrambe SAT-based, che risolvono il problema della soddisfacibilità per due logiche in cui tale problema appartiene ad una diversa classe di complessità. In particolare, l'analisi da noi compiuta si concentra sulla differenza reale esistente tra la classe di problemi NP-complete e la classe di problemi PSPACE-complete. Il confronto avviene tra la procedura di decisione ESAT, che realizza un algoritmo NP-complete, e la procedura EMCNSAT, che realizza un algoritmo PSPACE-complete.

1.4 Struttura della tesi

Il lavoro è organizzato come come segue:

Capitolo 1: Introduzione. Vengono descritte le motivazioni, lo stato attuale della ricerca e gli obiettivi del lavoro.

Capitolo 2: Logiche modali classiche. Viene discussa la rilevanza di tali logiche dal punto di vista dell'indagine epistemica per i sistemi multiagente. Viene data la definizione del linguaggio delle logiche modali e vengono presentate le strutture semantiche $\mathcal{MS}$ che costituiscono una formalizzazione per sistemi di agenti non onniscienti. Infine vengono definiti gli otto distinti sistemi di logica modale in senso classico di cui ci occupiamo in questo lavoro. Per ogni sistema deduttivo vengono enunciate le proprietà di decidibilità, correttezza e completezza rispetto ai modelli $\mathcal{MS}$.

Capitolo 3: Algoritmi di decisione SAT-based. Oltre ad una descrizione generale della tecnologia SAT-based, vengono presentati gli algoritmi che sfruttano tale tecnologia per risolvere il problema delle soddisfacibilità nelle logiche modali classiche. Per ognuna delle logiche viene stabilita una connessione formale tra il ragionamento proposizionale e il ragionamento modale che dimostra in maniera rigorosa l'applicabilità della tecnologia SAT-based al problema di decisione a cui siamo interessati.

Capitolo 4: Realizzazione dei decisori. Viene presentata la procedura di decisione Davis-Putnam-Longemann-Loveland sui cui si basa la realizzazione degli algoritmi SAT-based. Per ogni logica viene presentata una versione "base" della procedura di decisione che permette di mettere in luce le caratteristiche deduttive che intendiamo meccanizzare. Viene fornita una dimostrazione di correttezza e completezza relativa ad un singolo decisore che si dimostra estendibile agli altri tenendo conto dei risultati del capitolo 3. Infine, per ogni logica presa in esame, viene presentata la corrispondente procedura di decisione nella versione effettivamente realizzata, cioè completa delle tecniche da noi introdotte per automatizzare in maniera efficiente la deduzione.

Capitolo 5: Risultati sperimentali. Viene presentata la metodologia utilizzata per analizzare su base sperimentale le prestazioni dei decisori. In particolare, viene descritta ad alto livello l'architettura dell'ambiente di testing e vengono formiti particolari riguardanti:

- la generazione di formule casuali,
- la semplificazione e l'ordinamento delle formule (*preprocessing*),
- la scelta dei parametri per la generzione delle formule,
- la scelta dei valori in base ai quali svolgere l'analisi dei dati raccolti.

Vengono inoltre presentati i *case study* relativi a due fra le logiche prese in esame. In particolare l'analisi relativa a K vede il confronto tra la nostra procedura di decisione e il decisore allo stato dell'arte presentato in [Sebastiani, 1997]. Un'ultima sezione è dedicata al confronto tra il comportamento del decisore per E (il cui problema di soddisfaciblità è NP-complete) e il comportamento del decisore per K (il cui test di soddisfacibilità è PSPACE-complete).

Capitolo 6: Conclusione. Riassume i punti rilevanti del lavoro descrito e traccia possibili linee di sviluppo future.

Appendice A:I decisori in LISP. Viene descritta l'architettura software con cui sono realizzate le procedure di decisione e presenta le parti più rilevanti del codice sorgente.

Capitolo 2

Logiche modali classiche

> "Non nello spazio debbo cercare la mia dignità, ma nel retto uso del mio pensiero. Non avrei nulla di più con il possesso di terre; mediante lo spazio l'universo mi comprende e mi inghiotte come un punto; mediante il pensiero, io lo comprendo"
> [Blaise Pascal, "Pensieri"]

Lo scopo di questo capitolo è quello di fornire una breve introduzione alle logiche modali classiche per le quali abbiamo sviluppato le procedure di decisione oggetto di questa tesi. Vengono pertanto esposte, oltre alle definizioni formali, anche le ragioni per cui le logiche modali classiche sono interessanti nella formalizzazione dei sistemi che modellano la conoscenza. Il compito della sezione 2.1 è proprio quello di introdurre tali argomentazioni. Le sezioni successive di questo capitolo sono dedicate alle questioni formali. In 2.2 definiamo la *sintassi* e cioè il modo con cui frasi concernenti una qualche *modalità* sono tradotte in formule simboliche. Nella sezione 2.3 parliamo delle strutture che danno corpo ad un particolare tipo di inferenza modale e quindi un senso ai simboli del linguaggio. Infine in 2.2 introduciamo i sistemi logico-formali corretti e completi rispetto alle strutture semantiche.

2.1 Il problema dell'onniscienza logica

La *logiche epistemiche*, cioè le logiche delle nozioni epistemiche come *conoscenza* e *credenza* costituiscono una delle principali aree di ricerca nel campo dell'intelligenza artificiale e dell'informatica teorica. Le questioni epistemologiche, pur essendo una tematica di rilevante interesse scientifico, sono state affrontate sino ai tempi più recenti in termini puramente filosofici (si veda ad esempio [Russell, 1956]). Anche se più tardi rispetto alla logica proposizionale o al calcolo dei predicati, le logiche modali hanno beneficiato dell'introduzione dei sistemi formali come strumento di indagine e ricerca scientifica. In letteratura (vedi [Fagin *et al.*, 1995] al riguardo) il primo approccio alla teoria della conoscenza fondato su semantiche formali viene fatto risalire all'opera di Hintikka 1962. Le nozioni fondamentali di questa caratterizzazione sono un insieme W di *mondi* (detti anche *stati*) e una relazione tra i mondi detta di *possibilità* o di *accessibilità*. La conoscenza di un agente[1] che è situato in un particolare mondo w è caratterizzato da un sottoinsieme W_w di W che rappresenta l'insieme delle *alternative epistemiche* a w; l'agente in questione conosce (o crede) un fatto φ se φ è soddisfacibile in tutti i mondi W_w. Nel seguito, riferendoci a questo tipo di semantica, parleremo di strutture $\mathcal{K}$ o di *semantica a mondi possibili*. La descrizione che abbiamo dato è sufficiente per continuare la nostra discussione ma è ben lungi dall'essere una trattazione approfondita di tali strutture: per ulteriori approfondimenti si rimanda ai capitoli 3 e 4 di [Chellas, 1980] e al capitolo 2 di [Fagin *et al.*, 1995].

Un problema fondamentale del modello sino ad ora delineato, è il cosiddetto problema dell'*onniscienza logica*, così definito dallo stesso Hintikka che per primo ne dimostrò l'esistenza. Essenzialmente, il fatto è che in una semantica a mondi possibili un agente conosce sempre tutte le conseguenze della sua conoscenza. Più formalmente, utilizzando il consueto simbolo "$\Box$" per denotare la conoscenza dell'agente, l'assioma $\Box\varphi \wedge \Box(\varphi \to \psi) \to \Box\psi$ risulta una proprietà fondamentale per le strutture $\mathcal{K}$. Questa situazione è, in molti casi di interesse, non intuitiva o quantomeno, non realistica. L'esempio forse più ovvio è proprio il ragionamento degli esseri umani. Gli "agenti", in questo caso, sono tipicamente non onniscienti: una persona può conoscere un insieme di fatti senza conoscerne tutte le possibili conseguenze. Per esempio, è possibile conoscere tutte le regole per il gioco degli scacchi senza peraltro essere in grado di determinare se in un "mondo possibile", rappresentato da una certa configurazione dei pezzi sulla scacchiera, esiste una strategia vincente per i propri pezzi.

Quando consideriamo situazioni reali, la mancanza di onniscienza logica può essere originata da svariati motivi. Una causa importante potrebbe essere la mancanza di "potenza di calcolo": per esempio un agente potrebbe non disporre sufficienti risorse, in termini di tempo o spazio, tali da essere in grado di determinare una strategia vincente nel gioco degli scacchi. Altre volte l'agente potrebbe essere "in cattiva fede" nel senso che, pur disponendo delle risorse necessarie, non le utilizza per computare le conseguenza della sua conoscenza.

Dal punto di vista formale, in [Vardi, 1986] il problema dell'onniscienza logica viene imputato all'interpretazione che le strutture $\mathcal{K}$ assegnano alla conoscenza in termini di relazioni di accessibilità tra mondi. Quello che viene suggerito in [Vardi, 1989] è un modello formale alternativo, la semantica alla Montague-Scott (strutture $\mathcal{MS}$ da ora in poi). In questo modello della conoscenza abbiamo *strutture epistemiche* che, pur mantenendo l'idea

[1]con il termine "agente" si vuole denotare l'ente in grado di ragionare all'interno del modello sfruttandone le proprietà logiche.

dei mondi multipli, non utilizzano il concetto di relazione di accessibilità. La conoscenza è caratterizzata da un insieme di *proposizioni*[2] associate ad un certo mondo. Il fatto che la conoscenza sia, in un certo senso, assegnata esplicitamente, ci fa pensare che sia possibile modellare agenti con diverse potenze di calcolo semplicemente imponendo particolari condizioni di chiusura sull'insieme di proposizioni note nei mondi delle strutture. In particolare è addirittura possibile modellare agenti logicamente onniscienti nel senso prima descritto e cioè le strutture $\mathcal{K}$ risultano una particolare sottoclasse delle strutture $\mathcal{MS}$. Per ora, rimandiamo la dimostrazione di queste proprietà alla presentazione dettagliata delle strutture $\mathcal{MS}$ contenuta nella sezione 2.3.

Una volta scelto il particolare modello di conoscenza che si intende adottare è necessario anche introdurre l'aspetto sintattico è cioè definire quali sono le logiche simboliche la cui deduzione trova una corrispondenza nelle strutture $\mathcal{MS}$. In particolare siamo interessati a sistemi di logiche modali che siano corretti e completi rispetto a particolari classi di strutture $\mathcal{MS}$ (vedi la sezione 2.4 al proposito). Tali logiche modali, sono note come *classiche* o *non normali*, per distinguerle dalle logiche *normali* che sono corrette e complete rispetto alla semantica $\mathcal{K}$.

La relativa "generalità" delle strutture $\mathcal{MS}$ fa sì che le logiche modali classiche comprendano un grande numero di possibili sistemi deduttivi. In particolare noi considereremo solamente la gerarchia comunemente presentata in letteratura (si veda al riguardo [Chellas, 1980]) che va dalla logica modale nota con il nome di E, la logica più semplice dal punto di vista del ragionamento epistemico, sino alla logica K e cioè la logica corretta e completa rispetto alle strutture $\mathcal{K}$.

Un'ultima osservazione riguarda le procedure di decisione presentate in questo lavoro. Essendo procedure per il test di soddisfacibilità nelle logiche modali classiche, esse meccanizzano esattamente la forma di ragionamento non onnisciente di cui si è finora discusso. Senza ritornare sull'importanza del particolare modello di conoscenza utilizzato, è bene osservare che la non onniscenza non implica necessariamente una minore complessità delle procedure di decisione. Di fatto gli algoritmi per la soddisfacibilità in $\mathcal{MS}$ sono provati essere *al meglio* NP-hard e in qualche caso addirittura PSPACE-hard. Problemi come questi sono sempre stati considerati uno scoglio insormontabile anche per gli algoritmi più efficienti e questo dimostra che il particolare ragionamento epistemico a cui i nostri algoritmi sono dedicati rappresenta una sfida decisamente interessante, anche dal punto di vista applicativo.

[2] In letteratura si chiama "proposizione" o "intensione" di un fatto l'insieme dei mondi per cui il fatto è vero.

2.2 Sintassi

Presentiamo ora alcune nozioni di base sul linguaggio delle logiche modali. Le definizioni e la notazione introdotte in questa sezione sono utilizzate nel corso di tutta la tesi e, pur non essendo indispensabili per la comprensione del lavoro, sono qui esposte per completezza e con la funzione di riferimento per gli argomenti presentati.

Iniziamo con la definizione del linguaggio logico che utilizziamo:

Definizione 2.1 (Linguaggio modale) *Dato un insieme enumerabile di formule atomiche* $\{P_0, P_1, P_2, \ldots\}$*, e dato l'insieme degli operatori* $\{\neg, \wedge, \Box\}$ *chiamiamo linguaggio modale l'insieme* $\mathcal{L}$ *definito come segue:*

- $P_n \in \mathcal{L}$ *per* $n = 0, 1, 2, \ldots$
- $\neg\varphi \in \mathcal{L}$ *se e solo se* $\varphi \in \mathcal{L}$
- $\varphi \wedge \psi \in \mathcal{L}$ *se e solo se* $\varphi, \psi \in \mathcal{L}$
- $\Box\varphi \in \mathcal{L}$ *se e solo se* $\varphi \in \mathcal{L}$

Usiamo inoltre abbreviazioni standard quali:

- $\top$ usato nel senso di $\varphi \vee \neg\varphi$;
- $\bot$ usato nel senso di $\varphi \wedge \neg\varphi$;
- $\varphi_1 \vee \varphi_2$ per $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$;
- $\varphi_1 \to \varphi_2$ per $\neg(\varphi_1 \wedge \neg\varphi_2)$;
- $\varphi_1 \leftrightarrow \varphi_2$ per $\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\varphi_2 \wedge \neg\varphi_1)$.

Le formule atomiche sono anche chiamate *atomi proposizionali* e talvolta *letterali*, mentre le formule del tipo $\Box\varphi$ sono chiamate *atomi modali*. L'operatore unario "$\Box$" è chiamato *operatore modale* o anche *operatore epistemico*. Indichiamo le formule di $\mathcal{L}$ con le lettere greche minuscole $\alpha, \beta, \ldots$ e gli insiemi di formule con le lettere maiuscole $\Gamma, \Delta, \ldots$; in particolare Σ denota un generico sistema deduttivo e Λ viene utilizzata per denotare una qualsiasi logica fra quelle presentate nella sezione 2.4. Seguiamo inoltre le seguenti convenzioni per congiunzioni e disgiunzioni multiple:

- $\varphi_1 \wedge \ldots \wedge \varphi_n$ è indicata come $\bigwedge_i^n \varphi_i$ talvolta omettendo l'apice n quando dal contesto è chiaro il numero dei congiunti φ_i. Se $n = 0$ $\bigwedge_i^n \varphi_i = \top$.
- $\varphi_1 \vee \ldots \vee \varphi_n$ è indicata come $\bigvee_i^n \varphi_i$ talvolta omettendo l'apice n quando dal contesto è chiaro il numero dei disgiunti φ_i. Se $n = 0$ $\bigvee_i^n \varphi_i = \bot$

Un'ultima considerazione riguardante le formule del tipo $\bigwedge_i^n \varphi_i \to \varphi$. Nel caso in cui $n = 0$ assumiamo che tali formule coincidano con il conseguente e cioè $\bigwedge_i^0 \varphi_i \to \varphi$ equivale a φ.

Diamo ora alcune definizioni strettamente collegate con la forma sintattica delle formule modali. Definiamo innanzitutto il concetto di profondità *modale* di una formula φ, cioè il numero massimo di operatori modali annidati all'interno di qualche ramo sintattico della formula.

Definizione 2.2 (Profondità modale di φ) *Data $\varphi \in \mathcal{L}$, definiamo induttivamente la funzione depth* : $\mathcal{L} \to \mathbf{N}$:

- *per* $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $depth(\varphi) = 0$;
- *per* $\varphi = \Box\psi$ *con* $\psi \in \mathcal{L}$, $depth(\varphi) = depth(\psi) + 1$;
- *per* $\varphi = \neg\psi$ *con* $\psi \in \mathcal{L}$, $depth(\varphi) = depth(\psi)$;
- *per* $\varphi = \psi \wedge \xi$ *con* $\psi, \xi \in \mathcal{L}$, *risulta* $depth(\varphi) = \max(depth(\xi), depth(\psi))$.

Per esempio, diremo che la formula $\Box(\Box P_1 \to P_2)$ ha profondità due, in quanto P_1 si trova all'interno di $\Box P_1$ che è a sua volta all'interno di $\Box(\ldots)$.

Vediamo invece cosa intendiamo per profondità *sintattica* di una formula. Al riguardo possiamo dare due definizioni differenti di cui la prima considera la formula nel suo complesso, mentre la seconda si limita ad "espandere" un'unico livello modale.

Definizione 2.3 (Profondità sintattica) *Data $\varphi \in \mathcal{L}$, definiamo induttivamente la funzione dtree* : $\mathcal{L} \to \mathbf{N}$:

- *per* $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $dtree(\varphi) = 1$;
- *per* $\varphi = \Box\psi$ *con* $\psi \in \mathcal{L}$, $dtree(\varphi) = dtree(\varphi) + 1$;
- *per* $\varphi = \neg\psi$ *con* $\psi \in \mathcal{L}$, $dtree(\varphi) = dtree(\psi) + 1$;
- *per* $\varphi = \psi \wedge \xi$ *con* $\psi, \xi \in \mathcal{L}$, $dtree(\varphi) = \max(dtree(\psi), dtree(\xi)) + 1$.

Notiamo che la funzione *dtree* così definita risulta essere la massima profondità dell'albero sintattico di una formula. Consideriamo ad esempio $varphi = \Box(\Box P_1 \wedge P_2) \wedge \neg P_2$ il cui albero di costruzione sintattica è riportato in figura 2.1. Utilizzando la definizione 2.3 risulta $dtree(\varphi) = 5$ che è appunto il numero di nodi incontrati sul percorso più lungo dalla radice φ alla foglia P_1.

Definizione 2.4 (Profondità sintattica ad un livello) *Data $\varphi \in \mathcal{L}$, definiamo induttivamente la funzione dlevel* : $\mathcal{L} \to \mathbf{N}$:

- *per* $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $dlevel(\varphi) = 1$;

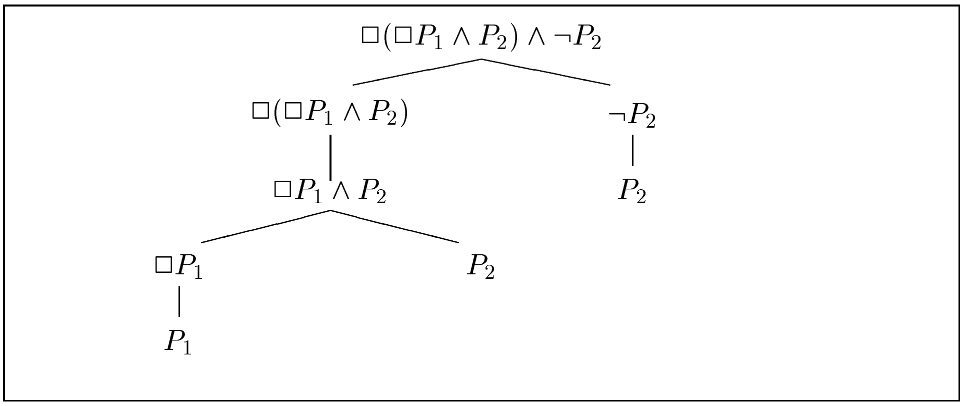

Figura 2.1: L'albero sintattico di $\Box(\Box P_1 \wedge P_2) \wedge \neg P_2$

- *per* $\varphi = \Box\psi$ *con* $\psi \in \mathcal{L}$, $dlevel(\varphi) = 1$*;*
- *per* $\varphi = \neg\psi$ *con* $\psi \in \mathcal{L}$, $dlevel(\varphi) = dlevel(\psi) + 1$*;*
- *per* $\varphi = \psi \wedge \xi$ *con* $\psi, \xi \in \mathcal{L}$, $dlevel(\varphi) = \max(dlevel(\psi), dlevel(\xi)) + 1$.

Notiamo come la funzione *dlevel* così definita risulta essere la masssima profondità dell'albero sintattico di φ, ottenuto considerando atomiche le sottoformule modali di φ e pertanto differisce dalla funzione *dtree* definita in 2.3. Come esempio, consideriamo la formula φ presentata in figura 2.1. Risulta $dlevel(\varphi) = 3$ in quanto la sottoformula $\Box(\Box P_1 \wedge P_2)$, considerata come un atomo, non dà origine ad alcun sottoalbero.

Diamo ora l'usuale definizione dell'insieme composto da tutte le possibili sottoformule di φ.

Definizione 2.5 (Insieme delle sottoformule) *Data* $\varphi \in \mathcal{L}$, *definiamo induttivamente l'insieme* $sub(\varphi) \subseteq \mathcal{L}$ *sottoformule di* φ *come segue:*

- *per* $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $sub(\varphi) = \{\varphi\}$*;*
- *per* $\varphi = \Box\psi$ *con* $\psi \in \mathcal{L}$, $sub(\varphi) = \{\varphi\} \cup sub(\psi)$*;*
- *per* $\varphi = \neg\psi$ *con* $\psi \in \mathcal{L}$, $sub(\varphi) = \{\varphi\} \cup sub(\psi)$*;*
- *per* $\varphi = \psi \wedge \xi$ *con* $\psi, \xi \in \mathcal{L}$ *e* $sub(\varphi) = \{\varphi\} \cup sub(\psi) \cup sub(\xi)$.

Sfruttando ancora la formula φ presentata in figura 2.1, troviamo che $sub(\varphi) = \{\varphi, \Box(\Box P_1 \wedge P_2), \neg P_2, \Box P_1 \wedge P_2, \Box P_1, P_2, P_1\}$ è proprio l'insieme di tutti i nodi dell'albero sintattico di φ.

In qualche caso saremo interessati alle sole sottoformule modali, cioè ai nodi dell'albero sintattico costituiti da atomi modali. Pertanto diamo la seguente:

Definizione 2.6 (Insieme delle sottoformule modali) *Data $\varphi \in \mathcal{L}$, definiamo induttivamente l'insieme $sub_\Box(\varphi) \subseteq \mathcal{L}$ sottoformule modali di φ come segue:*

- *per $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $sub_\Box(\varphi) = \emptyset$*
- *per $\varphi = \Box\psi$ con $\psi \in \mathcal{L}$, $sub_\Box(\varphi) = \{\psi\} \cup sub_\Box(\psi)$;*
- *per $\varphi = \neg\psi$ con $\psi \in \mathcal{L}$, $sub_\Box(\varphi) = sub_\Box(\psi)$;*
- *per $\varphi = \psi \wedge \xi$ con $\psi, \xi \in \mathcal{L}$, $sub_\Box(\varphi) = sub_\Box(\psi) \cup sub_\Box(\xi)$.*

Con la formula φ dell'esempio precedente vediamo che in questo caso $sub_\Box(\varphi) = \{(\Box P_1 \wedge P_2), P_1\}$ e quindi risulta $sub_\Box(\psi) \subset sub(\psi)$.

Concludiamo questa sezione con il concetto di insieme di atomi di una formula. In questo caso siamo interessati all'insieme delle foglie dell'albero sintattico ottenuto considerando atomiche le sottoformule modali di una formula.

Definizione 2.7 (Insieme degli atomi) *Data $\varphi \in \mathcal{L}$ definiamo induttivamente l'insieme $Atom(\varphi) \subseteq \mathcal{L}$:*

- *per $\varphi \in \{P_0, P_1, P_2, \ldots\}$, $Atom(\varphi) = \{\varphi\}$;*
- *per $\varphi = \Box\psi$ con $\psi \in \mathcal{L}$, $Atom(\varphi) = \{\varphi\}$;*
- *per $\varphi = \neg\psi$ con $\psi \in \mathcal{L}$, $Atom(\varphi) = Atom(\psi)$;*
- *per $\varphi = \psi \wedge \xi$ con $\psi, \xi \in \mathcal{L}$ e $Atom(\varphi) = Atom(\psi) \cup Atom(\xi)$.*

2.3 Semantica

Abbiamo già discusso nella sezione 2.1 dei problemi originati dal modello della conoscenza basato sulla semantica alla Hintikka. Abbiamo anche introdotto, in maniera informale, i modelli proposti in [Vardi, 1989] al fine di realizzare un modello della conoscenza alternativo a $\mathcal{K}$. Nella sezione 2.3.1 introduciamo formalmente le strutture $\mathcal{MS}$ mostrandone definizioni e proprietà e nella sezione 2.3.2 le particolari classi di $\mathcal{MS}$ che sono interessanti ai fini della nostra ricerca. Infine, nella sezione 2.3.3 introduciamo il concetto di *assegnamento* per una formula modale.

2.3.1 Le strutture $\mathcal{MS}$

Iniziamo la trattazione dei modelli $\mathcal{MS}$ (detti anche *modelli minimali*) con la definizione formale della loro struttura.

Definizione 2.8 (Struttura $\mathcal{MS}$) *Una tripla $M = \langle W, N, P\rangle$ è una struttura $\mathcal{MS}$ se e solo se:*

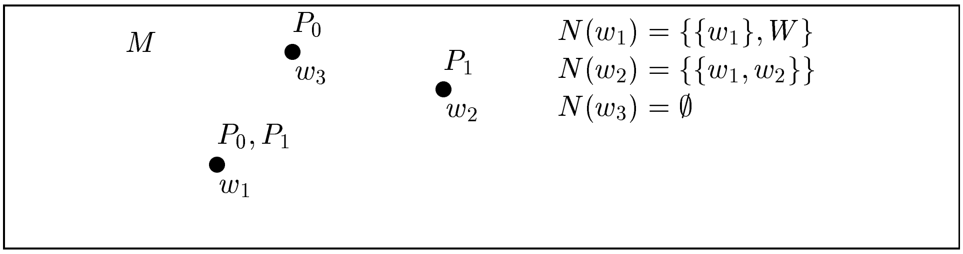

Figura 2.2: Una struttura $\mathcal{MS}$

- *W è un insieme*
- *N è una relazione $N : W \to \mathcal{P}\ (\mathcal{P}\ (W))$, cioè tale che per ogni $w \in W$, $N(w) \subseteq \mathcal{P}\ (W)$.*
- *P è una relazione $P : \mathbf{N} \to \mathcal{P}\ (W)$, cioè per ogni n $P(n) \subseteq W$*

Denotiamo con $\mathcal{E}$ la classe di tutti i modelli che soddisfano la definizione 2.8. In particolare, per ogni $M \in \mathcal{E}$ chiamiamo l'insieme W *insieme dei mondi* e utilizziamo le lettere w, u, v (talvolta con pedici) per riferirci ai suoi elementi Per indicare N usiamo il termine di *assegnamento epistemico* del modello M e chiamiamo P *assegnamento intensionale* di particolare M. Ancora, un modello è detto *finito* se il suo insieme di mondi ha solo un numero finito di elementi e *infinito* nel caso contrario. I modelli finiti sono ovviamente quelli interessanti dal punto di vista delle procedure che presentiamo in questa tesi.

Introduciamo ora il concetto di *verità* di una formula $\varphi \in \mathcal{L}$ in un modello M e in un mondo $w \in W$ con la seguente:

Definizione 2.9 *Dati un modello $M = \langle W, N, P \rangle \in \mathcal{E}$, un mondo $w \in W$ e una formula $\varphi \in \mathcal{L}$, diciamo che φ è vera nel modello M e nel mondo $w \in W$, indicato con $M, w \models \varphi$, nei seguenti casi:*

- *se $\varphi = P_n$ con con $P_n \in \{P_0, P_1, P_2, \ldots\}$ allora $M, w \models \varphi$ se e solo se $w \in P(n)$;*
- *se $\varphi = \neg\psi$ allora $M, w \models \varphi$ se e solo se non vale $M, w \models \psi$;*
- *se $\varphi = \psi_1 \wedge \psi_2$ allora $M, w \models \varphi$ se e solo se vale $M, w \models \psi_1$ ed anche $M, w \models \psi_2$;*
- *se $\varphi = \Box\psi$ allora $M, w \models \varphi$ se e solo se $\{u \mid M, u \models \psi\} \in N(w)$;*

Nel seguito, utilizziamo anche $M, w \not\models \varphi$ per indicare il fatto che φ non vale nel mondo w del modello M. Data la semantica attribuita a "$\neg$" risulta che $M, w \not\models \varphi$ se e solo se $M, w \models \neg\varphi$.

Ora che siamo in possesso della nomenclatura e delle nozioni di base, ci soffermiamo brevemente sul loro significato intuitivo. Si consideri l'esempio in figura 2.2 in cui è raffigurata una semplice struttura $M \in \mathcal{E}$ che consta di tre mondi e cioè tale che $W = \{w_1, w_2, w_2\}$. I mondi sono rappresentati in figura come punti etichettati dal loro nome. Abbiamo elencato solamente i particolari atomi di $\{P_0, P_1, P_2, \ldots\}$ che intendiamo veri nel particolare mondo e quindi risulta $P(0) = \{w_1, w_3\}$, $P(1) = \{w_1, w_2\}$ e $P(n) = \emptyset$ per $n \geq 2$. In base a ciò, risulterà $M, w_1 \models P_0$ ed anche $M, w_2 \models P_1$, mentre w_3 è tale che $M, w \not\models P_1$. Data la definizione di verità 2.9 risulta che i mondi sono completi per quanto concerne il ragionamento proposizionale. Applicando tale definizione all'esempio, possiamo concludere che $M, w_1 \models P_0 \wedge P_1$ oppure che $M, w_2 \models P_0 \wedge \neg P_1$. Inoltre vale la pena di notare che, qualunque sia l'assegnamento intensionale P, i mondi di M sono sempre tali da verificare le tautologie proposizionali. Ad esempio $P_1 \wedge P_1$ è vera per tutti i mondi.

Rimane ancora da chiarire a fondo il ruolo giocato da N nella semantica dell'operatore "$\Box$" e cioè il particolare meccanismo con cui le strutture $\mathcal{MS}$ modellano la conoscenza. Vediamo la seguente:

Definizione 2.10 (Intensione di una formula) *Data* $\varphi \in \mathcal{L}$ *e dato* $M = \langle W, N, P \rangle \in \mathcal{E}$ *indichiamo con* $\|\varphi\|^M \subseteq W$ *l'insieme definito come* $\|\varphi\|^M = \{u \in W \mid M, u \models \varphi\}$

L'insieme $\|\varphi\|^M$ verrà anche chiamato *proposizione* espressa dal fatto φ. Ora che siamo in possesso della nozione di intensione di una formula in un modello ritorniamo all'esempio di figura 2.2. Notiamo che sul lato destro è riportata la definizione degli insiemi $N(w)$ per tutti i mondi facenti parte del modello. Nelle parole di [Vardi, 1989], gli insiemi N(w) rappresentano un insieme di proposizioni *conosciute* (o credute) nel mondo w. In questa ottica, la semantica attribuita a "$\Box$" nella definizione 2.9, lega il fatto che $M, u \models \Box\varphi$ con il fatto che $\|\varphi\|^M \in N(u)$. In altre parole, posso dire che il fatto "conosco φ" è vero in un mondo di un modello se e soltanto se la proposizione "φ" è nell'insieme di proposizioni associate da N a quel mondo. Al riguardo si parla di rappresentazione *esplicita* della conoscenza (si veda [Fagin *et al.*, 1995] per eventuali approfondimenti). Nel modello di figura 2.2, per esempio, abbiamo che l'insieme $\|P_1\|^M = \{w_1, w_2\}$ è contenuto in $N(w_2)$ e quindi vale $M, w_2 \models \Box P_1$. Notiamo come, inserendo opportuni insiemi all'interno degli $N(w)$, possiamo modellare situazioni interessanti. Ad esempio, dato che $W \in N(w_1)$ risulta che se $M, w \models \varphi$ per tutti i $w \in W$, si ha anche $M, w_1 \models \Box\varphi$: il mondo w_1 è quindi automaticamente a conoscenza di tutte le formule valide in M. Infine, notiamo che l'assegnamento epistemico per un mondo può anche essere vuoto. È il caso di w_3 in cui risulta $M, w_2 \not\models \Box\varphi$ per qualsiasi $\varphi \in \mathcal{L}$: il mondo w_3 modella quindi la più totale "ignoranza" in senso logico.

Formalizziamo alcuni concetti sino a qui esposti in maniera discorsiva. In particolare diamo le seguenti definizioni, rispettivamente, di validità in un modello e in una classe di modelli $\mathcal{MS}$:

Definizione 2.11 *Data $\varphi \in \mathcal{L}$ diciamo che φ è valida in $M = \langle W, N, P \rangle \in \mathcal{E}$, denotato con $M \models \varphi$, se e solo se per ogni $w \in W$ si ha $M, w \models \varphi$*

Definizione 2.12 *Data $\varphi \in \mathcal{L}$ diciamo che φ è valida in una classe di modelli $\mathcal{M} \subset \mathcal{E}$ denotato con $\mathcal{M} \models \varphi$, se e solo se per ogni $M \in \mathcal{M}$ si ha $M \models \varphi$.*

Nel caso in cui $\mathcal{M} \equiv \mathcal{E}$ scriviamo indifferentemente $\models_{\mathcal{E}} \varphi$ oppure $\mathcal{E} \models \varphi$. Se $M \models \varphi$ diciamo che M è un modello per φ mentre con $M \not\models \varphi$ indichiamo il fatto che per almeno un mondo w del modello M si ha $M, w \not\models \varphi$ e diciamo che M è un *contromodello* per φ. Diciamo che φ è *soddisfacibile* in un modello M nel caso in cui $M \not\models \neg\varphi$, cioè esiste almeno un mondo w di M tale che $M, w \models \varphi$. Nel caso contrario diremo che φ è *insoddisfacibile* nel modello M. Nel caso in cui non esista un modello $M \in \mathcal{M}$ che soddisfa φ diremo che φ è insoddisfacibile nella classe dei modelli $\mathcal{M}$.

L'ultimo risultato di questa sezione stabilisce un'importante connessione tra la logica proposizionale (PL nel seguito) e le strutture $\mathcal{MS}$. In particolare, risulta che tali strutture sono corrette e complete rispetto al ragionamento proposizionale. Enunciamo il seguente:

Teorema 2.1 *Sia φ una conseguenza tautologica di $\bigwedge_i^n \varphi_i$ con $n \geq 0$. Se $\mathcal{M} \in \mathcal{E}$ è una qualsiasi classe di modelli tali che $\mathcal{M} \models \varphi_1, \ldots, \mathcal{M} \models \varphi_n$ allora risulta $\mathcal{M} \models \varphi$.*

La prova di un enunciato più generale di questo teorema è riportata in [Chellas, 1980]. In particolare esso è valido per una qualsiasi struttura $M = \langle W, \ldots, P \rangle$ che rispetti le condizioni da noi imposte, senza alcuna dipendenza dal particolare assegnamento epistemico N impiegato (si veda sempre [Chellas, 1980] per approfondimenti).

2.3.2 Una semantica per le logiche modali

In questa sezione introduciamo le classi di strutture $\mathcal{MS}$ che utilizziamo per dare una semantica formale alle logiche modali classiche. Nella sezione 2.4, quando trattiamo l'assiomatizzazione, riferendoci a tali classi parleremo spesso di *gerarchia* di strutture $\mathcal{MS}$. Il termine gerarchia suggerisce l'esistenza di una relazione ben precisa tra i suoi membri e, nel nostro caso, ad ogni passaggio verso un livello superiore corrisponde l'acquisizione di maggiore potenza deduttiva dal punto di vista epistemico.

Le strutture alla base della gerarchia sono proprio gli $M \in \mathcal{E}$ e cioè le generiche strutture $\mathcal{MS}$. La forma di ragionamento epistemico espressa da queste strutture è estremamente semplice come enunciato nel seguente:

Teorema 2.2 (Inferenza modale in $\mathcal{E}$) *Sia $\mathcal{M} \subseteq \mathcal{E}$ una qualsiasi classe di strutture $\mathcal{MS}$. Se $\mathcal{M} \models \varphi \leftrightarrow \psi$ allora $\mathcal{M} \models \Box\varphi \leftrightarrow \Box\psi$*

Dimostrazione - Sia $\mathcal{M} \subseteq \mathcal{E}$ la classe di strutture $\mathcal{MS}$ per cui $\mathcal{M} \models \varphi \leftrightarrow \psi$, cosicché, in base alla def. 2.10, si abbia $\|\varphi\|^M \equiv \|\psi\|^M$ per ogni $M \in \mathcal{M}$. Da questo segue che per qualsiasi mondo w in qualsiasi modello $M = \langle W, N, P\rangle \in \mathcal{M}$, $\|A\|^M \in N(w)$ se e solo se $\|\varphi\|^M \in N(w)$. Allora in ogni mondo w di ogni modello $M \in \mathcal{M}$ si ha $M, w \models \Box\varphi$ se e solo se $M, w \models \Box\psi$, che significa $\mathcal{M} \models \Box\varphi \leftrightarrow \Box\psi$.

Questo risultato, da solo, non basta a concludere che le strutture $\mathcal{MS}$ modellano agenti non onniscienti (tale risultato vale anche nelle semantiche alla Hintikka). Il teorema non ci dice infatti quali inferenze *non* siamo in grado di compiere. L'affermazione positiva in esso contenuta dice che, al meglio, non siamo in grado di distinguere tra formule logicamente equivalenti. In altre parole, questo significa che le strutture $\mathcal{MS}$ non sono "deboli" a tal punto da consentirci, contemporaneamente, sia di credere, sia di non credere allo stesso fatto. A tal proposito diciamo che in $\mathcal{E}$ le strutture non ci consentono conoscenza *contraddittoria*, che sembra essere il requisito minimo per una qualsiasi logica epistemica.

Al fine di ottenere un risultato limitativo riguardo alle capacità di inferenza modale nelle strutture in $\mathcal{E}$, diamo la seguente:

Definizione 2.13 (Schemi M, C, N) *Date $\varphi, \psi \in \mathcal{L}$ definiamo i seguenti schemi:*

M. $\Box(\varphi \wedge \psi) \rightarrow (\Box\varphi \wedge \Box\psi)$

C. $(\Box\varphi \wedge \Box\psi) \rightarrow \Box(\varphi \wedge \psi)$

N. $\Box\top$

La peculiarità di questi tre schemi è quella di rappresentare tre distinti aspetti dell'onniscenza logica. Se presi come assiomi, vincolano i modelli alla particolare sottoclasse di $\mathcal{MS}$ che abbiamo chiamato strutture $\mathcal{K}$. D'altronde, in generale vale il seguente risultato:

Teorema 2.3 (Minimalità di $\mathcal{E}$) *Nessuno tra gli schemi M, C, ed N è valido nella classe di tutte le strutture $\mathcal{MS}$*

Questo teorema (vedi [Chellas, 1980] per una dimostrazione) è tale da garantirci che la classe $\mathcal{E}$ composta dalle generiche strutture $\mathcal{MS}$ realizza la *più semplice inferenza modale possibile.*

Ora che abbiamo fissato la classe $\mathcal{E}$ quale nostre punto di partenza, vogliamo precisare le proprietà che le nostre strutture devono soddisfare per essere in grado di realizzare le inferenze epistemiche più sofisticate presentate nella definizione 2.13. Diamo allora la seguente:

Definizione 2.14 (Vincoli epistemici) *Dato $M = \langle W, N, P\rangle$, con $U, V \subseteq W$ si considerino le seguenti condizioni tali che per ogni mondo $w \in W$ risulta:*

(M) se $U \subseteq V$ e $U \in N(w)$ allora $V \in N(w)$;

(C) se $U \in N(w)$ e $V \in N(w)$ allora $U \cap V \in N(w)$;

(N) $W \in N(w)$

Nel seguito, per indicare le sottoclassi di $\mathcal{E}$ comprendenti tutti e soli gli elementi che soddisfano una qualche combinazione delle proprietà elencate nella definizione 2.14, aggiungeremo ad $\mathcal{E}$ i pedici M, N, C. Quindi, esempio, $\mathcal{E}_M \subset \mathcal{E}$ è la classe di tutti e soli i modelli che soddisfano la condizione (M), mentre $\mathcal{E}_{MC} \subset \mathcal{E}_M$ è la classe di tutti e soli i modelli che soddisfano sia la condizione (M) sia la condizione (C) (pertanto $\mathcal{E}_C \cap \mathcal{E}_M \equiv \mathcal{E}_{MC}$). A seconda del fatto che una struttura $\mathcal{MS}$ soddisfi una fra le condizioni (M), (C) e (N) diciamo che è un modello *supplementato*, *chiuso per intersezioni* e *contenente l'unità*. In particolare, un modello che possieda tutte e tre le caratteristiche viene chiamato *filtro*; nella nostra notazione, la classe di tutti e soli i filtri è data da $\mathcal{E}_{MCN}$: tutti i modelli di questa classe sono modelli semantici alla Hintikka (si veda [Chellas, 1980] per una dimostrazione che $\mathcal{K} \equiv \mathcal{E}_{MCN}$).

Esplicitiamo il legame che intendiamo stabilire tra la potenza dell'inferenza epistemica (così come espressa dalla definizione 2.13) e le condizioni che particolari sottoclassi di $\mathcal{E}$ devono verificare in modo da costituire il supporto a tale ragionamento. Formalmente:

Teorema 2.4 (Inferenza modale e proprietà semantiche) *Le strutture $\mathcal{MS}$ verificano le seguenti proprietà:*

- *lo schema M è valido nella classe dei modelli supplementati, pertanto risulta $\mathcal{E}_M \models \Box(\varphi \wedge \psi) \rightarrow (\Box\varphi \wedge \Box\psi)$;*
- *lo schema C è valido nella classe dei modelli chiusi per intersezioni, pertanto risulta $\mathcal{E}_C \models (\Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$;*
- *lo schema N è valido nella classe dei modelli che contengono l'unità, pertanto risulta $\mathcal{E}_N \models \Box\top$*

A questo punto è chiaro come sia possibile, combinando gli schemi M,C,N ottenere corrispondentemente dei vincoli sui modelli in $\mathcal{E}$ che ci consentano di ampliare le possibili inferenze modali rispetto al caso più generale.

Un'ultima precisazione riguarda la terminologia che utilizziamo. Anche nelle sezioni che seguono, come del resto abbiamo fatto sino ad ora, riferendoci alle formule del tipo "$\Box\varphi$" diciamo che *conosciamo* tali formule e parliamo di strutture che modellano la conoscenza. A ben vedere le strutture in $\mathcal{E}$ e tutte le sottoclassi che abbiamo considerato non soddisfano assiomi che consentano di precisare che l'operatore "$\Box$" denoti la conoscenza piuttosto che qualche altra modalità come, ad esempio, la credenza. A tal fine, i modelli proposti dovrebbero essere tali da soddisfare lo schema noto il letteratura con il nome di T (che corrisponde alla formula $\Box\varphi \rightarrow \varphi$). Di fatto così non è, in quanto in tutte le classi considerate la formula $\Box\varphi \wedge \neg\varphi$ risulta soddisfacibile. Pertanto nel corso del lavoro il termine "conoscenza" andrà inteso come una forma linguistica per denotare l'operatore "$\Box$" piuttosto che una affermazione formale sulla valenza epistemica di tale operatore.

2.3.3 Assegnamenti e loro proprietà

In questa sezione presentiamo il concetto di *assegnamento* per una formula modale. Costruire un assegnamento significa attribuire dei valori di verità $b \in \{0, 1\}$ agli atomi di una formula di modo che valutandola dopo aver sostituito i valori agli atomi corrispondenti, si ottenga "vero". Sfruttando un'analogia con il progetto di reti combinatorie, l'assegnamento è quell'insieme di valori che deve essere assegnato alle porte di ingresso per ottenere in uscita il livello logico corrspondente a "1". Fissiamo questa intuizione con la seguente:

Definizione 2.15 (Assegnamento per una formula) *Siano* $\varphi \in \mathcal{L}$ *e* $\psi_1, \psi_2 \in Atom(\varphi)$. *Un assegnamento per* φ *è una funzione* $\mu : Atom(\varphi) \rightarrow \{0, 1\}$ *che soddisfa le seguenti condizioni:*

- $\mu(\psi_1) = 1$ *se e solo se* $\mu(\neg\psi_1) = 0$*;*
- $\mu(\psi_1) = 1$ *se e solo se* $\mu(\psi_1) = 1$ *e* $\mu(\psi_2) = 1$*;*
- $\mu(\varphi) = 1$*;*

L'assegnamento così definito, considera solo le sottoformule di φ contenute in $Atom(\varphi)$. Richiamando brevemente la definizione 2.7 riguardante tale insieme, ci ricordiamo che $Atom(\varphi) \subseteq sub(\varphi)$. Quindi un assegnamento considera un solo livello della formula, assegnando valori di verità sia ad atomi proposizionali, sia ad atomi modali, ma senza considerare le sottoformule *all'interno* degli atomi modali. Al riguardo in [Sebastiani, 1997] si parla di assegnamenti che *soddisfano proposizionalmente* una formula.

Enunciamo ora una proprietà che lega l'insieme degli assegnamenti per una formula alla formula stessa. Vale cioè la seguente:

Proprietà 2.1 *Dato* $\mathcal{A} = \{\mu_1 \ldots \mu_r\}$ *che contiene tutti e soli i* $\mu_i \in \mathcal{A}$ *con* $\mu_i : Atom(\varphi) \rightarrow \{1, 0\}$ *per cui* $\mu_i(\varphi) = 1$*, allora, per ogni logica* Λ*, risulta* $\vdash_\Lambda \bigvee_i^r \mu_i \leftrightarrow \varphi$*.*

In altre parole, ogni formula è logicamente equivalente alla disgiunzione dei suoi assegnamenti.

Gli assegnamenti costruiti in base alla definizione 2.15 hanno la caratteristica di essere *completi*. Questo significa che attribuiscono un valore di verità a tutti gli atomi della formula anche quando non sia strettamente necessario. Come esempio consideriamo $\varphi = P_0 \vee P_1$ in cui è sufficiente assegnare P_0 oppure P_1 a vero affinché, indipendentemente dal valore non considerato, φ sia vera. Per gestire situazioni di questo tipo sfruttiamo la seguente:

Proprietà 2.2 *Un qualsiasi assegnamento* μ *per* $\varphi \in \mathcal{L}$ *può essere visto come insieme di coppie* (ψ, b) *con* $\psi \in Atom(\varphi)$ *e* $b \in \{0, 1\}$*, dove* $b := 1$ *se e solo se* $\mu(\psi) = 1$ *e* $b := 0$ *se e solo se* $\mu(\psi) = 0$.

Possiamo visualizzare l'insieme descritto nella proprietà 2.2 come segue:

$$\begin{aligned} \mu \quad \equiv \{ \quad & \Box\alpha_1 := 1, \ldots, \Box\alpha_n := 1, \Box\beta_1 := 0, \ldots, \Box\beta_m := 0, \\ & A_1 := 1, \ldots, A_r := 1, A_{r+1} := 0, \ldots, A_s := 0 \quad \} \end{aligned}$$

con la seguente notazione:

- gli atomi modali assegnati a vero sono indicati con $\Box\alpha$ e si intende che l'assegnamento comprende un numero $n \geq 0$ di tali formule;
- gli atomi modali assegnati a falso sono indicati con $\Box\beta$ e si intende che l'assegnamento comprende un numero $m \geq 0$ di tali formule;
- gli atomi proposizionali assegnati sono indicati con la lettera A e si intende che $r \geq 0$ siano assegnati a vero e $s \geq 0$ siano assegnati a falso

Vogliamo ora precisare cosa intendiamo per assegnamento *parziale* e per assegnamento *incompleto* anche se, in quest'ultimo caso, non si tratta di assegnamento nel senso della definizione 2.15. In particolare diciamo che un assegnamento è parziale se consente di valutare la formula a vero anche senza assegnare valori di verità a tutti gli atomi. Ad esempio, nel caso di $\varphi \equiv P_0 \vee P_1$ un possibile assegnamento parziale è $\mu_p \equiv \{P_0 := 1\}$ che può rapidamente essere ricondotto ad uno completo $\mu \equiv \{P_0 := 1, P_1 := 0\}$. Nel seguito, a meno che non sia esplicitamente indicato, non distinguiamo tra assegnamenti parziali e completi. Un assegnamento incompleto è invece tale da non consentire di valutare la formula a vero. Per esempio, se $\varphi = P_0 \wedge P_1$ l'assegnamento $\mu_i \equiv \{P_0 := 1\}$ non è tale da consentirci di valutare φ. Un particolare assegnamento incompleto è l'assegnamento *vuoto* che indicheremo con $\emptyset$.

Nel corso dei capitoli 3 e 4 utilizzeremo un'utile formulazione degli assegnamenti, ricavata sostanzialmente dalla notazione introdotta con la proprietà 2.2. Tale formulazione si basa sulla seguente:

Proprietà 2.3 *Un qualsiasi assegnamento μ per $\varphi \in \mathcal{L}$ può essere visto come una congiunzione $\mu \in \mathcal{L}$:*

$$\mu = \bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$$

con $\gamma = \bigwedge_{k=1}^{r} A_k \ \wedge \ \bigwedge_{h=r+1}^{s} \neg A_h$.

Notiamo che in questa formulazione l'assegnamento vuoto corrisponde alla formula $\mu = \top$. Nel seguito, salvo diversa indicazione, intendiamo con μ un assegnamento scritto come congiunzione di atomi di φ così come definito dalla proprietà 2.3 e utilizziamo la notazione introdotta con 2.2 per denotarne i componenti.

2.4 Sistemi deduttivi modali

In questa sezione esaminiamo dal punto di vista deduttivo la classe di sistemi che in logica modale vengono chiamati *classici*. Prima di entrare nel vivo dell'argomento è necessario soffermarsi su alcune definizioni preliminari Abbiamo già definito nella sezione 2.2 il particolare linguaggio $\mathcal{L}$ che utilizzeremo, dobbiamo ora definire cosa intendiamo per *sistema* di logica modale. Diamo allora la seguente:

Definizione 2.16 (Sistemi di logica modale) *Un insieme di formule $\Sigma \in \mathcal{L}$ è un sistema di logica modale se e solo se è chiuso sotto la seguente regola di inferenza:*

$$\frac{\varphi}{\varphi_1 \dots \varphi_n} RPL$$

dove φ è conseguenza tautologica di $\varphi_1 \dots \varphi_n$.

Questa definizione cattura il fatto, già anticipato nella sezione 2.3.2, che ogni sistema di logica modale è basato sulla logica proposizionale.

Associata alla definizione di un sistema deduttivo si ha sempre la nozione di teorema. Se Σ è un sistema di logica modale (nel seguito Σ ha sempre questo significato)i *teoremi* di Σ sono le formule che ne fanno parte. Formalmente:

Definizione 2.17 (Teoremi nei sistemi di logiche modali) *Data $\varphi \in \mathcal{L}$ diciamo che φ è un teorema nel sistema Σ e lo denotiamo con $\vdash_\Sigma \varphi$ se e solo se $\varphi \in \Sigma$*

Unitamente a quest'ultima la definizione 2.16 ci garantisce che tutte le forme di ragionamento proposizionalmente corrette sono ammesse in un sistema di logica modale. In particolare, l'insieme delle tautologie, che abbiamo indicato con PL (si veda 2.3.2), è alla base di ogni sistema deduttivo modale.

Teorema 2.5 (PL come sistema di logica modale) *Le seguenti proprietà valgono per PL:*

- *è un sistema di logica modale*
- *ogni sistema di logica modale lo include*
- *è il più piccolo sistema di logica modale*

La dimostrazione di questo risultato è molto semplice considerando le definizioni 2.17 e 2.16. Una dimostrazione dettagliata è comunque presente in [Chellas, 1980].

Concludiamo questa fase preliminare con la notazione usata per ideintificare i singoli sistemi deduttivi. Il più piccolo sistema di logica modale in senso classico è chiamato E e per assegnare un nome ai sistemi che prenderemo in esame, usiamo $ES_1 \ldots S_2$. Con questa scrittura intendiamo il sistema di logica modale che risulta dall'aver considerato gli schemi $S_1 \ldots S_n$ come teoremi. Formalizziamo questa notazione nella seguente:

Definizione 2.18 (Sistemi classici) $ES_1 \ldots S_n$ *è il più piccolo sistema classico di logica modale contenente ogni instanza degli schemi* $S_1 \ldots S_n$

L'ordine con cui vengono indicati gli schemi è indifferente. Per esempio potremmo scrivere sia EMC che ECM per denotare la più piccola logica modale classica contenente sia lo schema M che lo schema C (vedi la sezione 2.3.2 per la definizione di M e C).

Siamo ora in grado di descrivere l'intera classe delle logiche modali classiche così come presentata in figura 2.3. Si noti come l'ordine delle logiche esprima, andando dal basso verso l'alto, l'aggiunta di assiomi e il corrispondente aumento di capacità di inferenza epistemica. Al vertice della gerarchia si trova K, la più semplice delle logiche basata sulla semantica alla Hintikka. Nel seguito, per riferirci ad una qualsiasi fra le logiche presentate, useremo la lettera greca Λ.

Cominciamo la nostra presentazione dalla logica E, cioè dalla caratterizzazione di un qualsiasi sistema di logica modale classico, con la seguente:

Definizione 2.19 (Sistemi classici) *Un sistema di logica modale è classico se e solo se è chiuso sotto la seguente regola di inferenza:*

$$\frac{\Box\varphi \leftrightarrow \Box\psi}{\varphi \leftrightarrow \psi} RE$$

Con E intendiamo il più piccolo sistema classico di logica modale.

Definiamo ora il nostro punto di arrivo e cioè la regola di chiusura per i sistemi modali *normali* (i sistemi con semantica a mondi possibili):

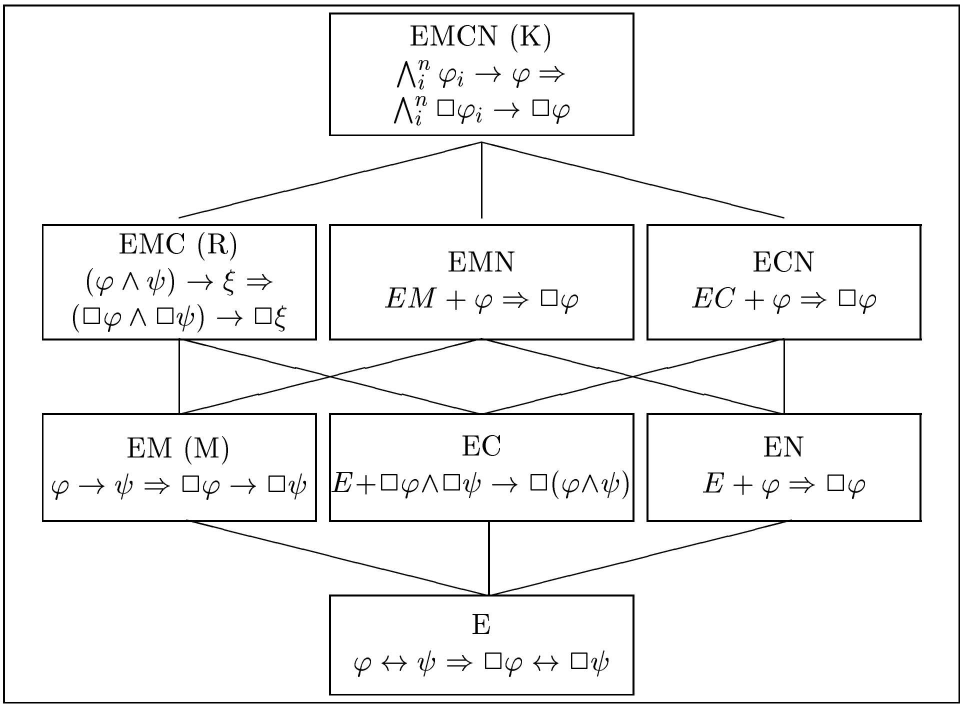

Figura 2.3: Le logiche modali classiche Λ

Definizione 2.20 (sistemi normali) *Un sistema di logica modale è normale se e solo se è chiuso sotto la seguente regola di inferenza:*

$$\frac{\bigwedge_i^n \Box\varphi_i \to \Box\psi}{\bigwedge_i^n \varphi_i \to \psi} RK$$

Con K intenderemo il più piccolo dei sistemi normali definiti in 2.20.

Vogliamo ora investigare la relazione tra il sistema E e il sistema K. Per fare questo diamo le seguenti defnizioni:

Definizione 2.21 (Sistemi monotoni) *Un sistema di logica modale è monotono se e solo se è chiuso sotto la seguente regola di inferenza:*

$$\frac{\Box\varphi \to \Box\psi}{\varphi \to \psi} RM$$

Definizione 2.22 (Sistemi regolari) *Un sistema di logica modale è regolare se e solo se è chiuso sotto la seguente regola di inferenza:*

$$\frac{(\Box\varphi \to \Box\psi) \to \Box\xi}{(\varphi \to \psi) \to \xi} RR$$

In letteratura, il più piccolo sistema monotono è designato con M e il più piccolo sistema regolare con R (vedi [Chellas, 1980]). Volendo utilizzare la nostra nomenclatura, dobbiamo dimostrare che M ed R (e anche K) sono esprimibili a partire da E, aggiungendo come assiomi gli schemi M, C ed N introdotti in 2.3.2.[3]

Il seguente risultato caratterizza la relazione tra modelli classici, monotoni, regolari e normali come una sequenza di logiche modali in cui ciascun sistema include le possibilità di inferenza di quelli che lo precedono. Formalmente:

Teorema 2.6 *Sia Σ un qualsiasi sistema di logica modale.*

- *Se Σ è chiuso sotto RM allora è chiuso sotto RE*
- *Se Σ è chiuso sotto RR allora è chiuso sotto RM*
- *Se Σ è chiuso sotto RK allora è chiuso sotto RR*

La prova di questo risultato è presente in letteratura (si veda, ad esempio, [Chellas, 1980]). Notiamo come la gerarchia presentata in figura 2.3 descriva con immediatezza questa relazione. Esiste infatti un percorso che da E passa attraverso M ed R per giungere a K. La caratterizzazione che abbiamo presentato sino ad ora non ci consente peraltro di indagare gli altri “percorsi” all'interno della gerarchia nè, tantomeno, di collegare l'aggiunta di capacità di inferenza epistemica all'aggiunta di assiomi.

Vediamo ora di giungere ad una caratterizzione più intuitiva dei sistemi classici e che nello stesso tempo catturi tutte le logiche Λ. A tal fine richiamiamo brevemente la definizione degli schemi introdotti nella sezione 2.3.2:

M. $\Box(\varphi \wedge \psi) \rightarrow (\Box\varphi \wedge \Box\psi)$

C. $(\Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$

N. $\Box\top$

e presentiamo (senza dimostrazione) il seguente:

Teorema 2.7 *Sia Σ un sistema di logica modale.*

- *Σ è monotono se e solo se contiene M ed è chiuso sotto RE;*
- *Σ è regolare se e solo se contiene C ed è chiuso sotto RM;*
- *Σ è regolare se e solo se contiene C ed M ed è chiuso sotto RE.*

[3]si noti l'overloading del simbolo M utilizzato per denotare uno schema di formule ed anche il più piccolo sistema monotono. Nel seguito, il compito di effettuare la distinzione, viene lasciato al contesto

Questo risultato (si veda [Chellas, 1980] per la dimostrazione) conferma che, come anticipato in figura 2.3, EM è il più piccolo dei sistemi monotoni e EMC è il più piccolo dei sistemi regolari.

Per completare il discorso è ancora necessario introdurre lo schema N. In particolare, vogliamo dimostrare che il sistema K, così come è stato definito risulta equivalente nella nostra notazione a $EMCN$, cioè al più piccolo sistema regolare di logiche modali a cui venga aggiunto lo schema N. Al riguardo diciamo che vale il seguente:

Teorema 2.8 *Sia Σ un sistema di logica modale. Σ è normale se e solo se contiene N, C, M ed è chiuso sotto RE*

Quanto detto sinora conclude la definizione formale dei sistemi deduttivi di cui ci occuperemo in questa tesi. Il complesso di sistemi che abbiamo ottenuto è rappresentato in figura 2.3. Includendo E, le otto differenti logiche classiche presentate sono ottenute aggiungendo ad E gli schemi M, C, ed N in tutte le possibili combinazioni. Le relazioni di inclusione tra i sistemi sono evidenziate con delle linee che uniscono le logiche più semplici a quelle che le includono (pertanto l'estensione di un sistema è ottenuta andando dall'alto verso il basso).

Concludiamo questa sezione con tre importanti proprietà che, sotto punti di vista diversi, giustificano sia l'introduzione dei sistemi modali classici sia il fatto che le nostre procedure di decisione abbiano senso come tali. In particolare siamo interessati a dimostrare che:

- le logiche presentate in figura 2.3 sono tutte distinte e cioè implementano diversi modelli di ragionamento epistemico. Questa è la motivazione per cui dovremo introdurre, almeno per l'inferenza modale, otto diversi algoritmi SAT based.

- tutti i sistemi di logica modale classica sono dimostrati essere *corretti* e *completi* rispetto a sottoclassi delle strutture $\mathcal{MS}$ introdotte nella sezione 2.3.2. Questo ci assicura che una derivazione in una logica Λ, eccetto K, corrisponde in qualche modo al ragionamento di un agente non onnisciente (aspetto interessante alla luce delle motivazioni in 2.1.

- e infine, ma non meno importante, il fatto che le logiche Λ sono tutte *decidibili*, cioè se una formula è soddisfacibile allora lo è in un modello con un numero di mondi finito. Se così non fosse non saremmo in grado di realizzare procedure di decisione corrette e complete per le logiche classiche.

Di fatto, tutte queste proprietà sono soddisfatte. Valgono infatti i seguenti teoremi:

Teorema 2.9 (Correttezza e completezza) *Sia $S_1 \dots S_n \subseteq \{M, C, N\}$ Allora il sistema di logica modale $ES_1 \dots S_n$ è corretto e completo rispetto alla classe dei modelli finiti $\mathcal{E}_{S_1} \cap \dots \cap \mathcal{E}_{S_n}$.*

Teorema 2.10 *Le logiche* $ES_1 \ldots S_n$ *con* $\{S_1 \ldots S_n\} \subseteq \{M, C, N\}$ *sono tutte distinte*

La prova è presente in letteratura (si veda [Chellas, 1980] per approfondimenti).

Capitolo 3

Algoritmi di decisione SAT-based

> "E in che modo, Socrate, cercherai ciò che non sai assolutamente che cosa sia? Quale delle cose che non sai ti proporrai di cercare? E se per caso ti imbatterai felicemente in essa, come saprai che è ciò che non conoscevi?"
> [Platone, "Dialoghi filosofici"]

Iniziamo in questo capitolo la presentazione della tecnologia SAT-based e degli algoritmi di alto livello che ne fanno uso per risolvere il problema della soddisfacibilità nelle logiche Λ. Alla sezione 3.1 è affidato il compito di descrivere tale tecnologia e di discuterne l'applicabilità alle logiche modali classiche. La sezione 3.2 è dedicata agli algoritmi per le logiche situate nella parte inferiore della gerarchia presentata nella sezione 2.4 e cioè le logiche E, EM, EC. Sulla base di questi algoritmi vengono costruiti nella sezione 3.3 quelli per le restanti logiche. Per ogni algoritmo verrà presentato un risultato di complessità computazionale relativo al problema della soddisfacibilità nella logica considerata.

3.1 Schema di un algoritmo SAT-based

Nelle parole di [Giunchiglia e Sebastiani, 1996b], il termine SAT-based trae origine dall'idea che un decisore per logiche modali o terminologiche possa essere costruito sopra un decisore per SAT. La motivazione teorica di questo risultato è che il ragionamento in logiche modali può essere visto come una appropriata composizione di fasi di ragionamento proposizionale all'interno di teorie distinte (si vedano, ad esempio, [Giunchiglia e Serafini, 1994] per le logiche normali e [Giunchiglia *et al.*, 1993] per alcune logiche classiche).

Lo schema che intendiamo tracciare in questa sezione prevede una automatizzazione completa del ragionamento, sia per quanto riguarda la parte proposizionale, sia per quanto riguarda l'inferenza modale. L'approccio che seguiamo nel valutare la soddisfacibilità di una formula è di tipo *top-down*, cioè sfrutta una discesa ricorsiva nei diversi livelli modali della formula (operatori "$\Box$" annidati).L'alternanza tra i due tipi di ragionamento è espressa dal fatto che, nel generico livello modale, il decisore SAT considera atomiche le sottoformule modali. Ad esempio la formula $\varphi = \neg P_0 \wedge \Box P_1$ è soddisfatta con l'assegnamento $\mu \equiv \{\Box P_1 := 1, P_0 := 0\}$ in cui la sottoformula $\psi = \Box P_1$ è stata considerata alla stessa stregua di P_0. Il compito dell'inferenza modale è quello di estrarre dall'eventuale assegnamento un test di consistenza nella particolare struttura epistemica. Questo test (ricorsivo) ha profondità modale inferiore rispetto alla sottoformula di partenza, per cui le due fasi di ragionamento si alternano sino a quando non si arriva a formule puramente proposizionali.

Cerchiamo di formalizzare con più precisione quanto sinora detto circa la strategia di decisione top-down. Richiamiamo brevemente la definizione 2.7 data nella sezione 2.2 relativa all'insieme di atomi di una generica formula φ. Ricordiamo che tale insieme comprendeva tutti le foglie dell'albero sintattico di φ ottenuto considerando atomiche le sottoformule modali. Ricordiamo inoltre che, in base alla proprietà 2.3, un assegnamento per φ può essere visto sia come un insieme di coppie (ψ, b) dove $\psi \in Atom(\varphi)$ e $b \in \{1, 0\}$, sia come una formula $\mu \in \mathcal{L}$. Il problema di valutare la soddisfacibilità nella logica Λ di una generica formula modale φ può essere scomposto nei due passi seguenti:

1. trovare un assegnamento μ per φ rappresentato come:

$$\mu = \bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$$

   e se tale assegnamento non esiste, allora φ è insoddisfacibile in Λ.

2. controllare che μ sia consistente nella particolare logica modale considerata. Questo comporta un test ricorsivo sulle sottoformule α e β. Se il test fallisce μ non è consistente in Λ.

Si noti la diversa terminologia utilizzata nel caso del ragionamento modale rispetto al caso proposizionale. Diciamo infatti che un assegnamento è *inconsistente* se viola una particolare proprietà epistemica dei modelli di Λ e quindi parliamo di test di *consistenza* per l'assegnamento. Una formula φ è *insoddisfacibile* in Λ quando non esiste un assegnamento, oppure quando tutti gli assegnamenti sono inconsistenti in Λ. Nel seguito, riferendoci agli algoritmi e alle procedure presentati, usiamo sempre questa terminologia.

Un'ultima nota riguardante il procedimento sopra descritto concerne la proprietà della terminazione. Per tutte le logiche prese in esame, l'algoritmo top-down è garantito terminare in un tempo finito perché il numero degli assegnamenti considerati è, esso stesso, finito e ogni applicazione del passo 2 determina una diminuzione della profondità modale delle formule valutate:

quando la formula è puramente proposizionale l'algoritmo termina con il passo 1.

Nelle seguenti sezioni esaminiamo la parte inerente al passo 2. Sotto forma di equivalenze all'interno delle singole logiche vediamo come sia possibile effettuare un controllo di consistenza modale sull'assegnamento μ. Per semplicità di notazione, per indicare che φ è soddisfacibile in Λ, usiamo $\Lambda\textsc{sat}(\varphi)$ al posto di $\not\vdash_\Lambda \neg\varphi$.

3.2 Gli algoritmi di base

In questa sezione prendiamo in esame gli algoritmi relativi alle logiche di base E, EM, EC. Quello che rende questi algoritmi particolarmente importanti è che la loro struttura fornisce la base per lo sviluppo degli algoritmi relativi a tutte le altre logiche modali classiche.

Prima di entrare nel vivo del'argomento enunciamo una proprietà utile per le dimostrazioni presentate nel seguito. Tale proprietà stabilisce una relazione tra la parte puramente modale di un assegnamento e le condizioni che l'assegnamento epistemico di un mondo deve soddisfare affinché $\mu = \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j$ possa dirsi consistente nel mondo considerato.

Proprietà 3.1 *Dato un modello* $M = \langle W, N, P\rangle \in \mathcal{E}$, *se un mondo* $w \in W$ *è tale che* $M, w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j$ *allora, per ogni* i *e* j, $\|\alpha_i\|^M \in N(w)$ *e* $\|\beta_j\|^M \notin N(w)$)

Dimostrazione - Partendo dal fatto che $M, w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j$ per interpretazione di "$\wedge$" si ricava che $M, w \models \Box\alpha_i$ e $M, w \models \neg\Box\beta_j$ per ogni i e j. D'altronde per interpretazione di "$\Box$" si ha che $\Box\alpha_i$ è vero in w sse $\|\alpha_i\|^M \in N(w)$ mentre $\neg\Box\beta_j$ è vero in w sse $\|\beta_j\|^M \notin N(w)$ da cui la tesi.

3.2.1 Logica E

Iniziamo la discussione riguardante la logica modale E introducendo due proprietà riguardanti le strutture $\mathcal{E}$. La prima proprietà sancisce il fatto che, all'interno di un generico modello, l'esistenza di un mondo w in cui $N(w)$ contiene l'intensione di φ ma non l'intensione di ψ, è sufficiente a concludere che nel modello considerato le intensioni delle formule φ e ψ differiscono per almeno un elemento.

Proprietà 3.2 *Dati* $M = \langle W, N, P\rangle \in \mathcal{E}$ *e* $\varphi, \psi \in \mathcal{L}$, *se un mondo* $w \in W$ *è tale che* $\|\varphi\|^M \in N(w)$ *e* $\|\psi\|^M \notin N(w)$ *allora* $\|\varphi\|^M \not\equiv \|\psi\|^M$.

Dimostrazione - Per assurdo, se fosse $\|\varphi\|^M \equiv \|\psi\|^M$ allora avremmo che $\|\varphi\|^M \in N(w)$ e quindi anche $\|\psi\|^M \equiv \|\varphi\|^M \in N(w)$.

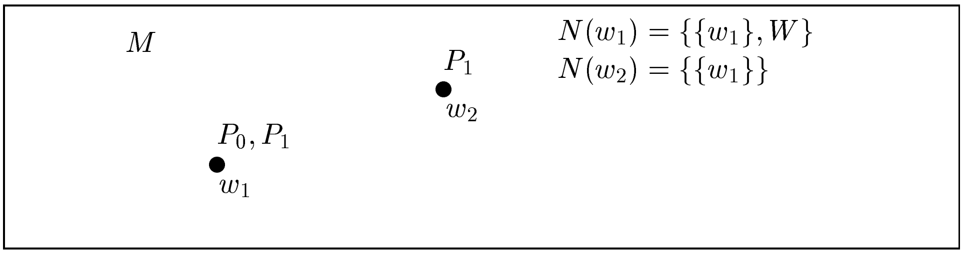

Figura 3.1: Una struttura epistemica $M \in \mathcal{E}$

Per comprendere meglio la proprietà 3.2 consideriamo il seguente esempio. Sia $M = \langle W, N, P\rangle \in \mathcal{E}$ la struttura rappresentata in figura 3.1. Consideriamo le formule $\varphi = P_0$ e $\psi = P_1$. In w_2 si ha $M, w_2 \models \Box P_0 \wedge \neg\Box P_1$ perché $N(w_2)$ contiene $\{w_1\} \equiv \|P_0\|^M$ ma non $W \equiv \|P_1\|^M$. Questo implica l'esistenza di un mondo (in questo caso proprio w_2) che appartiene a $\|P_1\|^M$ ma non a $\|P_0\|^M$ e che ci garantisce che le due intensioni non coincidano. Se fosse stato, ad esempio $\|P_0\|^M \equiv W$, non sarebbe stato possibile "costruire" $N(w_2)$ in maniera tale che $\|P_1\|^M \not\in N(w_2)$.

Consideriamo ora la proprietà che mette in relazione la diversità fra le intensioni di due formule in un modello con la soddisfacibilità di una particolare formula all'interno dello stesso modello.

Proprietà 3.3 *Date $\varphi, \psi \in \mathcal{L}$, se $M = \langle W, N, P\rangle \in \mathcal{E}$ è tale che $\|\varphi\|^M \not\equiv \|\psi\|^M$ allora esiste un mondo $w \in W$ tale che $M, w \models \varphi \leftrightarrow \neg\psi$*

Dimostrazione - Per assurdo. Se non esiste w tale che $M, w \models \varphi \leftrightarrow \neg\psi$, allora significa che $M \models \varphi \leftrightarrow \psi$ e quindi, per definizione di intensione di verità, $\|\varphi\|^M \equiv \|\psi\|^M$ da cui l'assurdo.

Rifacendoci ancora alla struttura presentata nella figura 3.1, consideriamo sempre le formule $\varphi = P_0$ e $\psi = P_1$. Come avevamo visto in precedenza, il modello è tale che $\|P_0\|^M$ non coincide con $\|P_1\|^M$ e pertanto, in base alla proprietà 3.3, ci aspettiamo di trovare in M un mondo che soddisfi la negazione della coimplicazione tra P_0 e P_1. In effetti questo mondo esiste ed è w_2 in quanto $M, w_2 \models P_1 \wedge \neg P_0$. Si noti che allorquando avessimo $M \models P_0 \leftrightarrow P_1$, non sarebbe più possibile l'esistenza di un mondo come w_2 e quindi le intensioni di P_0 e P_1 coinciderebbero.

Formalizziamo l'algoritmo SAT-based per E con il seguente:

Lemma 3.1 (E-Lemma) *In E si ha la seguente equivalenza:*

$$\textsc{Esat}(\textstyle\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma)$$
$$\textit{se e solo se}$$
$$\textit{per ogni } i,j.\ \textsc{Esat}(\alpha_i \leftrightarrow \neg\beta_j) \quad e \quad \textsc{Esat}(\gamma)$$

Per la dimostrazione che segue è conveniente riscrivere il lemma nella seguente forma:

$$\begin{array}{l} \text{esiste } M.\text{ esiste } w.\ M,w \models (\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma) \\ \qquad\qquad\qquad \text{se e solo se} \\ \text{per ogni } i,j.\text{ esiste } M_{ij}.\text{ esiste } w_{ij}.\ M_{ij},w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j \quad \text{e} \\ \text{esiste } M.\text{ esiste } w.\ M,w \models \gamma \end{array} \tag{3.1}$$

dove $M = \langle W, N, P\rangle \in \mathcal{E}$.
Dimostrazione "$\Rightarrow$" - Partendo da:

$$\text{esiste } M.\text{ esiste } w.\ M,w \models (\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma) \tag{3.2}$$

cerchiamo di far vedere che riusciamo a trovare i modelli $M_{ij} \in \mathcal{E}$ e, all'interno di questi, i mondi w_{ij} tali che $M_{ij}, w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j$ per ogni i e j. Per interpretazione di "$\wedge$" otteniamo:

$$\begin{array}{l} \text{esiste } M.\text{ esiste } w.\ M,w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \quad \text{e} \\ \text{esiste } M.\text{ esiste } w.\ M,w \models \gamma \end{array} \tag{3.3}$$

Da questo primo risultato è evidente che non si da il caso per cui γ non sia soddisfacibile in $\mathcal{E}$.
Ora, siano M e w il particolare modello e il particolare mondo per i quali $M, w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j$.
Applicando la proprietà 3.1 posso concludere che, comunque presi i e j, risulta $\|\alpha_i\|^M \in N(w)$ e $\|\beta_j\|^M \notin N(w)$. Prendiamo ora due generici α_i e β_j per cui sia valida la condizione precedente.
Possiamo applicare la proprietà 3.2 per concludere che in M risulta $\|\alpha_i\|^M \not\equiv \|\beta_j\|^M$ e quindi, utilizzando la proprietà 3.3, abbiamo:

$$\text{esiste } M_{ij}.\text{ esiste } w_{ij}.\ M_{ij},w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j \tag{3.4}$$

Dato che tale risultato non dipende dagli α_i e β_j considerati, concludiamo che:

$$\text{per ogni } i,j.\text{ esiste } M_{ij}.\text{ esiste } w_{ij}.\ M_{ij},w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j \tag{3.5}$$

Dimostrazione "$\Leftarrow$" - Partendo dall'ipotesi:

$$\text{per ogni } i,j.\text{ esiste } M_{ij}.\text{ esiste } w_{ij}.\ M_{ij},w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j \tag{3.6}$$

cerchiamo di dimostrare che esistono almeno un modello $M \equiv \langle W, N, P\rangle \in \mathcal{E}$ e un mondo $w \in W$ tali che $M, w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma$.

Per ipotesi posso pensare di avere $m \times n$ modelli distinti, cioè tali che $\bigcap_{ij} W_{ij} = \emptyset$, rappresentati in figura 3.2.

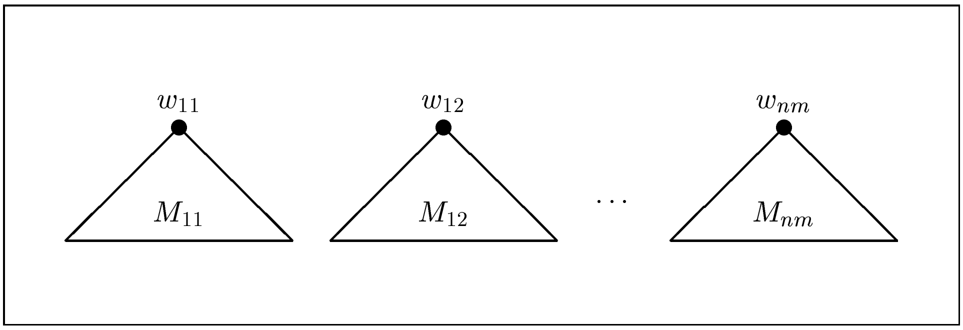

Figura 3.2: Insieme dei modelli $M_{ij}, w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j$

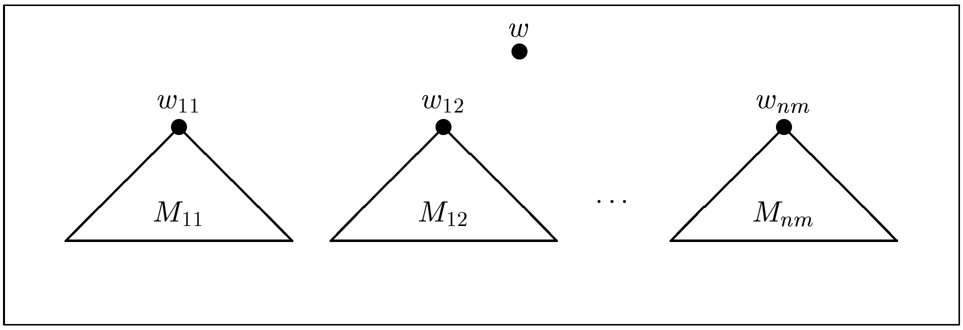

Figura 3.3: Modello $M, w \models \bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$

I modelli in figura 3.2 dovranno essere tali da rispettare le seguenti condizioni:

$$\begin{array}{r}
M_{11}, w_{11} \models \alpha_1 \leftrightarrow \neg\beta_1 \\
M_{12}, w_{12} \models \alpha_1 \leftrightarrow \neg\beta_2 \\
\dots \\
M_{nm}, w_{nm} \models \alpha_n \leftrightarrow \neg\beta_m
\end{array} \tag{3.7}$$

per cui, comunque presi i e j, vale la proprietà che $\|\alpha_i\|^{M_{ij}} \neq \|\beta_j\|^{M_{ij}}$ in quanto c'è almeno un mondo (w_{ij}) in cui o vale α_i o vale β_j ma non entrambe.

Sulla base dei modelli M_{ij} costruisco il modello M presentato nella figura 3.3. L'idea è , ovviamente, quella di costruire $M = \langle W, N, P\rangle$ come unione di tutti gli M_{ij} e di un mondo tale che $M, w \models \bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$ con $w \notin \bigcup_{ij} W_{ij}$.

La parte difficile nella costruzione di M è la definizione dell'assegnamento epistemico N. È infatti necessario che i valori di verità delle formule, almeno per quanto concerne le sottoformule di $\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$, vengano preservati in M.

Per quanto riguarda W, è sufficiente prendere $W \equiv \bigcup_{ij} W_{ij} \cup w$. Definiamo ora $P_w : \mathbf{N} \to \mathcal{P}(W)$ come:

$$P_w(n) = \begin{cases} \{w\} & \text{se } P_n \text{ occorre positivo in } \gamma \\ \emptyset & \text{altrimenti} \end{cases} \tag{3.8}$$

Componiamo le $P_{ij} : \mathbf{N} \to \mathcal{P}(W_{ij})$ in modo che per ogni $n \in \mathbf{N}$ risulti:

$$P(n) \equiv \bigcup_{ij} P_{ij}(n) \cup P_w(n) \tag{3.9}$$

Costruiamo N come segue. Da ora in poi sia $\varphi = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$. Per ogni $\psi \in sub(\varphi)$ definiamo $sub_k(\varphi)$, l'insieme delle sottoformule di φ aventi profondità modale k, dicendo che $\psi \in sub_k(\varphi)$ se e solo se $depth(\psi) = k$. Naturalmente risulta $sub(\varphi) \equiv \bigcup_k sub_k(\varphi)$.

Definiamo le intensioni per le sottoformule di φ aventi profondità k e cioè $\|\psi\|_k^M : sub_k(\varphi) \to \mathcal{P}(W)$, come segue:

- per ogni $n \in \mathbf{N}$, se $P_n \in sub_0(\varphi)$ risulta $\|P_n\|_0^M \equiv P(n)$;
- se $\psi = \neg\xi$ e $\psi \in sub_k(\varphi)$, allora risulta $\|\psi\|_k^M \equiv (W - \|\xi\|_k^M)$;
- se $\psi = \xi_1 \wedge \xi_2$ e $\xi_1, \xi_2 \in sub_k(\varphi)$, allora risulta $\|\psi\|_k^M \equiv \|\xi_1\|_k^M \cap \|\xi_2\|_k^M$;
- se $\psi = \Box\xi$ e $\psi \in sub_k(\varphi)$ allora:
  - per tutti gli $u \in \bigcup_{ij} W_{ij}$, se $u \in W_{ij}$ mettiamo $U \subseteq W$ in $N_k(u)$ se e solo se $\|\xi\|^{M_{ij}} \in N_{ij}(u)$ e $U \equiv \|\xi\|_{k-1}^M$;
  - mettiamo $U \subseteq W$ in $N_k(w)$ se e solo se per qualche α_i risulta $\xi = \alpha_i$ e $U \equiv \|\xi\|_{k-1}^M$.

Se poniamo $N \equiv \bigcup_k N_k$ allora per ogni $\psi \in sub(\varphi)$:

$$\|\psi\|^M \equiv \bigcup_k \|\psi\|_k^M \tag{3.10}$$

Per come abbiamo costruito M, risulta chiaramente $M, w \models \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ e questo è possibile perché, vista l'equazione 3.10, risulta che $\|\alpha_i\|^M \not\equiv \|\beta_j\|^M$ per ogni i e j.

Concludiamo questa sezione con il risultato relativo alla complessità del test di soddisfacibilità per la logica E. Verificare se una generica formula modale φ è soddisfacibile in una struttura $\mathcal{E}$, è un problema risolvibile in tempo polinomiale con un algoritmo non deterministico e quindi le procedure che implementano l'algoritmo 3.1 possono, al meglio, essere NP-hard. Il motivo di quest'ultima limitazione è immediato, essendo le logiche modali classiche un'estensione consistente della logica proposizionale (vedi definizione 2.16).

Più in particolare il problema della soddisfacibilità in $\mathcal{E}$ è riconducibile in tempo polinomiale a SAT (NP-completeness). L'intuizione dietro questo risultato è il fatto che, data una qualsiasi formula modale φ e considerato l'insieme delle possibili sottoformule modali $sub_{\Box}(\varphi)$ (vedi definizione 2.6 nella sezione 2.2), il numero di possibili test ricorsivi tra coppie di formule modali è limitato superiormente da $\mid sub_{\Box}(\varphi) \mid^2$. Questo significa che, nel caso pessimo, si esegue un numero polinomiale di test NP e quindi l'algoritmo può essere ricondotto in tempo polinomiale a SAT. Presentiamo, senza dimostrazione, il seguente:

Teorema 3.1 *Il problema della soddisfacibilità rispetto alle strutture $\mathcal{E}$ è NP-complete.*

La prova di questo risultato è presente in letteratura. Si veda ad esempio [Vardi, 1989] per la presentazione di un possibile algoritmo per il test di soddisfacibilità in E e la relativa prova del teorema 3.1. Una dimostrazione più completa si trova in [Fagin *et al.*, 1995] dove, tra l'altro, si afferma che se una formula è soddisfacibile in $\mathcal{E}$, allora è soddisfacibile in un modello con al più un numero polinomiale di mondi rispetto alle dimensioni della formula.

3.2.2 Logica EM

Adattiamo la proprietà 3.2 per tenere conto dei vincoli addizionali imposti sull'assegnamento epistemico in $\mathcal{E}_M$. Come avevamo visto nella sottosezione 2.3.2 tali modelli si dicono supplementati, nel senso che la funzione N risulta chiusa per sovrainsiemi (se $U \in N(w)$ allora per qualsiasi $V \supseteq U$ si ha che $V \in N(w)$). Teniamo conto di questo fatto con la seguente:

Proprietà 3.4 *Dati $M = \langle W, N, P\rangle \in \mathcal{E}_M$ e $\varphi, \psi \in \mathcal{L}$, se un mondo $w \in W$ è tale che $\|\varphi\|^M \in N(w)$ e $\|\psi\|^M \notin N(w)$ allora $\|\varphi\|^M \not\subseteq \|\psi\|^M$.*

Dimostrazione - Per assurdo, assumiamo che valga $\|\varphi\|^M \in N(w)$ e $\|\psi\|^M \notin N(w)$. Consideriamo solo il caso relativo all'inclusione stretta $\|\varphi\|^M \subset \|\psi\|^M$ mentre per l'uguaglianza rinviamo alla dimostrazione della proprietà 3.2. Sfruttiamo la proprietà dei modelli in $\mathcal{E}_M$ per cui se $U \in N(w)$ allora qualsiasi $V \supseteq U$ è tale che $V \in N(w)$. Risulta allora $\|\varphi\|^M \in N(w)$ e $\|\psi\|^M \in N(w)$ da cui l'assurdo.

Per spiegare la proprietà 3.4 utilizziamo l'esempio di figura 3.4. Supponiamo di sapere che $M = \langle W, N, P\rangle \in \mathcal{E}_M$ consta di tre mondi w_1, w_2 e w_3 per i quali l'assegnamento intensionale è quello indicato in figura ma di cui supponiamo di non conoscere l'assegnamento epistemico. Ci chiediamo allora se sia possibile costruire N di modo che in uno dei tre mondi sia soddisfatta la formula $\varphi = \Box P_0 \wedge \neg\Box P_1$. A tal fine poniamo $\{w_1\} \equiv \|P_0\|^M$ in $N(w_1)$. Questo significa che data la proprietà delle strutture $\mathcal{E}_M$ anche

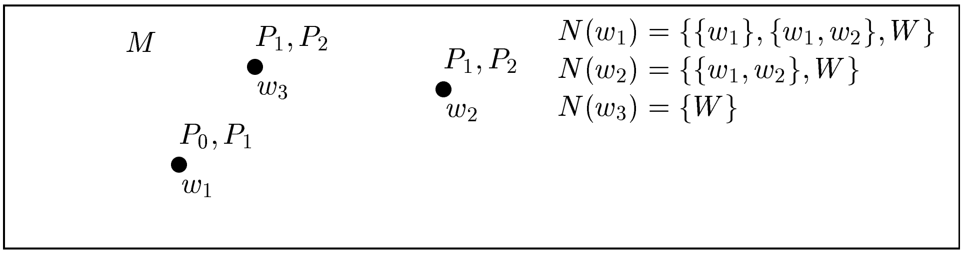

Figura 3.4: Una struttura epistemica in $\mathcal{E}_M$

$W = \|P_1\|^M$ appartiene ad $N(w_1)$ e quindi $M, w_1 \not\models \varphi$. In base alla proprietà 3.4 dovevamo aspettarci questo risultato perché $\|P_0\|^M \subset \|P_1\|^M$. In particolare si noti che $M \not\models \varphi$ perché, data la particolare funzione P in M, non è mai possibile includere l'intensione di P_0 in qualche $N(w)$ senza includere anche l'intensione di P_1.

Colleghiamo ora le intensioni di due generiche formule con la soddisfacibilità di una particolare formula:

Proprietà 3.5 *Date $\varphi, \psi \in \mathcal{L}$ se $M = \langle W, N, P \rangle \in \mathcal{E}$ è tale che $\|\varphi\|^M \not\subseteq \|\psi\|^M$ allora esiste un mondo $w \in W$ tale che $M, w \models \varphi \wedge \neg\psi$*

Dimostrazione - Per assurdo, suppponiamo che valga $\|\varphi\|^M \not\subseteq \|\psi\|^M$. Se non esiste w tale che $M, w \models \varphi \wedge \neg\psi$, allora significa che $M \models \varphi \rightarrow \psi$ e quindi in utti i mondi dove è vero φ è vero anche ψ da cui, per definizione di insieme di verità, $\|\varphi\|^M \subseteq \|\psi\|^M$ che è assurdo.

Ritornando all'esempio 3.4 consideriamo una coppia di formule per cui valga l'ipotesi della proprietà 3.5. Notiamo che $\|P_2\|^M \not\subseteq \|P_0\|^M$ ed anche che $\|P_1\|^M \not\subseteq \|P_2\|^M$. Ci aspettiamo che esista almeno un mondo in cui è soddisfacibile $\psi_1 = P_2 \wedge \neg P_0$ e infatti risulta $M, w_3 \models \psi_1$ ed anche $M, w_2 \models \psi_1$; ci aspettiamo anche un mondo in cui sia soddisfacibile $\psi_2 = P_1 \wedge \neg P_2$ e infatti risulta $M, w_1 \models \psi_2$.

Nell'esempio appena considerato, sarebbe stato impossibile soddisfare formule del tipo $\Box P_0 \wedge \Box P_2 \wedge \neg\Box P_1$ e questo a causa del fatto che $M \models P_0 \rightarrow P_1$ e $M \models P_2 \rightarrow P_1$. Vogliamo ora formalizzare questa proprietà con il seguente:

Lemma 3.2 (EM-Lemma) *In EM si ha la seguente equivalenza:*

$$\begin{gathered}\text{EM}\textsc{sat}(\textstyle\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma) \\ \textit{se e solo se} \\ \textit{per ogni } i,j.\, \text{EM}\textsc{sat}(\alpha_i \wedge \neg\beta_j) \quad \textit{e} \quad \text{EM}\textsc{sat}(\gamma)\end{gathered}$$

Riscriviamo il lemma nella seguente forma:

$$\begin{array}{l}\text{esiste } M.\text{ esiste } w.\; M, w \models (\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma) \\ \qquad\qquad\text{se e solo se} \\ \text{per ogni } i,j.\text{ esiste } M_{ij}.\text{ esiste } w_{ij}.\; M_{ij}, w_{ij} \models \alpha_i \wedge \neg\beta_j \quad \text{e} \\ \text{esiste } M.\text{ esiste } w.\; M, w \models \gamma\end{array} \tag{3.11}$$

dove $M = \langle W, N, P\rangle \in \mathcal{E}_M$.
Dimostrazione "$\Rightarrow$" - La dimostrazione è uguale a quella per E (si veda il lemma 3.1 della sottosezione precedente) sino al punto in cui si considerano il modello M e il mondo w per i quali $M, w \models \bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j$.
Anche in questo caso applichiamo la proprietà 3.1 per concludere che, comunque presi i e j, risulta $\|\alpha_i\|^M \in N(w)$ e $\|\beta_j\|^M \notin N(w)$. Prendiamo ora due generici α_i e β_j per cui sia valida la condizione precedente. Dato che $M \in \mathcal{E}_M$ possiamo applicare la proprietà 3.4 per concludere che $\|\alpha_i\|^M \not\subseteq \|\beta_j\|^M$ e quindi, utilizzando la proprietà 3.5, possiamo ridurci al test di soddisfacibilità:

$$\text{esiste } M_{ij}.\, \text{esiste } w_{ij}.\, M, w_{ij} \models \alpha_i \wedge \neg\beta_j \tag{3.12}$$

da cui, con argomentazioni simili a quelle del lemma 3.1, segue la tesi.

Dimostrazione "$\Leftarrow$" - Si ottiene analogamente a quella per E, a partire da $m \times n$ modelli $M_{ij} \in \mathcal{E}_M$, ciascuno con un mondo $w_{ij} \in W_{ij}$ tale che $M_{ij} w_{ij} \models \alpha_i \wedge \neg\beta_j$. Si costruisce un modello M con la stessa tecnica usata per E e si dimostra che $M \in \mathcal{E}_M$ da cui la tesi.

Il problema della soddisfacibilità per EM, così come quello per E, è riconducibile in tempo polinomiale a SAT. Questo è evidente se si considera che i test di soddisfacibilità in cui viene scomposto il test di consistenza per l'assegnamento coinvolgono sempre coppie di formule. Varrà allora il limite superiore di $\mid sub_{\Box}(\varphi) \mid^2$ possibili test ricorsivi e quindi il seguente risultato:

Teorema 3.2 *Il problema della soddisfacibilità rispetto alle strutture $\mathcal{E}_M$ è NP-complete.*

La prova è banale, una volta che viene dato per scontato il teorema 3.1. In particolare, in [Vardi, 1989] tale risultato è presente, senza dimostrazione, come corollario del corrispondente teorema per E. Sempre per analogia con tale logica, risulta anche che in EM se una formula è soddisfacibile in $\mathcal{E}_M$, allora è soddisfacibile in un modello con al più un numero polinomiale di mondi rispetto alle dimensioni della formula.

3.2.3 Logica EC

Incominciamo adattando la proprietà 3.2 alla particolare struttura dei modelli in $\mathcal{E}_C$. In via preliminare, abbiamo bisogno della seguente:

Definizione 3.1 (Chiusura per intersezioni) *Dati gli insiemi $s_1 \dots s_k$ e l'insieme $S = \{s_1 \dots s_k\}$ definiamo $\bigcap^* S$ l'insieme costruito come segue:*

- $s_i \in \bigcap^* S$ *per ogni i, $1 \leq i \leq k$;*

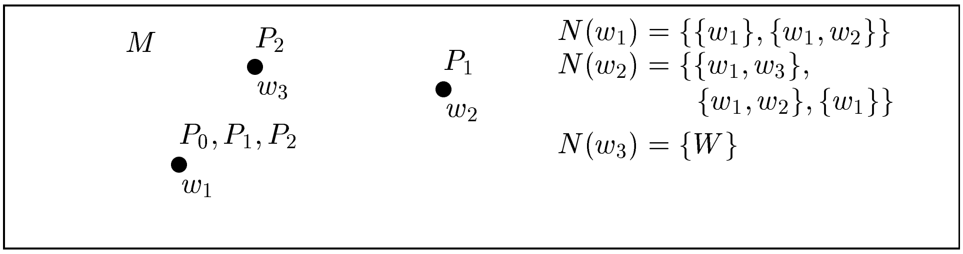

Figura 3.5: Una struttura epistemica in $\mathcal{E}_C$

- *se* $s, t \in \bigcap^* S$ *allora anche* $s \cap t \in \bigcap^* S$.

Ricordiamo che gli assegnamenti epistemici dei modelli in $\mathcal{E}_C$ (si veda la sezione 2.3.2) si dicono chiusi per intersezione in quanto la funzione N è tale che se $U, V \in N(w)$ allora anche $U \cap V \in N(w)$ per ogni $w \in W$. Quindi vale la seguente:

Proprietà 3.6 *Dati* $M = \langle W, N, P \rangle \in \mathcal{E}_C$ *e* $\varphi_1 \ldots \varphi_k, \psi \in \mathcal{L}$*, se un mondo* $w \in W$ *è tale che* $\|\varphi_i\|^M \in N(w)$ *per ogni* i $(1 \leq i \leq k)$ *e* $\|\psi\|^M \notin N(w)$ *allora* $\|\psi\|^M \notin \bigcap^*\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$.

Dimostrazione - Per assurdo, assumiamo che valga $\|\varphi_i\|^M \in N(w)$ per ogni φ_i e $\|\psi\|^M \notin N(w)$ anche con $\|\psi\|^M \in \bigcap^*\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$. Sfruttiamo la proprietà dei modelli in $\mathcal{E}_C$ per cui se $U, V \in N(w)$ allora anche $V \cap U \in N(w)$. Risulta allora $N(w) \supseteq \bigcap^*\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$ da cui $\|\psi\|^M \in N(w)$ e l'assurdo.

Il significato intuitivo della proprietà 3.6 può essere meglio compreso considerando l'esempio di figura 3.5. Supponiamo di non conoscere l'assegnamento epistemico e di voler costruire un $N(w)$ tale che valga $M, w \models \Box P_1 \wedge \Box P_2 \wedge \neg\Box P_0$. Cominciamo con il porre $N(w) \equiv \{\|P_1\|^M, \|P_2\|^M\}$ e vediamo che $N(w) \not\equiv \bigcap^* N(w)$. Per soddisfare questa condizione dobbiamo aggiungere a $N(w)$ l'insieme $\|P_1\|^M \cap \|P_2\|^M = \{w_1\}$. D'altro canto, visto che $\|P_0\|^M \equiv \{w_1\}$, risulta $M, w \not\models \Box P_1 \wedge \Box P_2 \wedge \neg\Box P_0$ per qualsiasi $w \in W$ e questo perchè (proprietà 3.6) risulta proprio $\|P_0\|^M \in \bigcap^*\{\|P_1\|^M \|P_2\|^M\}$.

Prima di collegare la condizione $\|\psi\|^M \notin \bigcap^*\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$ sulle intensioni ad una corrispondente proprietà sulla soddisfacibilità di particolari formule è necessario raffinarla ulteriormente con la seguente:

Proprietà 3.7 *Date* $\varphi_1 \ldots \varphi_k, \psi \in \mathcal{L}$ *in* $M = \langle W, N, P \rangle \in \mathcal{E}_C$ *si ha* $\|\psi\|^M \notin \bigcap^*\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$ *se e solo se, dato l'insieme* $\Delta \equiv \{\varphi_i \mid \|\varphi_i\|^M \supseteq \|\psi\|^M\}$*, vale* $\Delta \equiv \emptyset$ *oppure* $\|\psi\|^M \neq \bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M$.

Dimostrazione "$\Rightarrow$" - Per assurdo, se $\Delta \not\equiv \emptyset$ e $\|\psi\|^M \equiv \bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M$ dato che (vedi definizione 3.1):

$$\bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M \in \bigcap^{*}\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\} \tag{3.13}$$

allora anche $\|\psi\|^M \in \bigcap^*\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\}$.

Dimostrazione "$\Leftarrow$" - Si danno due casi. Se $\Delta \equiv \emptyset$ significa che $\|\psi\|^M \not\subseteq \|\varphi_i\|^M$ per qualsiasi φ_i e quindi $\|\psi\|^M$ non può far parte di alcuna intersezione finita dei $\|\varphi_i\|^M$. Se $\Delta \not\equiv \emptyset$ ma $\|\psi\|^M \not\equiv \bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M$ allora esiste $w \in W$ tale che $w \in \|\varphi_i\|^M$ per qualsiasi φ_i in Δ e tale che $w \notin \|\psi\|^M$. Infatti dati $U_1, U_2 \in \bigcap^*\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\}$ se $U_1, U_2 \in \{\|\varphi_i\|^M \mid \varphi_i \in \Delta\}$ allora esiste un mondo w tale che $w \in U_1 \cap U_2$ e $w \notin \|\psi\|^M$. Se U_1 o U_2 non appartengono a $\{\|\varphi_i\|^M \mid \varphi_i \in \Delta\}$ allora $U_1 \cap U_2 \not\subseteq \|\psi\|^M$. Allora posso concludere che $\|\psi\|^M \not\equiv U$ per qualsiasi $U \in \bigcap^*\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\}$.

Ora possiamo trasformare le condizioni sulle intensioni in condizioni sulla soddisfacibilità di particolari formule. In particolare, valgono le seguenti proprietà:

Proprietà 3.8 *Siano $\varphi, \psi \in \mathcal{L}$ e $M = \langle W, N, P\rangle \in \mathcal{E}_C$. Se $\|\psi\|^M \not\subseteq \|\varphi\|^M$ allora esiste un $w \in W$ tale che $M, w \models \neg\varphi \wedge \psi$.*

Dimostrazione - Per assurdo, se non esiste un mondo $w \in W$ tale che $M, w \models \neg\varphi \wedge \psi$ allora $M \models \psi \rightarrow \varphi$ da cui, per definizione di intensione, $\|\psi\|^M \subseteq \|\varphi\|^M$ e l'assurdo.

Proprietà 3.9 *Siano $\varphi_1 \dots \varphi_k, \psi \in \mathcal{L}$, $M = \langle W, N, P\rangle \in \mathcal{E}_C$ e Δ un insieme non vuoto tale che $\Delta \subseteq \{\varphi_1 \dots \varphi_k\}$ e $\Delta \equiv \{\varphi_i \mid \|\varphi_i\|^M \supseteq \|\psi\|^M\}$. Se $\|\psi\|^M \not\equiv \bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M$ allora esiste un mondo $w \in W$ tale che $M, w \models \bigwedge_{\varphi_i \in \Delta} \varphi_i \wedge \neg\psi$.*

Dimostrazione - Per assurdo, se non esiste un mondo tale che $M, w \models \bigwedge_{\varphi_i \in \Delta} \varphi_i \wedge \neg\psi$ allora $M \models \bigwedge_{\varphi_i \in \Delta} \varphi_i \rightarrow \psi$ e quindi, per definizione di intensione, $\| \bigwedge_{\varphi_i \in \Delta} \varphi_i\|^M \subseteq \|\psi\|^M$ da cui risulta, per interpretazione di "$\wedge$" e definizione di intensione, $\bigcap_{\varphi_i \in \Delta} \|\varphi_i\|^M \subseteq \|\psi\|^M$. Dato che $\|\varphi_i\|^M \supseteq \|\psi\|^M$ per ogni $\varphi_i \in \Delta$, possiamo concludere che $\bigcup_{\varphi_i \in \Delta} \|\varphi_i\|^M \equiv \|\psi\|^M$ da cui l'assurdo.

Considerando tutte le proprietà esposte diciamo che vale il seguente:

Lemma 3.3 (EC-Lemma) *Dati gli insiemi:*

$$\Delta_j \equiv \{\alpha_i \mid non\, \text{EC}\textsc{sat}(\neg\alpha_i \wedge \beta_j)\}$$

in EC si ha la seguente equivalenza:

$$\begin{array}{c} \text{ECSAT}(\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma) \\ \textit{se e solo se} \\ \textit{per ogni } j.\,(\Delta_j \equiv \emptyset \quad \textit{oppure} \quad \text{ECSAT}(\bigwedge_{\alpha_i \in \Delta_j} \alpha_i \wedge \neg\beta_j)) \\ \textit{e} \quad \text{ECSAT}(\gamma) \end{array}$$

Riscrivendo i Δ_j come:

$$\Delta_j \equiv \{\alpha_i \mid \text{per ogni } M_{ij}.\,\text{per ogni } w_{ij}.\, M_{ij}, w_{ij} \not\models \beta_j \rightarrow \alpha_i\} \tag{3.14}$$

possiamo riscrivere il lemma come segue:

$$\begin{array}{c} \text{esiste } M.\,\text{esiste } w.\; M, w \models (\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma) \\ \text{se e solo se} \\ \text{per ogni } j.\,(\;\Delta_j \equiv \emptyset \quad \text{oppure} \\ \text{esiste } M_j.\,\text{esiste } w_j.\; M_j, w_j \models \bigwedge_{\alpha_i \in \Delta_j} \alpha_i \wedge \neg\beta_j\;) \\ \text{e} \quad \text{esiste } M.\,\text{esiste } w.\; M, w \models \gamma \end{array} \tag{3.15}$$

con $M = \langle W, N, P\rangle \in \mathcal{E}_C$.

Dimostrazione "$\Rightarrow$" - Le linee guida di questa prova ricalcano quelle viste per i lemmi 3.1 e 3.2. Riprendiamo dal punto in cui consideravamo il particolare modello M e il particolare mondo w per i quali $M, w \models \bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j$. Possiamo sempre applicare la proprietà 3.1 e quindi abbiamo che per ogni i $\|\alpha_i\|^M \in N(w)$ mentre per ogni j $\|\beta_j\|^M \notin N(w)$. A questo punto bisogna considerare le proprietà specifiche per $\mathcal{E}_C$. In particolare la proprietà 3.6 ci consente di derivare che per ogni j $\|\beta_j\|^M \notin \bigcap^*\{\|\alpha_1\|^M \ldots \|\alpha_n\|^M\}$. Consideriamo allora un generico β_j per cui valga quest'ultima condizione e utilizziamo la proprietà 3.7 per ridurci a due casi: o l'insieme $\Delta_j \equiv \emptyset$ o $\|\beta_j\|^M \not\equiv \bigcap_{\alpha_i \in \Delta_j} \|\alpha_i\|^M$ Questi due casi possono essere ridotti al corrispondente test di soddisfacibilità utilizzando, rispettivamente, le proprieta 3.8 e 3.9. Pertanto possiamo conlcudere che:

$$\begin{array}{l} \{\alpha_i \mid \text{per ogni } M_{ij}.\,\text{per ogni } w_{ij}.\, M_{ij}, w_{ij} \not\models \beta_j \rightarrow \alpha_i\} \equiv \emptyset \quad \text{oppure} \\ \text{esiste } M_j.\,\text{esiste } w_j.\; M_j, w_j \models \bigwedge_{\alpha_i \in \Delta_j} \alpha_i \wedge \neg\beta_j) \end{array} \tag{3.16}$$

da cui, vista l'indipendeza dal β_j considerato, segue la tesi.

Dimostrazione "$\Leftarrow$" - Anche in questo caso, è sufficiente applicare la costruzione vista per E (lemma 3.1) imponendo che, sia i modelli di partenza, sia i modelli di arrivo, siano strutture in $\mathcal{E}_C$

A differenza delle logiche E ed EM, lo sforzo computazionale per determinare la soddisfacibilità nella logica EC è superiore a NP. In particolare in [Fagin *et al.*, 1995] si parla del test di soddisfacibilità per le strutture $\mathcal{E}_C$ come di un problema PSPACE-hard: tali problemi sono universalmente considerati più difficili da risolvere degli stessi problemi NP-hard. Prima di enunciare il risultato formale, cerchiamo di spiegare le ragioni intuitive per cui il test per EC è intrinsecamente più complesso. Come abbiamo brevemente richiamato nella parte introduttiva di questa sottosezione, i modelli per EC (la classe $\mathcal{E}_C$) sono tutte e sole le strutture $\mathcal{MS}$ i cui assegnamenti epistemici risultano chiusi per intersezione. Formalmente, qualsiasi $M = \langle W, N, P \rangle \in \mathcal{E}_C$ è tale che per ogni $w \in W$ risulta $N(w) = \bigcap^* N(w)$. Consideriamo allora una formula $\varphi = \Box\alpha_1 \wedge \ldots \wedge \Box\alpha_n \wedge \neg\Box\beta$. Abbiamo visto nei teoremi 3.1 e 3.2 che, per determinare la soddisfacibilità di φ in assenza della condizione di chiusura, è sufficiente confrontare il singolo β con tutti gli α. Questo fa si che, al massimo, per decidere φ in strutture come $\mathcal{E}$ e $\mathcal{E}_M$ siano sufficienti n test ricorsivi (tanti quanti sono gli α). Se siamo in $\mathcal{E}_C$ dobbiamo mostrare che β non deve essere logicamente equivalente a nessuna delle possibili congiunzioni tra gli α. In altre parole, la presenza della condizione di chiusura su N ci costringe a mostrare che non conoscere β è simultaneamente consistente con la conoscenza di qualsiasi congiunzione degli α. Per vedere se φ è soddisfacibile in $\mathcal{E}_C$, dobbiamo quindi considerare insiemi di sottoformule di φ, piuttosto che coppie di sottoformule. Dato che $\mid \mathcal{P}\ (sub(\varphi)) \mid = 2^{|sub(\varphi)|}$ il problema è PSPACE-hard.

Teorema 3.3 *Il problema della soddisfacibilità rispetto alle strutture in $\mathcal{E}_C$ è PSPACE-hard.*

Si veda [Vardi, 1989] per la presentazione di un possibile algoritmo per il test di soddisfacibilità in EC e la relativa prova del teorema 3.3.

3.3 Combinazione di algoritmi

Gli algoritmi e i risultati di questa sezione fondano la loro validità su quanto abbiamo esposto nella sezione precedente. In particolare tutte le logiche risultanti da E, EM ed EC con la semplice aggiunta dell'assioma N sono riunite nella sottosezione 3.3.1. Alla logica EMC è dedicata l'intera sottosezione 3.3.2, in quanto l'algoritmo è diverso dalla semplice combinazione di quelli visti per le logiche EM ed EC. Infine, in 3.3.3 consideriamo l'algoritmo per $EMCN$ e cioè l'algoritmo di decisione corretto e completo rispetto alle strutture $\mathcal{K}$.

3.3.1 Logiche EN, EMN ed ECN

L'aggiunta dell'assioma N, come avevamo visto in 2.3.2 comporta un particolare vincolo sull'assegnamento epistemico di tutte le strutture $\mathcal{E}_N$. Ricordiamo infatti che ogni $M = \langle W, N, P \rangle \in \mathcal{E}_N$ è tale che l'assegnamento epistemico per tutti i mondi contiene l'unità e cioè per ogni $w \in W$ si ha $W \in N(w)$. La stessa proprietà vale naturalamente per le classi $\mathcal{E}_{MN}$ e $\mathcal{E}_{CN}$

a cui vanno aggiunte, rispettivamente, chiusura per sovrainsiemi e chiusura per intersezione degli $N(w)$. Nelle dimostrazioni che seguono sfrutteremo pertanto le seguenti proprietà, valide per $\mathcal{E}_N$ e per tutte le sue sottoclassi:

Proprietà 3.10 *Dato un modello $M = \langle W, N, P\rangle \in \mathcal{E}_N$ se un mondo $w \in W$ è tale che $M, w \models \bigwedge_j \neg\Box\beta_j$ allora, per ogni j, $\|\beta_j\|^M \not\equiv W$*

Dimostrazione - Partendo dal fatto che $M, w \models \bigwedge_j \neg\Box\beta_j$ per interpretazione di "$\wedge$" si ricava che $M, w \models \neg\Box\beta_j$ per ogni j. D'altronde per interpretazione di "$\Box$" si ha che $\neg\Box\beta_j$ è vero in w sse $\|\beta_j\|^M \notin N(w)$. Per assurdo, se fosse $W \equiv \|\beta_j\|^M$ allora, dato che $M \in \mathcal{E}_N$, risulterebbe $\|\beta_j\|^M \in N(w)$.

Proprietà 3.11 *Data $\varphi \in \mathcal{L}$, se $M = \langle W, N, P\rangle \in \mathcal{E}_N$ è tale che $\|\varphi\|^M \not\equiv W$ allora esiste un mondo $w \in W$ tale che $M, w \models \neg\varphi$*

Dimostrazione - Banalmente, se non esistesse $w \in W$ tale che $M, w \models \neg\varphi$ allora sarebbe $M \models \varphi$ e quindi, per definizione di intensione di φ anche $W \equiv \|\varphi\|^M$.

A questo punto, sappiamo che i vincoli addizionali imposti dall'assioma N riguardano la parte $\bigwedge_j \neg\Box\beta_j$ dell'assegnamento e sappiamo anche come tradurli in un test di soddisfacibilità. Cominciamo quindi a includere tale test nella logica E per ottenere il seguente:

Lemma 3.4 (EN-Lemma) *In EN si ha la seguente equivalenza:*

$$\begin{gathered}
\text{EN}\textsc{sat}(\textstyle\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma) \\
\textit{se e solo se} \\
\textit{per ogni } i,j.\ \text{EN}\textsc{sat}(\alpha_i \leftrightarrow \neg\beta_j) \quad \textit{e} \\
\textit{per ogni } j.\ \text{EN}\textsc{sat}(\neg\beta_j) \quad \textit{e} \quad \text{EN}\textsc{sat}(\gamma)
\end{gathered}$$

Dimostrazione "$\Rightarrow$" - Si consideri l'equivalente dimostrazione per E a partire dal risultato 3.3 e siano M e w il particolare modello e il particolare mondo per i quali $M, w \models \bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j$. Applicando le proprietà 3.1 e 3.10 posso concludere che, comunque presi i e j risulta $\|\alpha_i\|^M \in N(w)$, $W \not\equiv \|\beta_j\|^M$ e $\|\beta_j\|^M \notin N(w)$. Particolarizzando per due generici α_i e β_j, con lo stesso ragionamento che avevamo fatto per E più l'utilizzo della proprietà 3.11 otteniamo:

$$\begin{aligned}
&\text{esiste } M_{ij}.\ \text{esiste } w_{ij}.\ M_{ij}, w_{ij} \models \alpha_i \leftrightarrow \neg\beta_j \quad \text{e} \\
&\text{esiste } M_j.\ \text{esiste } w_j.\ M_j, w_j \models \neg\beta_j
\end{aligned} \tag{3.17}$$

da cui la tesi.

Dimostrazione "$\Rightarrow$" - Si può riutilizzare la costruzione effettuata per la dimostrazione del lemma 3.1 a patto che, sia gli assegnamenti epistemici dei modelli di partenza (M_{ij}) sia l'assegnamento N del modello complessivo (M) siano tali da contenere l'unità.

Includendo l'assioma N nella logica EMN dobbiamo modificare l'algoritmo in base al seguente:

Lemma 3.5 (EMN-Lemma) *In EMN si ha la seguente equivalenza:*

$$\begin{gathered}\mathrm{EMN}\textsc{sat}(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)\\ \textit{se e solo se}\\ \textit{per ogni } i,j.\ \mathrm{EMN}\textsc{sat}(\alpha_i \wedge \neg\beta_j) \quad e \quad \mathrm{EMN}\textsc{sat}(\gamma)\end{gathered}$$

Non forniamo la dimostrazione di questa proprietà in quanto può essere ricavata con le stesse considerazioni che abbiamo fatto a proposito della dimostrazione per il lemma 3.4.

Come abbiamo enunciato nei teoremi 3.1 e 3.2 il problema della soddisfacibilità in $\mathcal{E}$ e $\mathcal{E}_M$ è NP-complete. Di fatto, l'aggiunta dell'assioma N non è tale da aumentare la complessità del problema per cui il test di soddisfacibilità in EN e EMN è ancora NP-complete. Intuitivamente è lecito aspettarsi questo risultato, in quanto il test di soddisfacibilità per l'assioma N coinvolge, al più, un numero $\mid sub_\Box(\varphi) \mid$ di test NP. Pertanto, nel caso peggiore, gli algoritmi per EN ed EMN svolgono al massimo un numero $\mid sub_\Box(\varphi) \mid^2$ di test NP. Notiamo che in questo caso la stima è migliore rispetto agli algoritmi per E e per EM, in cui, di fatto, avevamo considerato anche i test coinvolgenti la coppia (ψ, ψ) che non sono mai eseguiti. Formalizziamo quanto detto nel seguente:

Teorema 3.4 *Il problema della soddisfacibilità rispetto alle strutture $\mathcal{E}_N$ e $\mathcal{E}_{MN}$ è NP-complete.*

Il risultato di questo teorema è presentato in [Vardi, 1989] sotto forma di corollario al teorema 3.1 per la soddisfacibilità in EN e sotto forma di generalizzazione del corollario stesso e del teorema 3.2 per la logica EMN.

L'ultimo algoritmo che presentiamo in questa sezione e quello relativo alla logica ECN. Inlcudendo l'assioma N, il lemma 3.3 risulta modificato come segue:

Lemma 3.6 (ECN-Lemma) *Dati gli insiemi:*

$$\Delta_j \equiv \{\alpha_i \mid \textit{non}\ \mathrm{EC}\textsc{sat}(\neg\alpha_i \wedge \beta_j)\}$$

in EC si ha la seguente equivalenza:

$$\begin{gathered}\mathrm{ECN}\textsc{sat}(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)\\ \textit{se e solo se}\\ \textit{per ogni } j.\ (\ (\Delta_j \equiv \emptyset \quad e \quad \mathrm{ECN}\textsc{sat}(\neg\beta_j)) \quad \textit{oppure}\\ \mathrm{ECN}\textsc{sat}(\bigwedge_{\alpha_i \in \Delta_j} \alpha_i \wedge \neg\beta_j)\ )\\ e \quad \mathrm{ECN}\textsc{sat}(\gamma)\end{gathered}$$

Così come accadeva per EC, anche in ECN il caso pessimo è tale da costringerci a esplorare completamente tutti i possibili sottoinsiemi delle formule in $sub_{\Box}(\varphi)$. Pertanto vale il seguente:

Teorema 3.5 *Il problema della soddisfacibilità rispetto alle strutture in $\mathcal{E}_{CN}$ è PSPACE-hard.*

Per la prova di questo risultato facciamo sempre riferimento a [Vardi, 1989].

3.3.2 Logica R (EMC)

Nella sezione 2.4 avevamo parlato della logica R nei temini di sistema deduttivo regolare e avevamo visto che è possibile fornire diverse caratterizzazioni assiomatiche. La più complessa, che rispecchia la progressiva inclusione dei modelli $\mathcal{E} \supset \mathcal{E}_M \supset \mathcal{E}_{CM}$ richiederebbe un test che verifichi la consistenza in EM oppure in EC a cui andrebbe aggiunto il test di validità relativo all'altro assioma, cioe C oppure M. D'altronde abbiamo visto che è possibile caratterizzare la classe $\mathcal{E}_{CM}$, detta anche dei *quasi-filtri*, per chiusura sulla regola RR (vedi la def. 2.22 sempre in 2.4).

Vediamo quali sono le particolari proprietà della logica R. Ricordiamo la definizione 3.1 di chiusura per intersezione di un insieme S denotata con $\bigcap^* S$ (vedi sezione 3.2.3). Vogliamo adattare tale definizione alle proprietà dei modelli in $\mathcal{E}_{MC}$ con la seguente:

Definizione 3.2 (Chiusura per intersezioni supplementata) *Dati gli insiemi $s_1 \dots s_k$ e l'insieme $S = \{s_1 \dots s_k\}$ definiamo $\bigcap^{\subset *} S$ l'insieme costruito come segue:*

- *$s_i \in \bigcap^{\subset *} S$ per ogni i, $1 \leq i \leq k$;*
- *se $s, t \in \bigcap^{\subset *} S$ allora anche $s \cap t \in \bigcap^{\subset *} S$;*
- *se $s \in \bigcap^{\subset *} S$ e $t \supset s$ allora $t \in \bigcap^{\subset *} S$.*

Consideriamo allora la seguente proprietà:

Proprietà 3.12 *Dati $M = \langle W, N, P\rangle \in \mathcal{E}_C$ e $\varphi_1 \dots \varphi_k, \psi \in \mathcal{L}$, se un mondo $w \in W$ è tale che $\|\varphi_i\|^M \in N(w)$ per ogni i $(1 \leq i \leq k)$ e $\|\psi\|^M \notin N(w)$ allora $\|\psi\|^M \notin \bigcap^{\subset *}\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\}$.*

Dimostrazione - Assumiamo, per assurdo, che valga $\|\varphi_i\|^M \in N(w)$ per ogni φ_i e $\|\psi\|^M \notin N(w)$ anche con $\|\psi\|^M \in \bigcap^{\subset *}\{\|\varphi_1\|^M \dots \|\varphi_k\|^M\}$. Allora sono dati tre casi:

- $\|\psi\|^M \equiv \|\varphi_i\|^M$ per qualche φ_i e questo è assurdo perché allora $\|\psi\|^M \equiv \|\varphi_i\|^M \in N(w)$;

- $\|\psi\|^M \in \bigcap^*\{\varphi_1 \ldots \varphi_k\}$ e dato che $\mathcal{E}_{MC} \subset \mathcal{E}_C$ risulta anche $\|\psi\|^M \in N(w)$ da cui l'assurdo;
- $\|\psi\|^M \supset \|\varphi_i\|^M$ per qualche φ_i e dato che $\mathcal{E}_{MC} \subset \mathcal{E}_C$ risulta ancora $\|\psi\|^M \in N(w)$ da cui l'assurdo.

Come al solito, cerchiamo di ricondurci dalle proprietà relative alle intensioni delle formule alle proprietà riguardanti la loro soddisfacibilità in particolari strutture. Cominciamo con la seguente:

Proprietà 3.13 *Dati $M = \langle W, N, P\rangle \in \mathcal{E}_C M$ e $\varphi_1 \ldots \varphi_k, \psi \in \mathcal{L}$, risulta $\|\psi\|^M \notin \bigcap^{\subset *}\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$ se e solo se $\|\psi\|^M \not\supseteq \bigcap_{i=1}^k \|\varphi_i\|^M$.*

Dimostrazione "$\Rightarrow$" - Banale, in quanto risulta che:

$$\bigcap_{i=1}^{k} \|\varphi_i\|^M \in \bigcap^{\subset *}\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\} \tag{3.18}$$

Dimostrazione "$\Leftarrow$" - Per assurdo, mostrando che se $\|\psi\|^M \not\supseteq \bigcap_{i=1}^k \|\varphi_i\|^M$ allora, essendo $\bigcap_{i=1}^k \|\varphi_i\|^M$ l'insieme più piccolo di $\bigcap^{\subset *}\{\|\varphi_1\|^M \ldots \|\varphi_k\|^M\}$, l'intensione di ψ è diversa da qualsiasi membro di $\bigcap^{\subset *}\{\ldots\}$.

L'ultimo passo è quello di ricondurci ad un test di soddisfacibilità con la seguente:

Proprietà 3.14 *Siano $\varphi_1 \ldots \varphi_k, \psi \in \mathcal{L}$, $M = \langle W, N, P\rangle \in \mathcal{E}_C$. Se $\|\psi\|^M \not\supseteq \bigcap_{i=1}^k \|\varphi_i\|^M$ allora esiste un mondo $w \in W$ tale che $M, w \models \bigwedge_i \varphi_i \wedge \neg\psi$.*

Dimostrazione - Per assurdo, se non esiste un mondo w tale che $M, w \models \bigwedge_i \varphi_i \wedge \neg\psi$, allora significa che $M \models \bigwedge_i \varphi_i \wedge \neg\psi$ da cui, per definizione di intensione, $\|\psi\|^M \supseteq \bigcap_{i=1}^k \|\varphi_i\|^M$ e l'assurdo.

Sulla base delle proprietà introdotte possiamo enunciare il seguente:

Lemma 3.7 (R-Lemma) *In R (EMC) si ha la seguente equivalenza:*

$$\begin{gathered}\textsc{Rsat}(\textstyle\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma) \\ \textit{se e solo se} \\ \textit{per ogni } j.\ \textsc{Rsat}(\textstyle\bigwedge_i \alpha_i \wedge \neg\beta_j) \quad e \quad \textsc{Rsat}\neg\gamma\end{gathered}$$

La dimostrazione del lemma 3.7 può essere affrontata sia utilizzando le proprietà dalla 3.12 alla 3.14 e le considerazioni che abbiamo fatto per le dimostrazioni di altri lemmi, sia partendo dalle strutture $\mathcal{K}$ e generalizzandole opportunamente per tenere conto della mancanza dello schema N. In particolare quest'ultimo approccio è quallo seguito in letteratura (vedi, ad esempio, [Fitting, 1983]) da cui, salvo qualche piccola modifica, è possibile ricavare la dimostrazione del lemma 3.7.

Il risultato relativo alla complessità di una generica procedura di decisione in R è contenuto in [Vardi, 1989]. In particolare si dimostra che la complessità di tale problema è PSPACE-hard. Il ragionamento qualitativo è lo stesso fatto a proposito di EC, dato che anche in R siamo costretti a controllare sottoinsiemi di formule, piuttosto che coppie. Formalmente, diciamo che vale il seguente:

Teorema 3.6 *Il problema della soddisfacibilità rispetto alle strutture $\mathcal{E}_{MC}$ è PSPACE-hard.*

3.3.3 Logica K $(EMNC)$

L'algoritmo corrispondente a K è una generalizzazione di quello visto in precedenza per la logica R. In particolare è l'aggiunta del vincolo di consistenza relativo all'assioma N che li differenzia. Dal punto di vista semantico sappiamo che K è la prima logica i cui agenti sono logicamente onniscienti e quindi l'algoritmo che segue risolve il problema della soddisfacibilità in una generica struttura $\mathcal{K}$:

Lemma 3.8 (K-Lemma) *In K $(EMCN)$ si ha la seguente equivalenza:*

$$\begin{array}{c}\textsc{Ksat}(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma) \\ \textit{se e solo se} \\ \textit{per ogni } j.\ \textsc{Ksat}(\bigwedge_i \alpha_i \wedge \neg\beta_j) \quad e \quad \textsc{Ksat}\neg\gamma\end{array}$$

Qualitativamente la prova di K-Lemma non è diversa da quelle presentate per gli altri algoritmi. Anche in questo caso, l'implicazione "$\Rightarrow$" risulta piuttosto semplice. Si tratta di provare che, dato un modello in cui un mondo considera possibili solo mondi in cui valgono tutti gli α e nessuno dei β, questo significa che esisteranno m mondi in cui vale $\bigwedge_i \alpha_i$ e non β_j. Il fatto è ovvio se si considera che la regola di consistenza RK è tale che $\bigwedge_i \varphi_i \rightarrow \varphi \Rightarrow \bigwedge_i \Box\varphi \rightarrow \Box\varphi$ per cui l'assenza di un mondo in cui valgono tutti gli α e non β mi costringerebbe ad ammettere che la congiunzione degli α implica β per cui l'assegnamento diventerebbe incosistente. La freccia "$\Leftarrow$" coinvolge come nel caso degli altri algoritmi, la costruzione di un modello a partire dall'insieme dei modelli che soddisfano $\bigwedge_i \alpha_i \wedge \neg\beta_j$ per qualche β_j, con l'aggiunta di un mondo ad hoc per mostrare che l'assegnamento è consistente. Una dimostrazione formale di questo risultato è contenuta in [Sebastiani, 1997] a cui si rimanda per eventuali approfondimenti.

La complessità del problema della soddisfacibilità in una generica struttura $\mathcal{K}$ è nota, in letteratura, come PSPACE-complete. Risultati relativi alla complessità sono contenuti in [Vardi, 1989] ed anche in [Fagin *et al.*, 1995]. Tra l'altro, sempre in [Fagin *et al.*, 1995] viene dimostrato che se una formula φ è soddisfacibile in una struttura $\mathcal{K}$, allora questa struttura può, nel caso pessimo, avere un numero esponenziale di mondi (in particolare $2^{|\varphi|}$). Enunciamo allora il seguente:

Teorema 3.7 *Il problema della soddisfacibilità rispetto alle strutture $\mathcal{K}$ è PSPACE-complete.*

Capitolo 4

Realizzazione dei decisori

In questo capitolo presentiamo la realizzazione degli algoritmi di decisione introdotti formalmente nel capitolo precedente. Trattandosi di algoritmi SAT-based si è imposta la scelta di una procedura di decisione SAT sulla base della quale sviluppare i decisori. La sezione 4.1 illustra il funzionamento della procedura di decisione Davis-Putnam-Longemann-Loveland e i motivi per cui è stata scelta per realizzare i nostri algoritmi. Nella sezione 4.2 presentiamo il *framework* con il quale è possibile sviluppare tutti i decisori per le logiche Λ. Lo schema di procedura presentato è completato nella sezione 4.3 aggiungendo le procedure relative alla sola inferenza modale. Con questi preliminari, introduciamo la dimostrazione di correttezza e completezza a cui è dedicata l'intera sezione 4.4 e, nella sezione 4.5, descriviamo una realizzazione delle procedure DPLL-based che coniuga l'efficienza del ragionamento proposizionale con una potente tecnica di ausilio all'inferenza modale.

4.1 Il decisore proposizionale DPLL

Introdotta nell'articolo [Davis e Putnam, 1960] e in seguito portata alla forma attuale in [Davis *et al.*, 1962], la procedura DPLL (acronimo per Davis-Putnam-Longemann-Loveland) è, all'attuale stato dell'arte, uno fra i più efficienti decisori *completi* per la soddisfacibilità di formule proposizionali. Di fatto, oltre che nel campo dell'intelligenza artificiale, realizzazioni della procedura DPLL vengono utilizzate sia nel campo della ricerca su SAT (si veda, ad esempio, [Mitchell *et al.*, 1992]), sia nelle applicazioni quali verifica di circuiti e protocolli, dove DPLL ha mostrato di essere in grado di confrontarsi alla pari con procedure decisamente più complesse dal punto di vista concettuale (un'analisi di questo tipo si trova in [Uribe e Stickel, 1994]).

La versione base della procedura DPLL utilizzata per i nostri decisori è presentata in figura 4.1 e nel seguito, come in figura, ci riferiremo ad essa con il termine PLSAT$_{DP}$.

Figura 4.1: Il decisore proposizionale PLSAT$_{DP}$

---

function PLSAT$_{DP}(\varphi, \mu)$
  if $\varphi = \top$ **then return** μ;
  if $\varphi = \bot$ **then return** *False*;
  if {a unit clause (l) occurs in φ}
    then return PLSAT$_{DP}(assign(l, \varphi), \mu \wedge l)$;
  l := *choose-literal*(φ);
  return PLSAT$_{DP}(assign(l, \varphi), \mu \wedge l)$ **or**
        PLSAT$_{DP}(assign(\neg l, \varphi), \mu \wedge \neg l)$.

Descriviamo brevemente il funzionamento di PLSAT$_{DP}$ assumendo, per ora, che φ sia una formula proposizionale CNF e cioè:

$$\varphi = \bigwedge_{j}^{L} \bigvee_{j}^{R} l_{ij} \tag{4.1}$$

La formula in 4.1 è una congiunzione finita di L *clausole*, che a loro volta sono una disgiunzione finita di R *letterali*. Ogni letterale l è un atomo proposizionale, eventualmente negato. Sfruttando un'analogia con il progetto dei ciruiti digitali, la conversione di una formula in CNF non è altro che una sintesi AND-OR a due livelli della corrispondente rete combinatoria.

PLSAT$_{DP}$ viene invocata con la chiamata iniziale PLSAT$_{DP}(\varphi, \top)$ in cui φ è una formula proposizionale CNF e $\top$ rappresenta l'assegnamento vuoto (vedi 2.3.3). La procedura restituisce un assegnamento μ se φ è soddisfacibile e *False* altrimenti. In sostanza, PLSAT$_{DP}$ cerca di costruire μ partendo da $\top$ e aggiungendo un letterale ad ogni chiamata ricorsiva. Questo avviene secondo i passi seguenti:

[base] Se φ non contiene alcuna clausola (cioè $\varphi = \top$), μ è un assegnamento (eventualmente parziale) per la formula originale φ e viene ritornato;

[backtrack] Se φ contiene una clausola vuota (cioè $\varphi = \bot$), μ non è un assegnamento e pertanto la funzione ritorna *False*;

[unit] Se un letterale l occorre in φ come clausola unitaria, cioè:

$$\varphi = \varphi_1 \wedge l \wedge \varphi_2$$

allora l deve essere necessariamente assegnato a vero. Pertanto ΛSAT$_{DP}$ viene invocato ricorsivamente utilizzando come parametri la formula restituita da $assign(l, \varphi)$ e l'assegnamento $\mu \wedge l$, cioè $\mu \cup \{\psi := 1\}$ se $l = \psi$ e $\mu \cup \{\psi := 0\}$ se $l = \neg\psi$. La funzione $assign(l, \varphi)$ assegna $l := 1$ e semplifica φ di conseguenza. Questo equivale a rimuovere le occorrenze negative di l e rimuovere le clausole dove l occorre positivamente;

[split] Se nessuno dei casi precedenti si verifica è necessario, con un qualche criterio euristico, scegliere un letterale l non ancora assegnato. La chiamata ricorsiva viene effettuata con $assign(l, \varphi)$ e $\mu \wedge l$ e, in caso di fallimento, con $assign(\varphi, \neg l)$ e $\mu \wedge \neg l$.

Come avevamo anticipato all'inizio della nostra discussione, il decisore DPLL è una procedura corretta e completa rispetto ai modelli del ragionamento proposizionale. Formalizziamo questo fatto nel seguente:

Teorema 4.1 PLSAT$_{DP}$ *è corretto e completo rispetto ai modelli della logica proposizionale e cioè* PLSAT$_{DP}(\varphi)$ = *True se e solo se* $\not\vdash_{PL} \neg\varphi$.

Concludiamo questa sezione con alcuni commenti circa la nostra realizzazione della procedura PLSAT$_{DP}$. Ci sono tre importanti questioni che meritano attenzione quali:

1. la scelta della particolare euristica per *choose-literal*. Nel nostro caso la funzione è tale da ritornare sempre il letterale che occorre il maggior numero di volte all'interno della formula. Nel caso ottimo questa euristica realizza una "discesa rapida" nella valutazione della formula, ma una scelta sbagliata, soprattutto se iniziale, potrebbe costringerci a sprecare un numero elevato di passi computazionali. Si rimanda comunque a [Sebastiani, 1997] per una bibliografia dettagliata sulle possibili realizzazioni di *choose-literal*.

2. il fatto che la nostra procedura, nello spirito di [Armando e Giunchiglia, 1993], gestisca anche formule non CNF. Si rendono quindi necessari dei metodi per rivelare l'esistenza di clausole unitarie (senza che queste siano rappresentate esplicitamente) e metodi per la valutazione parziale delle formule (funzione *assign*). La procedura PLSAT$_{DP}$ utilizza gli stessi metodi, almeno in principio, presentati in [Armando e Giunchiglia, 1993].

3. il fatto che la procedura non tratti solo formule puramente proposizionali, ma formule appartenenti al linguaggio modale $\mathcal{L}$ definito nella sezione 2.2. I metodi di cui al punto precedente hanno dovuto essere modificati per tenere conto di questo fatto. In particolare, la modifica effettuata comporta una "etichettatura" univoca degli atomi modali in modo da poterli gestire dal punto di vista del ragionamento proposizionale.

Nella sezione 4.5 vediamo come l'etichettatura delle sottoformule modali sia necessaria anche per la realizzazione delle particolari strutture dati usate per rendere i decisori più efficienti. In ogni caso, per la descrizione di questo procedimento e di altre questioni tecniche riguardanti i punti sopra esposti, abbiamo predisposto l'intera appendice A a cui rimandiamo per eventuali approfondimenti.

4.2 Schema dei decisori modali DPLL-based

Come abbiamo visto nella sezione 3.1, un algoritmo di decisione basato su SAT è composto da due parti:

1. una procedura di decisione SAT utilizzata per costruire assegnamenti per la formula modale;
2. una regola di consistenza che analizza la parte modale dell'assegnamento e invoca ricorsivamente il decisore proposizionale.

Ci proponiamo di raffinare questa struttura, soprattutto per quanto riguarda il punto relativo al decisore proposizionale e all'interazione tra quest'ultimo e la parte modale. Il frammento di pseudo-codice presentato in figura 4.2 descrive ΛSAT cioè lo schema generale delle procedure di decisione parametrizzato rispetto alle diverse logiche. Nel seguito, quando parliamo di procedura ΛSAT intendiamo riferirci alla *classe* di procedure di decisione ottenute istanziando ΛSAT ad una particolare logica Λ. Tale procedimento prevede la sostituzione della procedura astratta ΛSAT_A con una fra le procedure presentate nelle sezioni 4.3 e 4.5.

Vediamo una descrizione dettagliata della procedura di decisione. ΛSAT prende in ingresso una formula modale φ e restituisce un valore booleano relativo alla soddisfacibilità nella particolare logica Λ. ΛSAT invoca ΛSAT_{DP} passando come argomenti la formula φ e l'assegnamento vuoto $\top$ (vedi al proposito la sezione 2.2). ΛSAT_{DP} cerca un assegnamento per φ che sia anche consistente nella particolare logica Λ. I casi gestiti dalla procedura sono i seguenti:

[base] Se φ viene semplificata a $\top$ significa che μ è un assegnamento per φ. Visto che se μ è consistente in Λ anche φ lo è, viene invocato il test di consistenza ΛSAT_A. Quest'ultima routine deve restituire *True* se μ è consistente in Λ, *False* nel caso contrario.

[backtrack] Se φ viene semplificata a $\perp$ significa che μ non è un assegnamento per φ. In questo caso è necessario costruire μ diversamente e pertanto ΛSAT_{DP} ritorna *False*.

Figura 4.2: Schema generale delle procedure DPLL-based

---

function $\Lambda\text{SAT}(\varphi)$
  return $\Lambda\text{SAT}_{DP}(\varphi, T)$.

function $\Lambda\text{SAT}_{DP}(\varphi, \mu)$
  if $\varphi = \top$ **then return** $\Lambda\text{SAT}_{A}(\mu)$;
  if $\varphi = \bot$ **then return** *False*;
  if {a unit clause (l) occurs in φ}
    then return $\Lambda\text{SAT}_{DP}(assign(l, \varphi), \mu \wedge l)$;
  if Λ-*likely-unsatisfiable*(μ)
    then if not $\Lambda\text{SAT}_{A}(\mu)$ **then return** *False*;
  $l :=$ *choose-literal*(φ);
  return $\Lambda\text{SAT}_{DP}(assign(l, \varphi), \mu \wedge l)$ **or**
      $\Lambda\text{SAT}_{DP}(assign(\neg l, \varphi), \mu \wedge \neg l)$.

function $\Lambda\text{SAT}_{A}(\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma)$
  foreach $\Gamma_{\Box\alpha}$ **do**
    foreach $\Gamma_{\Box\beta}$ **do**
      if {Λ-consistency test $\Lambda\text{SAT}(\Gamma_\alpha \circ \Gamma_\beta)$ fails}
        then return *False*
  return *True*.

[unit] Se un letterale l occorre in φ come clausola unitaria allora l deve essere assegnato a vero. Pertanto ΛSAT_{DP} viene invocato ricorsivamente utilizzando come parametri la formula restituita da $assign(l, \varphi)$ e l'assegnamento $\mu \wedge l$.

[early pruning] Sotto certe condizioni, dipendenti sia dalla particolare logica Λ, sia dalle caratteristiche della formula φ, è possibile che un assegnamento incompleto sia potenzialmente inconsistente. In questi casi è opportuno invocare un controllo di consistenza modale il cui fallimento può "tagliare" immediatamente un ramo di ricerca comunque infruttuoso.

[split] Se nessuno dei casi precedenti si verifica, è necessario scegliere un letterale l (per i dettagli si veda 4.1) sulla base del quale "dividere" l'albero di ricerca degli assegnamenti. Viene quindi effettuata una chiamata ricorsiva a ΛSAT_{DP} con $assign(\varphi, l)$ e $\mu \wedge l$ e, in caso di fallimento, una seconda chiamata ricorsiva con $assign(\varphi, \neg l)$ e $\mu \wedge \neg l$.

È evidente, dall'esame della procedura presentata in figura 4.2, che ΛSAT$_{DP}$ non è altro che una semplice variante della procedura PLSAT$_{DP}$ analizzata nella sezione introduttiva di questo capitolo. L'unica differenza sostanziale è l'aggiunta del passo denotato come *early pruning* di cui siamo interessati a motivare la particolare posizione all'interno di ΛSAT$_{DP}$ e a stabilire, informalmente, l'importanza per l'efficienza dei nostri decisori.

Il motivo per cui il test di early pruning si trova tra il passo *unit* e il passo *split* è dovuto semplicemente al fatto che la ricorsione sul primo (detta *unit propagation*) ha un costo molto minore rispetto alla ricorsione sul secondo (detta *splitting*). In particolare, unit propagation ha un costo al più polinomiale nelle dimensioni di φ mentre, ovviamente, splitting ha un costo al più esponenziale nelle dimensioni di φ. Assumiamo, per semplicità, che l'euristica Λ-*likely-unsatisfiable* ritorni sempre *True*. Quando tutte le clausole unitarie, tra cui vi possono essere atomi modali, sono state assegnate ma l'assegnamento μ è ancora incompleto, viene lanciato il test di consistenza modale. Si noti che quest'ultimo lavora su un assegnamento più piccolo e quindi più facile da gestire. Il caso ottimo si ha quando ΛSAT$_A(\mu)$ ritorna *False* perché in tal caso possiamo ritornare *False* e cioè fare un passo di *backtrack* alla ricerca di un nuovo assegnamento per φ. Notiamo che in tutto questo procedimento non abbiamo mai fatto splitting. Nel caso sfortunato, abbiamo "perso tempo" con un assegnamento incompleto e siamo comunque costretti ad eseguire un passo di splitting. Quello che possiamo aspettarci, avendo scelto di eseguire sempre il passo early pruning, è, in media, una maggiore efficienza soprattutto per formule molto grandi e molto "vincolate".

La procedura ΛSAT$_A$ (che chiamiamo anche *test di consistenza*) viene invocata su tutti gli assegnamenti costruiti per φ. Il suo compito è quello di eseguire una serie di chiamate ricorsive concernenti la soddisfacibilità di sottoformule di μ. Nel seguito ci riferiamo a queste chiamate con il termine *test ricorsivi* per evidenziare il fatto che la consistenza dell'assegnamento è messa alla prova invocando ricorsivamente il decisore ΛSAT. Il frammento di pseudo codice relativo a ΛSAT$_A$ presentato in figura 4.2 intende catturare l'essenza di tutte le procedure che sono contenute nella sezione 4.3 senza entrare nel dettaglio di come venga eseguito il test nella specifica logica. In particolare, con $\Gamma_{\Box\alpha}$ e $\Gamma_{\Box\beta}$ vogliamo indicare gli insiemi di sottoformule di μ, eventualmente singoli o vuoti, che vengono presi in esame e con Γ_α e Γ_β vogliamo indicare che il test ricorsivo avviene su formule ricavate dai corrispondenti insiemi $\Gamma_{\Box\alpha}$ e $\Gamma_{\Box\beta}$ considerando solo il "corpo" della formule precedute dall'operatore "$\Box$".

Ritornando alla procedura nel suo insieme, osserviamo che ΛSAT possiede tre caratteristiche fondamentali per un decisore:

[Terminazione] ΛSAT è garantita terminare in tempo finito a patto che valga sempre $depth(\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma) > depth(\Gamma_\alpha \circ \Gamma_\beta)$;

[**Correttezza**] ΛSAT è corretta rispetto ai modelli della generica logica Λ.

[**Completezza**] ΛSAT è una procedura completa rispetto ai modelli della generica logica Λ.

A proposito della terminazione, ricordiamo che, grazie ai risultati presentati nella sezione 2.4, siamo a conoscenza del fatto che le logiche Λ sono corrette e complete rispetto a strutture $\mathcal{MS}$con un numero finito di elementi. Questo ci garantisce che qualsiasi decisore corretto e completo rispetto ai modelli delle logiche Λ *possa* essere terminante. D'altronde, in maniera qualitativa, possiamo fare le seguenti osservazioni:

1. ΛSAT genera al più un numero finito $2^{|Atom(\varphi)|}$ di assegnamenti μ su cui viene invocato ΛSAT_A e per cui risulta $depth(\mu) = depth(\varphi)$ (si veda il lemma 4.1);

2. se l'assegnamento μ costruito da ΛSAT_{DP} è puramente proposizionale ($depth(\mu) = 0$), ΛSAT_A ritorna *True* e la ricorsione termina;

3. viceversa, se l'assegnamento μ è tale che $depth(\mu) > 0$, ΛSAT_A genera un numero finito di chiamate ricorsive tali che per la generica formula $\Gamma_\alpha \circ \Gamma_\beta$ vale sempre la diseguaglianza; $depth(\mu) > depth(\Gamma_\alpha \circ \Gamma_\beta)$. Se le chiamate hanno tutte successo la procedura termina, viceversa viene provato un'altro μ;

Nel loro complesso, i tre punti appena esposti, ci consentono di affermare che la procedura ΛSAT è terminante. In [Sebastiani, 1997] è contenuta una prova formale di terminazione per la procedura KSAT che può essere facilmente estesa a tutte le logiche Λ precisando il punto 3 dell'elenco precedente.

4.3 Inferenza modale

In questa sezione ci proponiamo di realizzare gli algoritmi presentati nelle sezioni 3.2 e 3.3. Pertanto presentiamo le versioni "base" delle procedure che gestiscono l'inferenza modale per tutte le logiche Λ. Per ogni procedura ci soffermiamo sia sulla descrizione dello psuedo codice, sia sul particolare ragionamento modale realizzato. In particolare, attraverso una serie di esempi, cerchiamo di dare l'intuizione relativa alle possibilità di inferenza epistemica di ogni logica.

4.3.1 La logica E

È possibile affrontare il problema della consistenza in E con due diverse strategie. Come dimostrato per il lemma 3.1 nella sezione 3.2.1, l'assegnamento μ è consistente in E a patto che non vi siano due atomi modali, di cui uno assegnato a vero e l'altro assegnato a falso, che siano logicamente equivalenti. Questa considerazione porta alla procedura descritta in figura 4.3.

Figura 4.3: Test di consistenza per E - Versione 1

---

function $\textsc{Esat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      if not $\textsc{Esat}(\alpha_i \leftrightarrow \neg\beta_j)$ **then return** *False*;
  return *True*.

La procedura consta di due loop annidati che scandiscono, rispettivamente, le formule α e le formule β: per ogni coppia viene invocato il test ricorsivo sulla soddisfacibilità della formula $\alpha \leftrightarrow \neg\beta$. Se $\textsc{Esat}(\alpha \leftrightarrow \neg\beta)$ ritorna *True* il loop prosegue, altrimenti $\textsc{Esat}_A$ ritorna *False* alla procedura chiamante. Se i test ricorsivi hanno sempre successo il valore di ritorno è *True*. Si noti che in assenza di formule α oppure in assenza di formule β, l'assegnamento è banalmente soddisfacibile (i loop non cominciano). Questo è corretto perché, in base a quanto abbiamo visto nelle sezioni 2.3 e 3.2.1, l'unico vincolo nei modelli in $\mathcal{E}$ è l'impossibilità per un'agente di avere conoscenza contraddittoria.

Il funzionamento della procedura dal punto di vista logico può essere chiarito considerando il seguente esempio. Supponiamo di avere una formula $\varphi = \Box P_0 \wedge \neg\Box(P_0 \wedge \top)$. Risulta che $\mu = \{\Box P_0 = T, \Box(P_0 \wedge \top) = F\}$ è un assegnamento per φ ma, d'altro canto, μ è *modalmente* inconsistente nei modelli $\mathcal{E}$. Intuitivamente sappiamo che è assurda la situazione per cui, contemporaneamente, si conosce P_0 e non si conosce $P_0 \wedge \top$. D'altro canto, risultano soddisfacibili formule come $\varphi_1 = \Box(\neg\psi \wedge \psi)$ oppure $\varphi_2 = \neg\Box(\neg\psi \vee \psi)$ e questo perché il particolare modello epistemico soggiacente alla logica E (vedi sezione 2.3) è stato scelto per consentirci la conoscenza di formule contraddittorie ($\varphi_1 = \Box\bot$) ed anche la non conoscenza delle formule valide ($\varphi_2 = \neg\Box\top$). Quindi è ovvio che un assegnamento non può essere modalmente inconsistente quando si limita a "proporre" conoscenza (solo formule α) oppure non conoscenza (solo formule β). Proprio per sottolineare queste limitate possibilità di inferenza epistemica la logica E viene definita in [Vardi, 1989] come la più "debole" fra le logiche modali classiche.

Un'ultima osservazione riguarda la forma del test ricorsivo. Si noti come sia stato possibile gestire direttamente l'equivalenza logica invocando $\textsc{Esat}(\alpha \leftrightarrow \neg\beta)$. Questo perché, come avevamo già fatto notare nella sezione 4.1, la procedura $\textsc{Esat}_{DP}$ accetta formule non clausificate. Di fatto, questa scelta ci consente una notevole flessibilità nella scrittura dei test ricorsivi per i nostri decisori.

Considerando il significato semantico del test ricorsivo possiamo ricondurci ad una seconda forma per la procedura $\textsc{Esat}_A$. Infatti, un generico α e un generico β non sono equivalenti nella logica E ogni volta che esiste almeno un modello, e in questo un mondo, in cui vale α ma non β o viceversa. Dal punto di vista formale vale la seguente equivalenza:

$$\begin{array}{c} \text{esiste } M.\ \text{esiste } w.\ M,w \models \alpha \leftrightarrow \neg\beta \\ \text{se e solo se} \\ (\text{esiste } M.\ \text{esiste } w.\ M,w \models \alpha \wedge \neg\beta \quad \text{oppure} \\ \text{esiste } M.\ \text{esiste } w.\ M,w \models \neg\alpha \wedge \beta) \end{array} \tag{4.2}$$

In base a questa considerazione otteniamo la procedura descritta nella figura 4.4.

Figura 4.4: Test di consistenza per E - Versione 2

function $\textsc{Esat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      if not $\textsc{Esat}(\alpha_i \wedge \neg\beta_j)$
        then if not $\textsc{Esat}(\neg\alpha_i \wedge \beta_j)$ **then return** *False*;
  return *True*.

Realizzare il test come disgiunzione in senso *lazy*[1]di due chiamate ricorsive può, nel caso pessimo, raddoppiarne il numero. D'altronde le formule $\alpha \wedge \neg\beta$ e $\neg\alpha \wedge \beta$ sono sicuramente più compatte rispetto alla coimplicazione e hanno comunque un diverso contenuto di informazione. Ad esempio sapere che $\alpha \leftrightarrow \neg\beta$ è soddisfacibile non ci dice nulla circa la validità di α o β. Invece se sappiamo che $\alpha \wedge \neg\beta$ è soddisfacibile, allora sappiamo che β non è una formula valida. D'altro canto l'insoddisfacibilità di $\alpha \leftrightarrow \neg\beta$ ci fornisce l'informazione sull'equivalenza logica tra α e β per cui sappiamo che non ha senso provare assegnamenti con α assegnato a vero e β assegnato a falso.

Non vogliamo approfondire ulteriormente, in questa fase, la distinzione tra le due versioni del test di consistenza modale per E. Ritorniamo sull'argomento parlando della procedura di decisione per la logica EN nelle sezioni 4.3.3 e 4.5.1 dove mostriamo che la seconda procedura può essere in qualche caso più efficiente della prima.

4.3.2 La logica EM

Il test per la logica EM si ricava immediatamente dall'algoritmo presentato nella sezione 3.2.2. La procedura presentata in figura 4.5 realizza il test

[1]Per disgiunzione in senso lazy si intende che solo se il primo dei disgiunti ritorna *False* viene invocato il secondo

di consistenza per un assegnamento così come è espresso dal lemma 3.1. Ricordiamo che una formula del tipo $\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma$ è consistente in EM a patto che non vi siano formule α e formule β tali che $\alpha \to \beta$.

Figura 4.5: Test di consistenza per EM

---

function $\text{EM}\textsc{sat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      if not $\text{EM}\textsc{sat}(\alpha_i \wedge \neg\beta_j)$ **then return** *False*;
  return *True*.

Si noti la somiglianza della procedura $\text{EM}\textsc{sat}_A$ con quella descritta nella sezione 4.3.1 a proposito della logica E. In particolare ritroviamo i due loop annidati che scandiscono tutte le coppie α e β in cui, per ciascuna coppia, viene invocato il test ricorsivo sulla soddisfacibilità della formula $\alpha\wedge\neg\beta$. Nel caso in cui il $\text{EM}\textsc{sat}(\alpha_i \wedge \neg\beta_j)$ fallisce, la procedura ritorna *False* mentre, se tutti i test hanno successo, conclude che l'assegnamento è consistente e pertanto ritorna *True*.

Vediamo di capire il meccanismo logico che governa questa procedura di decisione. Se il test ricorsivo di soddisfacibilità fallisce, allora la formula $\alpha \to \beta$ è valida in EM e questo ci consente di inferire che anche $\Box\alpha \to \Box\beta$ è valida in EM. Formalmente, è proprio la regola di inferenza RM che caratterizza la logica EM (si veda la sezione 2.4) che ci permette di giungere a tale conclusione. In definitiva, l'assegnamento non è consistente in quanto non è possibile, contemporaneamente, conoscere α e non conoscere β dal momento in cui sappiamo che la conoscenza di α implica la conoscenza di β.

Come accadeva per la logica E le cause dell'insoddisfacibilità di un assegnamento vanno ricercate nel confronto tra conoscenza e non conoscenza, cioè nel confronto tra formule α e formule β. Pertanto, anche in EM formule del tipo $\varphi_1 = \Box\bot$ oppure $\varphi_2 = \neg\Box\top$ sono perfettamente soddisfacibili. La differenza sostanziale rispetto ad E è una maggiore "potenza" del ragionamento epistemico. Vediamo cosa significa questa affermazione con il seguente esempio. Supponiamo di avere una formula $\varphi = \Box(P_0\wedge P_1)\wedge\neg\Box P_0$. Si noti che data la particolare struttura di φ essa coincide con il suo assegnamento. Dato che $(P_0 \wedge P_1) \to P_0$, ma non viceversa, questa formula è soddisfacibile in E mentre risulta insoddisfacibile in EM. All'interno delle strutture $\mathcal{E}_M$ è pertanto impossibile conoscere una congiunzione di due formule senza che questo implichi la conoscenza di ciascuna delle due. Al riguardo, in [Vardi, 1989] si dice che l'aggiunta dell'assioma M e quindi il passaggio dalla classe $\mathcal{E}$ alla classe $\mathcal{E}_M$, ci consente di fare *inferenze* basate sulla conoscenza. (ricordiamo $RM : \varphi \to \psi \Rightarrow \Box\varphi \to \Box\psi$). In questo senso la logica EM esprime "facoltà mentali" superiori a quelli della logica E.

4.3.3 La logica EN

Come avevamo anticipato in 4.3.1 parlando della logica E, le diverse strategie utilizzate per eseguire il test di consistenza modale (procedure in figura 4.3 e 4.4) potevano avere un impatto sulla forma del test per EN. Questo è dovuto al fatto che, similmente a quanto accadeva per l'algoritmo presentato in 3.3.1, la procedura che gestisce l'inferenza modale in EN è una estensione di quelle viste per E.

Figura 4.6: Test di consistenza per EN - Versione 1

function ENsat$_A(\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      if not ENSAT$(\alpha_i \leftrightarrow \neg\beta_j)$ **then return** *False*;
  foreach conjunct $\Box\beta_j$ **do**
    if not ENSAT$(\neg\beta_j)$ **then return** *False*;
  return *True*.

Consideriamo la procedura presentata in figura 4.6, in cui abbiamo sovrapposto al test per E (la versione con equivalenza logica) il test relativo all'assioma N. Anzitutto, notiamo che i primi due loop annidati vengono eseguiti solo se vi sono sia formule α che formule β. Questo corrisponde ad eseguire un test di consistenza dell'assegnamento per la logica E. In ogni caso, anche se non ci sono formule α, viene eseguito il secondo loop in cui, per ciascuno dei β, viene invocato il test ricorsivo sulle formule $\neg\beta$. Nel caso in cui almeno uno di questi test fallisca, ENSAT$_A$ ritorna *False* mentre se $\neg\beta$ è soddisfacibile per tutti i β l'assegnamento è consistente e la procedura ritorna *True*. Infine, nel caso in cui non vi siano formule β, l'assegnamento è banalmente consistente.

Soffermiamoci, come avevamo fatto nella sezione 4.3.2 per EM, sulla differenza tra E e EN. In particolare cerchiamo di capire il motivo per cui è stato necessario aggiungere un test relativo alla validità dei β. Come abbiamo visto nella sezione 2.4 la logica EN è caratterizzata sia dalla regola RE sia dall'assioma N. È proprio quest'ultimo che rende necessario il test sui soli β. Infatti, se in E era perfettamente lecito non conoscere le formule valide, in EN la formula $\neg\Box\top$ risulta inconsistente dato che, per N, da $\top$ segue $\Box\top$. In altre parole la possibilità di inferenza epistemica si estende sino a comprendere la necessaria conoscenza delle formule valide. La capacità che si acquisisce aggiungendo al sistema E il fatto che $\varphi \Rightarrow \Box\varphi$ e cioè il passaggio dai modelli in $\mathcal{E}$ a quelli in $\mathcal{E}_N$, viene chiamata in [Vardi, 1989] conoscenza di fatti *autoevidenti*.

Un altro punto degno di nota è che il secondo test si rende necessario comunque, perché sapere che la formula $\alpha \leftrightarrow \neg\beta$ è soddisfacibile in EN non è di per sè sufficiente a concludere che sia tale anche $\neg\beta$. In pratica non siamo in grado di utilizzare l'informazione circa la soddisfacibilità di $\neg\beta$ che ci viene dal test per E.

Figura 4.7: Test di consistenza per EN - Versione 2

---

function ENSAT$_A$($\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma$)
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      if not ENSAT($\alpha_i \wedge \neg\beta_j$)
        then if not ENSAT($\neg\alpha_i \wedge \beta_j$) **or**
              not ENSAT($\neg\beta_j$) **then return** *False*;
  return *True*.

Al contrario, la procedura in figura 4.7 mi consente di sfruttare una tale informazione. Infatti è evidente che se $\alpha \wedge \neg\beta$ è soddisfacibile allora β non è una formula valida nei modelli per EN e quindi posso evitare di ripetere il test solo per $\neg\beta$. D'altronde, se il primo test fallisce sono comunque costretto ad eseguire due test separati (solo se $\neg\alpha\wedge\beta$ è insoddisfacibile posso fermarmi subito) ed è anche possibile che il test relativo a $\neg\beta$ venga ripetuto più volte per lo stesso β. In ogni caso, come vedremo nella sezione 4.5, è possibile combinare in maniera efficiente i due test ricorsivi utilizzando la seconda versione di ENSAT$_A$.

4.3.4 La logica EMN

Similmente a quello che avevamo fatto per EN, costruiremo il test di consistenza per EMN estendendo la procedura per EM analizzata nella sezione 4.3.2.

La procedura presentata in figura 4.8 consta di due parti tra di loro mutuamente esclusive. Nel caso in cui l'assegnamento consti sia di formule α che di formule β, vengono eseguiti i due loop annidati che ricalcano fedelmente il test di consistenza per EM. Nel caso in cui questa prima serie di test abbia successo, siamo anche garantiti circa la soddisfacibilità delle formule $\neg\beta_j$ per ogni j e quindi non è necessario rieseguire lo specifico test per l'assioma N. Nel caso in cui qualche test ricorsivo su $\alpha \wedge \neg\beta$ fallisca,

Figura 4.8: Test di consistenza per EMN

function EMNsat$_A(\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma)$
  if {there is any conjunct $\Box\alpha_i$}
    then foreach conjunct $\Box\alpha_i$ **do**
      foreach conjunct $\Box\beta_j$ **do**
        if not EMNSAT$(\alpha_i \wedge \neg\beta_j)$ **then return** *False*;
    else foreach conjunct $\Box\beta_j$ **do**
      if not EMNSAT$(\neg\beta_j)$ **then return** *False*;
  return *True*.

non è comunque necessario eseguire il secondo loop: se l'assegnamento è già inconsistente in EM è tale anche in EMN. Solo nel caso in cui non vi siano formule α nell'assegnamento è necessario lanciare il loop sui soli β. Infine nel caso in cui non vi siano formule β, l'assegnamento è sempre banalmente consistente.

Da un punto di vista puramente algoritmico, il ragionamento realizzato dalla procedura EMNSAT$_A$ non sembrerebbe essere particolarmente interessante, proprio per il fatto che è una semplce combinazione del ragionamento monotono espresso da EM con l'aggiunta della conoscenza di fatti autoevidenti imposta dall'assioma N. Peraltro, cosa più interessante, EMNSAT è la procedura di decisione più "potente" dal punto di vista deduttivo, il cui test di soddisfacibilità è comunque NP-complete. Sembra quindi che EMNSAT possa offrirci un buon rapporto tra le elevate "prestazioni" del ragionamento epistemico e il "costo" di quest'ultimo in termini computazionali, tanto che vale la pena soffermarsi per cercare di capire meglio cosa significhi combinare EM ed EN all'interno di una stessa logica.

Anche se non lo abbiamo ancora dimostrato formalmente, ammettiamo che la procedura EMNSAT$_A$ sia parte di una procedura di decisione corretta e completa rispetto alla classe di strutture $\mathcal{MS}$che abbiamo definito come $\mathcal{E}_{MN}$. Questo tipo di modello della conoscenza viene chiamato in [Fagin *et al.*, 1995] *local reasoning structure*[2] e, tra l'altro, viene descritto come un modello particolarmente adatto a rappresentare le credenze di agenti umani.

Vediamo, sotto forma di semplici esempi, quali tipi di inferenza epistemica possiamo realizzare nelle local reasoning structure di cui EMNSAT rispecchia le caratteristiche. In [Fagin *et al.*, 1995] leggiamo che, in generale, è possibile che un agente umano creda fatti inconsistenti, cioè non è vero che la conoscenza di una fatto implica la non conoscenza del suo contrario.

[2]in assenza di un'appropriata terminologia in italiano ci rifacciamo, nel seguito, alla terminologia anglosassone

Rappresentiamo questa situazione con $\varphi = \Box\psi \wedge \Box\neg\psi$, aspettandoci che EMN$\textsc{sat}_A(\varphi)$ ritorni *True* e in effetti è così, perché φ, che coincide con il suo assegnamento, contiene soltanto formule α ma nessuna formula β.

Sempre in [Fagin *et al.*, 1995], si dice che possiamo consentire che un agente umano creda in fatti inconsistenti ($\Box(\psi \wedge \neg\psi)$ è ancora tale che EMN$\textsc{sat}_A(\varphi) = \mathit{True}$) ma non siamo, di solito, disposti ad accettare agenti che non credono in cose ovvie e infatti con $\varphi = \neg\Box(\psi \vee \neg\psi)$ risulta EMN$\textsc{sat}_A(\varphi) = \mathit{False}$ perché $\neg\beta = \psi \wedge \neg\psi$ è chiaramente insoddisfacibile.

Gli agenti hanno l'abilità di eseguire inferenze e infatti $\varphi = \Box(\psi_1 \wedge \psi_2) \wedge \neg\Box\psi_1$ è tale che EMN$\textsc{sat}_A(\varphi) = \mathit{False}$ perché $\alpha = \psi_1 \wedge \psi_2$ e $\beta = \psi_1$ sono tali che il test su $\alpha \wedge \neg\beta$ fallisce. Infine non siamo disposti ad accettare situazioni per cui un agente non crede nelle inferenze che riesce a fare e infatti $\varphi = \neg\Box(\Box(\psi_1 \wedge \psi_2) \rightarrow \Box\psi_1)$ non è soddisfacibile in EMN, mentre lo sarebbe stata sia in EN (M non è valido in EN), sia in EM ($\neg\Box\top$ è soddisfacibile in EM).

Notiamo che gli esempi sono stati scelti in modo da denotare l'inclusione successiva $\mathcal{E} \subset \mathcal{E}_N \subset \mathcal{E}_{MN}$ proprio per mostrare che l'aggiunta di proprietà restringe le classi di strutture sino ad arrivare a quelle rispetto alle quali EMN$\textsc{sat}$ risulta corretto e completo.

4.3.5 La logica EC

Nell'affrontare il problema della costruzione di una procedura di decisione per EC dobbiamo tenere ben presente il fatto che, come enunciato nel teorema 3.3 nella sezione 3.2.3, i decisori per questa logica e per tutte quelle i cui modelli sono sottoinsiemi di $\mathcal{E}_C$ sono destinati ad avere un complessità *superiore* a NP (abbiamo parlato al riguardo di problemi PSPACE-hard). Pertanto, anche in questa fase per alcuni aspetti preliminare alla costruzione dei decisori veri e propri, diventano cruciali alcune considerazioni sull'efficienza delle procedure. Rimandiamo comunque alla sezione 4.5 le possibili soluzioni all'intrinseca difficoltà dei problemi affrontati.

Come avevamo visto nella sezione 3.2.3 quando avevamo presentato l'algoritmo di alto livello per l'inferenza modale in EC (lemma 3.3), il controllo di consistenza è costituito da due diversi test ricorsivi. Dato l'assegnamento $\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma$ abbiamo i seguenti casi:

1. controlliamo per quali α_i la formula $\neg\alpha_i \wedge \beta_j$ è soddisfacibile in $\mathcal{E}_C$. Se questo vale per tutti gli α_i, allora l'assegnamento è consistente;

2. altrimenti, se la formula $\bigwedge_i \alpha_i \wedge \neg\beta_j$ è soddisfacibile nei modelli $\mathcal{E}_C$ per tutti gli α_i che non soddisfano il test al punto 1, l'assegnamento è consistente.

Ovviamente, nel caso in cui ambo i test falliscano l'assegnamento non è consistente in EC, mentre se non vi sono formule α o formule β l'assegnamento è banalmente consistente.

Nel costruire la procedura di decisione per EC abbiamo tenuto conto del fatto che il primo test è meno "duro" dal punto di vista computazionale rispetto al secondo. Riprendendo la discussione relativa al teorema 3.3 condotta nella sezione 3.2.3, ricordiamo che la maggiore difficoltà computazionale era originata dal fatto che non si confrontavano coppie di sottoformule, bensì insiemi di sottoformule. Pertanto, abbiamo costruito la procedura $\text{EC}\textsc{sat}_A$ in modo che il test al punto 1 della lista precedente venga comunque eseguito per primo.

Come si può vedere in figura 4.9, la prima parte della procedura $\text{EC}\textsc{sat}_A$ compie un test di soddisfacibilità su coppie di sottoformule α e β, per cui, se fosse l'unico test possibile, varrebbero anche per EC i risultati di complessità ottenuti per E e per EM. La scelta di eseguire il test ricorsivo su coppie per primo, pur non cambiando la complessità della procedura nel caso peggiore, dovrebbe garantirci, in media, tempi di esecuzione più rapidi rispetto al caso in cui il test $\bigwedge_i \alpha_i \wedge \neg\beta_j$ venga sempre eseguito per primo.

Figura 4.9: Test di consistenza per EC

---

function $\text{EC}\textsc{sat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  if {there is any conjunct $\Box\alpha_i$}
    then foreach conjunct $\Box\beta_j$ **do**
      $\Delta = \{\alpha_1, \ldots, \alpha_n\}$;
      foreach conjunct $\Box\alpha_i$ **do**
        if $\text{EC}\textsc{sat}(\neg\alpha_i \wedge \beta_j)$ **then** $\Delta = \Delta - \{\alpha_i\}$;
      if $\Delta = \emptyset$ **then return** *True*;
      if not $\text{EC}\textsc{sat}(\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j)$ **then return** *False*;
  return *True*.

$\text{EC}\textsc{sat}_A$, dopo aver controllato la presenza di formule α, inizia un loop che scandisce tutti i β. Ad ogni iterazione viene costruito l'insieme Δ di tutte le formule α presenti nell'assegnamento. Lo scopo di Δ è quello di tenere traccia di tutti gli α per cui il test ricorsivo $\text{EC}\textsc{sat}(\neg\alpha \wedge \beta)$ non è verificato. Al termine del loop più interno, se Δ è vuoto allora è possibile concludere che l'assegnamento è consistente. Viceversa, è necessario formare la congiunzione di ttuti gli α che sono rimasti in Δ e provare con il test $\text{EC}\textsc{sat}(\bigwedge_{\alpha_i \in \Delta} \Box\alpha_i \wedge \neg\Box\beta_j)$. Se anche questo fallisce, l'assegnamento è inconsistente e la procedura si interrompe ritornando *False*. Se l'insieme dei due test ricorsivi ha sempre successo, allora la procedura conclude restituendo *True*.

Cerchiamo di capire in che senso la procedura $\text{EC}\textsc{sat}$ estende le possibilità di inferenza rispetto a $\text{E}\textsc{sat}$ con il seguente esempio. La formula

$\varphi = \Box P_0 \wedge \Box P_1 \wedge \neg\Box(P_0 \wedge P_1)$ è chiaramente soddisfacibile in $\mathcal{E}$ in quanto $(P_0 \wedge P_1) \to P_1$ e $(P_0 \wedge P_1) \to P_0$ ma non viceversa. D'altronde, provando con $\text{EC}\textsc{sat}_A$ vediamo che $\neg P_0 \wedge P_0 \wedge P_1$ e $\neg P_1 \wedge P_0 \wedge P_1$ sono inconsistenti ed anche il test $P_0 \wedge P_1 \wedge \neg(P_0 \wedge P_1)$ è destinato a fallire. Questo significa che in EC non è possibile conoscere due fatti senza conoscere automaticamente la loro congiuzione. Nelle parole di [Vardi, 1989], EC è la più piccola logica i cui agenti sono in grado *combinare* fatti. Per inciso, visto il risultato 3.3, sembra che la possibilità di compiere questa inferenza epistemica sia, da sola, responsabile dell'aumento della complessità da NP a PSPACE.

Chiariamo ora il ruolo giocato dai due differenti test di ricorsivi all'interno della procedura $\text{EC}\textsc{sat}_A$. Sappiamo che la formula $\varphi = \Box(P_0 \wedge P_1) \wedge \neg\Box P_0$ deve essere soddisfacibile in $\mathcal{E}_C$ in quanto agli agenti non è assegnata l'abilità di condurre inferenze. Notiamo che è il primo test a sancire la soddisfacibilità dell'unico assegnamento possibile per φ, in quanto $\neg(P_0 \wedge P_1) \wedge P_0$ è soddisfacibile, mentre il secondo test e cioè $(P_0 \wedge P_1) \wedge \neg P_0$, dà origine ad una formula banalmente insoddisfacibile. D'altro canto consideriamo la formula $\psi = \Box(P_0 \vee P_2) \wedge \Box(P_1 \vee P_2) \wedge \neg\Box P_2$. Notiamo che ψ non viola il criterio di consistenza in EC in quanto $(P_0 \vee P_2) \wedge (P_1 \vee P_2)$ non implica affatto P_2. Se $\text{EC}\textsc{sat}$ eseguisse solo i test $\neg\alpha \wedge \beta$ allora dato che sia $\neg(P_0 \vee P_2) \wedge P_2$ che $\neg(P_1 \vee P_2) \wedge P_2$ sono insoddisfacibili, avremmo un risultato non corretto. Invece, dato che $(P_0 \vee P_2) \wedge (P_1 \vee P_2) \wedge \neg P_2$ è soddisfacibile, $\text{EC}\textsc{sat}$ dichiara soddisfacibile la formula ψ.

4.3.6 La logica ECN

La procedura di decisione per ECN è un'estensione piuttosto semplice di quella presentata nella sezione 4.3.5 per la logica EC.

Figura 4.10: Test di consistenza per ECN

---

function $\text{ECN}\textsc{sat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\beta_j$ **do**
    $\Delta = \{\alpha_1, \ldots, \alpha_n\}$;
    foreach conjunct $\Box\alpha_i$ **do**
      if $\text{ECN}\textsc{sat}(\neg\alpha_i \wedge \beta_j)$ **then** $\Delta = \Delta - \{i\}$;
    if not $\text{ECN}\textsc{sat}(\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j)$ **then return** *False*;
  return *True*.

Come si può notare in figura 4.10, abbiamo rimosso la condizione che decretava come soddisfacibile un assegnamento in cui non compariva alcuna formula α, in quanto questa proprietà non è più vera in ECN. Abbiamo rimosso anche la condizione per cui se Δ risultava vuoto al termine del loop sugli α l'assegnamento poteva essere dichiarato soddisfacibile. Infatti, sapere che per tutti gli α la formula $\neg\alpha \wedge \beta$ è soddisfacibile non implica affatto che anche $\neg\beta$ sia tale.

Il funzionamento di ECNsat$_A$ è quindi modificato per tenere conto del test di consistenza aggiuntivo relativo all'assioma N. In particolare, ritroviamo il loop che scandisce tutti i β e il loop più interno sui soli α. Al termine di quest'ultimo viene comunque eseguito il secondo test che, nel caso in cui $\Delta = \emptyset$ si riduce ovviamente a ECNsat$(\neg\beta)$. Infine, un assegnamento che consti di soli α è ancora banalmente soddisfacibile.

La procedura ECNsat combina le proprietà delle logiche EC ed EN. Avevamo già visto per EMN che l'introduzione dell'assioma N faceva si che risultassero insoddisfacibili formule per cui si affermava la non conoscenza delle proprie possibilità di inferenza. La stessa cosa accade per ECN in quanto la formula $\varphi = \neg\Box(\Box P_0 \wedge \Box P_1 \rightarrow \Box(P_0 \wedge P_1))$ risulta insoddisfacibile in ECN mentre è soddisfacibile separatmente in EC (come tutte le formule del tipo $\neg\Box\top$) e in EN (in quanto C non è uno schema valido in $\mathcal{E}_N$).

Operativamente, una formula come φ determina un unico test ricorsivo di tipo ECNsat$(\neg(\Box P_0 \wedge \Box P_1 \rightarrow \Box(P_0 \wedge P_1))$ che possiamo riscrivere come ECNsat$(\Box P_0 \wedge \Box P_1 \wedge \neg\Box(P_0 \wedge P_1))$. All'invocazione di ECNsat$_A$ seguirà necessariamente *False* in quanto sia $\neg P_0 \wedge (P_0 \wedge P_1)$ sia $\neg P_1 \wedge (P_0 \wedge P_1)$ risultano essere insoddisfacibili (dal primo loop sugli α) e anche $\neg P_0 \wedge \neg P_1 \wedge \neg(P_0 \wedge P_1)$ è insoddisfacibile (dal secondo test su $\bigwedge_{i \in B} \alpha_i \wedge \neg\beta$).

4.3.7 La logica R (EMC)

Il controllo di consistenza modale in R potrebbe essere visto come una combinazione delle procedure EMsat$_A$ e ECsat$_A$. D'altronde come abbiamo visto nella sezione 3.3.2 risultava più conveniente derivare l'algoritmo nei termini della chiusura sotto la regola RR. Lo stesso discorso vale per la procedura Rsat che è derivata direttamente dal lemma 3.7 in maniera indipendente dalle procedure di decisione delle logiche che ne includono i modelli.

Figura 4.11: Test di consistenza per R

function Rsat$_A$($\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma$)
  if {there is any conjunct $\Box\alpha_i$}
    then foreach conjunct $\Box\beta_j$ **do**
        if not Rsat($\bigwedge_i \alpha_i \wedge \neg\beta_j$) **then return** *False*;
  return *True*.

La procedura presentata in figura 4.11 ha un comportamento molto semplice. Innanzi tutto viene controllata la presenza di formule α, dato che, come accade sia in EC sia in EM, se non vi sono atomi modali assegnati a vero, l'assegnamento è sempre banalmente consistente. Se vi sono formule α, inizia un loop che scandisce tutti i β ed esegue il test ricorsivo $\textsc{Rsat}(\bigwedge_i \alpha_i \wedge \neg\beta)$. Se per qualche β il test fallisce la procedura ritorna *False*, mentre ritorna *True* nel caso contrario. Se non ci sono formule β nell'assegnamento esso è banalmente consistente per cui il loop non inizia e $\textsc{Rsat}_A$ torna immediatamente *True*.

Dal punto di vista logico R ha la particolarità di essere differente da K soltanto per il fatto che lo schema N non è valido nella classe dei quasi filtri rispetto ai quali $\textsc{Rsat}$ è un decisore corretto e completo. In realtà questa sembra essere una "distanza" molto minore rispetto a quella che separava EMN o ECN dalla realizzazione di un'inferenza modale onnisciente.

Cerchiamo, attraverso il ragionamento realizzato in $\textsc{Rsat}$ di capire meglio questa differenza. Consideriamo la formula $\varphi = \Box(P_0 \wedge P_1) \leftrightarrow \Box P_0 \wedge \Box P_1$ che esprime il fatto che è indifferente conoscere due fatti separatamente o conoscere la loro congiunzione. Sappiamo che φ deve risultare valida nelle strutture $\mathcal{E}_{MC}$ e, per dimostrarlo, invochiamo $\textsc{Rsat}$ sulle formule:

$$\Box(P_0 \wedge P_1) \wedge \neg(\Box P_0 \wedge \Box P_1) \tag{4.3}$$

$$\neg\Box(P_0 \wedge P_1) \wedge (\Box P_0 \wedge \Box P_1) \tag{4.4}$$

Ovviamente stiamo applicando il fatto che $\vdash_R \varphi \Leftrightarrow \sim\textsc{Rsat}(\neg\varphi)$. Possibili assegnamenti (parziali) per 4.3 sono $\mu_1 = \Box(P_0 \wedge P_1) \wedge \neg\Box P_0$ e $\mu_2 \Box(P_0 \wedge P_1) \wedge \neg\Box P_0$. Entrambi contengono almeno un atomo modale assegnato a vero, ma falliscono entrambi il test ricorsivo di soddisfacibilità $\bigwedge_i \alpha_i \wedge \neg\beta$. La stessa cosa succede con 4.4 dato che la formula coincide con l'assegnamento e quest'ultimo è tale che $\textsc{Rsat}$ ritorna *False*.

$\textsc{Rsat}$ mi ha consentito di provare il fatto che φ è una formula valida in R ed è corretto visto che φ è la congiunzione delgi scehmi M e C. La limitazione di R è quella per cui gli agenti non sono obbligati a conoscere le consenguenze delle proprie inferenze. Infatti $\psi = \neg\Box\varphi$ è tale che $\textsc{Rsat}(\psi) = \textsc{Rsat}_A(\psi) = \mathit{True}$ in quanto ψ contiene soltanto un atomo modale assegnato a falso per cui il loop sui β non inizia. Questo fa di $\textsc{Rsat}$ una procedura che, comunque, realizza un ragionamento epistemico non onnisciente.

4.3.8 La logica K ($EMCN$)

L'ultima procedura che presentiamo in questa sezione è quella per K. La procedura in figura 4.12 non è altro che una semplice restrizione al caso monomodale dell'equivalente decisore già descritto in [Giunchiglia e Sebastiani, 1996a], in [Giunchiglia e Sebastiani, 1996b] e, più approfonditamente, in [Sebastiani, 1997]. Ovviamente si tratta di una realizzazione diretta del lemma 3.8 proposto nella sezione 3.3.3 e dato che differisce dall'analoga procedura per R solo per il test sulla presenza delle formule α nell'assegnamento, non ci soffermiamo sulla sua descrizione.

Figura 4.12: Test di consistenza per K

---

function $\textsc{Ksat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\beta_j$ **do**
    if not $\textsc{Rsat}(\bigwedge_i \alpha_i \wedge \neg\beta_j)$ **then return** *False*;
  return *True*.

Ci proponiamo invece di capire in che senso KSAT realizza un ragionamento logicamente onnisciente. Consideriamo allora la formula $\varphi = (\Box P_0 \wedge \Box(P_0 \to P_1)) \to \Box P_1$ che sappiamo essere valida in $\mathcal{K}$ (i modelli di K) e mostriamo che la negazione di φ è soddisfacibile in $\mathcal{E}_{MN}$. Risulta infatti $\neg\varphi = \Box P_0 \wedge \Box(P_0 \to P_1) \wedge \neg\Box P_1)$ e visto che $P_0 \wedge \neg P_1$ e $(P_0 \to P_1) \wedge \neg P_1$ sono soddisfacibili in $\mathcal{E}_{MN}$ anche $\neg\varphi$ è soddisfacibile.

D'altronde, $\neg\varphi$ risulta insoddisfacibile sia in $\mathcal{E}_{MC}$ sia in $\mathcal{E}_{CN}$ da cui segue immediatamente che φ è valida in entrambe le classi di modelli. Comunque è sufficiente complicare di poco l'inferenza per mostrare come K sia più potente delle altre due logiche. Se $\vdash_K \varphi$ allora anche $\vdash_K \Box\varphi$ e quindi $\neg\Box\varphi$ è insoddisfacibile in $\mathcal{K}$. Naturalmente lo stesso non vale per $\mathcal{E}_{MC}$ in cui $\neg\Box\varphi$ è perfettamente soddisfacibile: in assenza dell'assioma N gli agenti non sono costretti a conoscere le conseguenze del proprio ragionamento.

Nel caso di $\mathcal{E}_{CN}$ $\neg\Box\varphi$ è insoddisfacibile, ma è sufficiente considerare $\psi = \varphi[P_0 \wedge P_1/P_0] = \Box(P_0 \wedge P_1) \to \Box P_1$ la cui negazione è soddisfacibile in $\mathcal{E}_{CN}$ e per cui risulta $\vdash_K \psi$.

Siamo quindi riusciti a trovare tre esempi di formule e cioè φ, $\Box\varphi$, e ψ che risultano valide in K e la cui negazione è soddisfacibile, rispettivamente, in $\mathcal{E}_{MN}$, in $\mathcal{E}_{MC}$ e in $\mathcal{E}_{CN}$. Quindi abbiamo mostrato che nessuna delle tre classi di strutture possiede separatamente tutte le capacità di inferenza epistemica di cui sono dotate le strutture $\mathcal{K}$.

4.4 Dimostrazione di correttezza e completezza

Affrontiamo in questa sezione la prova di correttezza e completezza per i decisori modali. In questa prova faremo riferimento ai decisori ottenuti combinando lo schema generale presentato nella sezione 4.2 con le procedure relative all'inferenza modale presentate nella sezione 4.3.

La prova fonda la sua validità sui seguenti lemmi:

- il lemma 4.1 stabilisce che la profondità modale di un qualsiasi assegnamento μ per una qualsiasi formula $\varphi \in \mathcal{L}$ è la stessa di φ;

- il lemma 4.2 stabilisce una relazione tra il valore in uscita dalla procedura ΛSAT e il valore restituito dalla procedura ΛSAT_A;
- il lemma 4.3 sancisce il fatto che la procedura ΛSAT_{DP} genera correttamente la parte proposizionale degli assegnamenti (quella a cui ci siamo sempre riferiti con γ).
- uno fra i lemmi esposti nelle sezioni 3.2, 3.3 del capitolo precedente che stabiliscono il legame tra ragionamento proposizionale e ragionamento modale.

Nel seguito utilizzeremo i termini ΛSAT per riferirci esclusivamente alla procedura implementata e utilizzeremo $\nvdash_\Lambda \neg\varphi$ per denotare il fatto che φ è soddisfacibile.

Cominciamo con le proprietà che ci consentono di stabilire un legame tra la profondità modale di una formula e quella di un assegnamento. In particolare la seguente proprietà formalizza il fatto che la profondità modale di una formula φ equivale alla massima profondità modale degli atomi che la compongono (si veda def. 2.7 nella sezione 2.2)

Proprietà 4.1 *Data $\varphi \in \mathcal{L}$ risulta*

$$\max\{depth(a) \mid a \in Atom(\varphi)\} = depth(\varphi) \tag{4.5}$$

Dimostrazione - Dimostriamo la proprietà per induzione completa sulla profondità sintattica di φ; impostiamo l'ipotesi induttiva come segue:

> Sia D tale che $D = dtree(\varphi)$; allora per $dtree(\psi) < D$, $\max\{depth(a) \mid a \in Atom(\psi)\} = depth(\psi)$

con *depth* e *dtree* dati, rispettivamente, dalle definizioni 2.2 e 2.3. Sono dati i seguenti casi: [3]

[NEG] sia $\varphi = \neg\psi$ con $\psi \in \mathcal{L}$; in questo caso, in base alle definizione 2.7, risulta $Atom(\varphi) = Atom(\psi)$ e, in base alla definizione 2.3, $dtree(\psi) < L$; vale quindi la seguente uguaglianza:

$$\begin{gathered}\max\{depth(a) \mid a \in Atom(\varphi)\} = \\ \max\{depth(a) \mid a \in Atom(\psi)\} = depth(\psi)\end{gathered} \tag{4.6}$$

e, dalla definizione 2.2, $depth(\psi) = depth(\varphi)$ da cui la tesi;

[CON] sia $\varphi = \xi \wedge \psi$ con $\xi, \psi \in \mathcal{L}$; dalla definizione 2.7 risulta $Atom(\varphi) = Atom(\xi) \cup Atom(\psi)$ e quindi:

$$\max\{depth(a) \mid a \in Atom(\varphi)\} = \max(X, P) \tag{4.7}$$

[3] I casi in cui φ è un atomo fanno si che $dtree(\varphi) = 1$ e pertanto sono compresi nell'ipotesi induttiva

dove X e P sono dati da:

$$X = \max\{depth(a) \mid a \in Atom(\xi)\} \tag{4.8}$$
$$P = \max\{depth(a) \mid a \in Atom(\psi)\} \tag{4.9}$$

In base alla definizione 2.3 si ha che $dtree(\xi) < L$ e $dtree(\psi) < L$; quindi, applicando l'ipotesi induttiva, risulta $X = depth(\xi)$ e $P = depth(\psi)$; d'altronde se consideriamo la definizione 2.2 risulta che $depth(\varphi) = \max(depth(\xi), depth(\psi))$ da cui la tesi.

Dalla proprietà 2.3 enunciata nella sezione 2.3.3, sappiamo che ogni assegnamento μ può essere visto come una formula $\mu \in \mathcal{L}$. In particolare, dato che μ assegna valori di verità agli atomi di una formula, riguardando μ come tale dobbiamo ritrovare ancora lo stesso insieme di atomi. Formalizziamo questo fatto con la seguente:

Proprietà 4.2 *Dati $\varphi \in \mathcal{L}$ e un qualsiasi assegnamento μ per φ, risulta $Atom(\varphi) = Atom(\mu)$.*

Utilizzando le proprietà 4.1 e 4.2 siamo allora in grado di concludere che la profondità modale di un assegnamento equaglia quella della corrispondente formula. Formalmente:

Lemma 4.1 (Profondità modale di un assegnamento) *Data $\varphi \in \mathcal{L}$, qualsiasi assegnamento μ per φ è tale che $depth(\varphi) = depth(\mu)$*

Dimostrazione - In base alla proprietà 4.1 risulta:

$$depth(\varphi) = \max\{depth(a) \mid a \in Atom(\varphi)\} \tag{4.10}$$
$$depth(\mu) = \max\{depth(a) \mid a \in Atom(\mu)\} \tag{4.11}$$

Essendo (proprietà 4.2) $Atom(\varphi) = Atom(\mu)$ risulta $depth(\mu) = depth(\varphi)$

I lemmi che enunciamo adesso sono una consenguenza diretta del teorema 4.1 che avevamo presentato nella sezione 4.1. Il primo si riferisce ad una proprietà strutturale del codice ΛSAT e precisamente:

Lemma 4.2 (Relazione tra ΛSAT e ΛSAT$_A$) *Data $\varphi \in \mathcal{L}$, $\Lambda\text{SAT}(\varphi) = True$ se e solo se $\Lambda\text{SAT}_{DP}(\varphi, \top) = True$ se e solo se esiste almeno un assegnamento μ per φ, ed è tale che $ESAT_A(\mu) = True$.*

Il secondo è l'affermazione che l'inferenza proposizionale realizzata dalla procedura ΛSAT è corretta:

Lemma 4.3 (Soddisfacibilità di γ) *Data $\varphi \in \mathcal{L}$, non è possibile che ΛSAT costruisca un assegnamento μ tale $\vdash_\Lambda \neg\gamma$.*

Possiamo ora enunciare il seguente risultato:

Teorema 4.2 (Correttezza e completezza) *Data una formula* $\varphi \in \mathcal{L}$, $\Lambda\text{SAT}(\varphi) = True$ *se e solo se* $\not\vdash_\Lambda \neg\varphi$.

Dimostare formalmente questo risultato è dato dal fatto che, almeno in principio, avremmo bisogno di otto diverse prove. Per le logiche come EM, EMC e K per cui esistono algoritmi di decisione a tableau provati corretti e completi (vedi [Fitting, 1983] al riguardo), la dimostrazione è banale una volta considerato il teorema 4.1 e la forte analogia tra le procedure per l'inferenza modale e le corrispondenti regole del metodo tableau. Per quanto riguarda le altre logiche, per motivi di spazio e di comprensibilità abbiamo scelto di presentare la sola dimostrazione per E. In ogni caso, le altre prove non sono concettualmente diverse da quella presentata, che può essere facilmente adattata inserendo nella struttura generale uno dei lemmi (algoritmi) presentati nel capitolo 3.
Dimostrazione "$\Rightarrow$" - La correttezza si dimostra per induzione sulla profondità modale della formula φ. Impostiamo l'ipotesi induttiva come segue:

> Sia D tale che $D = depth(\varphi)$; allora, per $depth(\psi) < D$, se $\text{ESAT}(\psi) = True$ allora $\not\vdash_E \neg\psi$.

Sia $\mathcal{A} = \{\mu_1, \ldots \mu_r\}$, l'insieme di tutti e soli i possibili assegnamenti per φ. In base alla proprietà 2.1 possiamo sempre scrivere $\vdash_E \varphi \leftrightarrow \bigvee_i^r \mu_i$ e quindi $\vdash_E \mu \rightarrow \varphi$ per ogni $\mu \in \mathcal{A}$. Supponiamo ora, per assurdo, che sia $\text{ESAT}(\varphi) = True$ mentre $\vdash_E \neg\varphi$. A partire da quest'ultima ipotesi, avremo che $\vdash_E \neg\mu$ per qualsiasi $\mu \in \mathcal{A}$. Possiamo esplicitare quest'ultimo risultato considerando un generico $\mu \in \mathcal{A}$:

$$\vdash_E \neg(\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma) \tag{4.12}$$

che, in base al lemma 3.1 (E-lemma), possiamo riscrivere come:

$$\text{non (per ogni } i,j.\ \not\vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{e} \quad \not\vdash_E \neg\gamma) \tag{4.13}$$

equivalente a:

$$\text{esistono } i,j.\ \vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{oppure} \quad \vdash_E \neg\gamma \tag{4.14}$$

Ora, visto che il risultato non dipende dall'assegnamento scelto in 4.12, possiamo concludere:

$$\text{per ogni } \mu \in \mathcal{A}. \text{ (esistono } i,j.\ \vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{oppure} \quad \vdash_E \neg\gamma) \tag{4.15}$$

Consideriamo ora le conseguenze di $\text{ESAT}(\varphi) = True$. In base al lemma 4.2 abbiamo che $\text{ESAT}(\varphi) = True$ implica che esiste $\mu \in \mathcal{A}$ tale che $\text{ESAT}_A(\mu) =$

True e, in base al lemma 4.3, per tale μ risulta anche $\not\vdash_E \neg\gamma$. Quindi possiamo scrivere:

$$\text{esiste } \mu.\ (\text{per ogni } i{,}j.\ \textsc{Esat}(\alpha_i \leftrightarrow \neg\beta j) = \mathit{True} \quad \text{e} \quad \not\vdash_E \neg\gamma) \tag{4.16}$$

Soffermiamoci sul particolare $\tilde{\mu} \in \mathcal{A}$ per cui vale:

$$\text{per ogni } i{,}j.\ \textsc{Esat}(\alpha_i \leftrightarrow \neg\beta j) = \mathit{True} \quad \text{e} \quad \not\vdash_E \neg\gamma \tag{4.17}$$

Ora, in base al lemma 4.1, è possibile applicare l'ipotesi induttiva e scrivere questo risultato come:

$$\text{per ogni } i{,}j.\ \not\vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{e} \quad \not\vdash_E \neg\gamma \tag{4.18}$$

equivalente, sempre per $\tilde{\mu}$, a:

$$\text{non esistono } i{,}j.\ \vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{e} \quad \not\vdash_E \neg\gamma \tag{4.19}$$

D'altronde, particolarizzando per $\tilde{\mu}$ il risultato 4.15 otteniamo:

$$\text{esistono } i{,}j.\ \vdash_E \alpha_i \leftrightarrow \beta_j \quad \text{oppure} \quad \vdash_E \neg\gamma \tag{4.20}$$

ma, considerando che vale anche 4.19, questo è chiaramente assurdo.

Dimostrazione "$\Leftarrow$" - La completezza si dimostra induzione completa sulla profondità modale della formula φ. L'ipotesi induttiva è la stessa formulata per il caso della correttezza. Sempre per assurdo, supponiamo che $\not\vdash_E \neg\varphi$ e $\textsc{Esat}(\varphi) = \mathit{False}$. In base al lemma 4.2, se quest'ultima affermazione è vera, allora non esiste un assegnamento μ tale che $\textsc{Esat}_A(\mu) = \mathit{True}$ e questo equivale a dire che $\textsc{Esat}_A(\mu) = \mathit{False}$ per ogni $\mu \in \mathcal{A}$. Per un generico μ risulta:

$$\text{esistono } i{,}j.\ \textsc{Esat}(\alpha_i \leftrightarrow \neg\beta_j) = \mathit{False} \tag{4.21}$$

$$\text{non per ogni } i{,}j.\ \textsc{Esat}(\alpha_i \leftrightarrow \neg\beta_j) = \mathit{True} \tag{4.22}$$

Sulla base della proprietà 4.1 possiamo applicare l'ipotesi induttiva e derivare:

$$\text{non per ogni } i{,}j.\ \not\vdash_E \alpha_i \leftrightarrow \beta_j \tag{4.23}$$

Possiamo allora sfruttare il lemma 3.1 per ottenere:

$$\vdash_E \neg(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma) \tag{4.24}$$

e cioè, essendo tale conclusione indipendente dal particolare assegnamento considerato, risulta $\vdash_E \neg\mu$ per ogni $\mu \in \mathcal{A}$ e quindi $\vdash_E \bigwedge_i \neg\mu_i$. Dal fatto che:

$$\vdash_E \neg\varphi \leftrightarrow \bigwedge_i \neg\mu_i \tag{4.25}$$

otteniamo $\vdash_E \neg\varphi$ e quindi l'assurdo.

4.5 La costruzione di decisori efficienti

Le procedure ΛSAT$_A$ descritte nella sezione 4.3, hanno il vantaggio di rispecchiare in maniera evidente la struttura dei corrispondenti lemmi esposti nel capitolo 3. D'altro canto, come avevamo già anticipato, molte di esse presentano ampi margini di miglioramento, almeno per quanto riguarda l'efficienza del ragionamento modale. Lo scopo di questa sezione è proprio quello di introdurre le versioni efficienti dei decisori sinora presentati e di discutere l'applicabilità di opportuni costrutti di controllo e memorizzazione dell'inferenza modale. Quest'ultima tecnica, nota in letteratura con il nome di *memoizing*, ci consente di ottenere un sostanziale miglioramento nelle prestazioni dei decisori, come abbiamo modo di verificare con i risultati sperimentali presentati nella sezione 5.3.

Nelle sezioni che seguono affrontiamo problemi connessi sia con la complessità delle singole logiche sia con il modo con cui gestiamo la combinazione delle procedure di decisione.

Sappiamo che per alcune logiche Λ, il problema della soddisfacibilità è risolubile in tempo polinomiale con un algoritmo non deterministico. In questi casi, la corrispondente procedura di decisione deve avere, al più, un numero *polinomiale* di chiamate ricorsive al decisore proposizionale. Le procedure che presentiamo in 4.5.1 relative alle logiche che hanno il test di soddisfacibilità NP-complete godono di questa proprietà e rispettano i vincoli teorici relativi alla complessità in spazio.

Le logiche per cui l'inferenza modale ha complessità maggiore, quali ad esempio EC, comportano sicuramente un maggiore sforzo realizzativo. In particolare, può essere utile cercare di definire opportune sequenze di controllo come abbiamo fatto nella sezione 4.3. Un esempio di tale strategia è il fatto che in ECSAT$_A$ si prova prima la soddisfacibilità delle formule $\neg\alpha_i \wedge \beta_j$ e, solo per quelle sono ancora nell'insieme Δ, si tenta di soddisfare $\bigwedge_{i \in \Delta} \alpha_i \wedge \neg\beta_j$. Anche in questo caso però, senza un adeguato supporto con tecniche di memoizing il primo test può generare un numero esponenziale di chiamate ricorsive, mentre il secondo test, che di per sè è tale da alzare la complessità della procedura di decisione, potrebbe essere reso più efficiente memorizzando opportunamente alcuni risultati intermedi. Questo, come mostriamo nella sezione 4.5.2 comporterà un compromesso con i vincoli teorici relativi alla complessità in spazio.

In entrambe le sezioni che seguono affrontiamo il problema della combinazione delle procedure di decisione e delle strategie usate per renderle efficienti. Tali problemi sorgono nel momento in cui decidiamo di gestire logiche come EMN oppure EMC riutilizzando le procedure per EM, EC o EN. Il tentativo che facciamo è quello di minimizzare, se possibile, i passi di inferenza memorizzati per il test di soddisfacibilità condividendoli fra i diversi test specifici per ogni logica.

4.5.1 Le procedure di decisione per E, EM, EN EMN

Anzitutto, vogliamo spiegare perché i decisori che abbiamo presentate per queste logiche non sono efficienti dal punto di vista computazionale. Consideriamo come esempio la procedura per E sia per la sua semplicità sia perché la sua struttura è adatta a rappresentare tutte le altre procedure descritte in questa parte. Abbiamo visto (teorema 3.1 e relativa discussione) che nel realizzare una procedura di decisione per E siamo di fronte a due vincoli teorici estremamente importanti:

- Il test di soddisfacibilità in una struttura $\mathcal{E}$ per una qualsiasi formula φ deve avere, al più, la stessa complessità di SAT a meno di un fattore moltiplicativo polinomiale nelle dimensioni di φ
- lo spazio allocato da una qualisasi procedura di decisione deve essere, al più, polinomiale nelle dimensioni di φ

In letteratura, per "dimensioni di φ" indicate con $\mid \varphi \mid$, si intende il *numero totale di simboli* che appaiono in φ (si veda [Fagin *et al.*, 1995]). In base a questa definizione, si ha, ad esempio, $\mid P_0 \wedge \Box P_0 \mid = 6$. Chiaramente, la catena di diseguaglianze $\mid \varphi \mid \geq \mid sub(\varphi) \mid \geq \mid sub_{\Box}(\varphi) \mid$ è valida per ogni φ.

Studiamo allora, in maniera informale, la complessità di ESAT così come risulterebbe utilizzando la procedura ESAT_A presentata nella sezione 4.3.1 in figura 4.3. In particolare consideriamo la formula $\varphi = (\Box(P_0 \wedge \top) \rightarrow \Box P_0) \rightarrow \Box(P_0 \wedge P_0)$. Notiamo che, data la particolare struttura degli atomi modali, la procedura ESAT_{DP} "vede" la formula come se si trattasse di $(A \rightarrow B) \rightarrow C$ e quindi può assegnare valori di verità diversi a ciascun atomo modale. Supponiamo che il primo assegnamento passato a ESAT_A sia $\mu_1 = \Box(P_0 \wedge P_0) \wedge \neg\Box(P_0 \wedge \top) \wedge \neg\Box P_0$. Il primo test ricorsivo, e cioè $\text{ESAT}((P_0 \wedge P_0) \leftrightarrow \neg(P_0 \wedge \top)$, fallisce (ovviamente) per cui il controllo ritorna a ESAT che deve tentare un nuovo assegnamento. Supponiamo che questo sia nella forma $\mu_2 = \Box(P_0 \wedge P_0) \wedge \Box P_0 \wedge \neg\Box(P_0 \wedge \top)$. Il primo test ricorsivo, nella forma $\text{ESAT}((P_0 \wedge P_0) \leftrightarrow \neg(P_0 \wedge \top)$, è un'esatta copia di quello già eseguito. Questo comportamento constrasta con il fatto che, al massimo, ci devono essere tanti test ricorsivi quante sono le possibili coppie di sottoformule modali di φ.

La soluzione che presentiamo è la conseguenza immediata del seguente ragionamento. Se nell'esempio precedente avessimo memorizzato il risultato del passo di inferenza con cui avevamo stabilito che $\vdash_E (P_0 \wedge P_0) \leftrightarrow (P_0 \wedge \top)$, non avremmo dovuto ripetere lo stesso test e avremmo potuto dichiarare immediatamente che l'assegnamento μ_2 è inconsistente. Allo scopo di realizzare questo meccanismo, supponiamo di avere a disposizione le seguenti primitive:

- Λ-*store*(φ, ψ, r) consente di memorizzare il risultato r di un test riguardante le formule φ e ψ nella particolare logica Λ;

- Λ-*get*(φ, ψ) restituisce *True* o *False* se un risultato riguardante φ e ψ è stato memorizzato e $\perp$ nel caso contrario (nessuna inferenza riguardante φ e ψ è stata memorizzata);

- Λ-*memoizing-decision*(φ, ψ, r) realizza una qualche forma di decisione euristica dipendente dalla particolare logica Λ e pertanto ritorna T o F a seconda che sia il caso o meno di memorizzare il risultato r per il test di φ e ψ.

La struttura dati che meglio si presta al supporto di queste primitive è la semplice tabella presentata in figura 4.13. Si noti che abbiamo usato T e F al posto di *True* e *False* per questioni di spazio. La tabella è costruita considerando una generica formula φ e $\{\psi_1 \ldots \psi_r\} \equiv sub_\Box(\varphi)$ cioè l'insieme di tutti gli atomi modali contenuti in φ. Nel seguito ci riferiamo a tale insieme anche con $\Gamma_\Box$. Come abbiamo visto nelle discussioni associate ai teoremi 3.1, 3.2, 3.4, relativi alle logiche presentate in questa sezione, tutte le logiche con test di soddisfacibilità NP-complete confrontano a coppie le sottoformule modali di un assegnamento. La matrice in figura 4.13 esprime proprio l'essenza di questo confronto e cioè il risultato relativo al test ricorsivo per la coppia (ψ_l, ψ_k) di generiche formule $\psi_l, \psi_k \in \Gamma_\Box$ è contenuto nella cella puntata dalla coppia di coordinate (riga, colonna).

Notiamo che, dal punto di vista teorico, la matrice risulta la struttura dati che più fedelmente riproduce le caratteristiche del ragionamento nelle logiche considerate. Infatti, lo spazio occupato dalla matrice cresce come $\mid sub_\Box(\varphi) \mid^2$ e quindi lo spazio di memoria occupato dai decisori con l'aggiunta di memoizing è al più quadratico nelle dimensioni di φ. L'efficienza in termini di spazio del memoizing è chiaramente collegata al fatto che φ è soddisfacibile in una delle logiche considerate se e solo se è soddisfacibile in una corrispondente struttura $\mathcal{MS}$ con al più un numero di mondi pari a $\mid \varphi \mid^2$. Anche in termini di tempo, siamo garantiti che una procedura che sfrutta il memoizing compie, al più, $\mid sub_\Box(\varphi) \mid^2$ test di soddisfacibilità e cioè quelli necessari per riempire completamente la matrice.

La struttura e le primitive di memoizing, così come le esponiamo, astraggono fortemente dai dettagli realizzativi. Nel seguito non entreremo nel merito di questi ultimi in quanto non sono necessari per la comprensione degli argomenti presentati. Si rimanda comunque all'appendice A e in particolare alla sezione A.6 per una esposizione dettagliata della realizzazione del memoizing.

Date le particolari forme di inferenza realizzate dalle diverse logiche oggetto di questa sezione, le primitive e la struttura dati per il memoizing possono avere un diverso comportamento o un diverso contenuto a seconda della logica considerata. Pertanto, oltre che per completezza di esposizione, analizzeremo le procedure $\Lambda\textsc{sat}_A$ con aggiunta del memoizing per ogni logica presa in esame. Per ottenere i decisori completi è sufficiente combina-

$$\begin{array}{cccccccc}
 & \psi_1 & \cdots & \psi_k & \cdots & \psi_l & \cdots & \psi_r \\
\psi_1 & - & \cdots & \perp & \cdots & T & \cdots & F \\
\vdots & & & & & & & \\
\psi_k & \perp & \cdots & - & \cdots & F & \cdots & \perp \\
\vdots & & & & & & & \\
\psi_l & T & \cdots & F & \cdots & - & \cdots & T \\
\vdots & & & & & & & \\
\psi_r & F & \cdots & \perp & \cdots & T & \cdots & -
\end{array}$$

Figura 4.13: Struttura del memoizing per le logiche NP

re le procedure presentate in questa parte con il framework descritto nella sezione 4.2.

Iniziamo dalla procedura di decisione per la logica E. In figura 4.14 è presentato in dettaglio il frammento di pseudo-codice relativo al controllo di consistenza modale dell'assegnamento.

Figura 4.14: Procedura $\text{E}\textsc{sat}_A$ con memoizing

---

function $\text{E}\textsc{sat}_A(\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      $r := E\text{-}get(\alpha_i, \beta_j)$;
      if $r = \perp$
        then $r :=$ **not** $\text{E}\textsc{sat}(\alpha_i \leftrightarrow \neg\beta_j)$;
          if $E\text{-}memoizing\text{-}decision(\alpha_i, \beta_j, r)$
            then $E\text{-}store(\alpha_i, \beta_j, r)$;
      if r **then return** *False*
  return *True*.

Dando temporaneamente per scontato il funzionamento delle primitive di memoizing, descriviamo brevemente la procedura $\text{E}\textsc{sat}_A$ così come risulta dopo l'aggiunta di tali costrutti. All'interno dei loop annidati che scandiscono gli α e i β, prima di invocare il test ricorsivo, la procedura controlla se è stato memorizzato un qualche risultato relativo alla validità di $\alpha \leftrightarrow \beta$. Se così è, e se il risultato è *True*, allora può concludere che l'assegnamento è inconsistente e ritornare *False*. Viceversa il loop prosegue. Nel caso in cui non venga reperito alcun risultato che consenta di evitare il test ricorsivo,

$\textsc{Esat}_A$ lo esegue e ne memorizza il risultato. Se i test hanno sempre successo, la procedura ritorna *True*. Notiamo che nella nostra attuale realizzazione (vedi appendice A) l'euristica corrispondente a *E-memoizing-decision* è tale da ritornare sempre *True*.

Per descrivere struttura e controllo del memoizing, facciamo un passo indietro e consideriamo la formula φ di cui stiamo controllando la soddisfacibilità. L'insieme $\Gamma_\Box \equiv sub_\Box(\varphi)$ determina la forma della matrice presentata in figura 4.13 che ha dimensioni $\mid \Gamma_\Box \mid \times \mid \Gamma_\Box \mid$. Dato che $\alpha_i \in \Gamma_\Box$ e $\beta_j \in \Gamma_\Box$ per ogni i e j e per qualsiasi assegnamento μ costruito dal decisore proposizionale $\textsc{Esat}_{DP}$, la matrice di memoizing ha dimensioni sufficienti per gestire il risultato di qualsiasi fra i possibili test ricorsivi tra coppie di formule in $\Gamma_\Box$. Riferendoci alla struttura di memoizing la chiamiamo Ω e diciamo che è una variabile *globale* nel senso che è comune a tutti i rami ricorsivi della procedura di decisione.

Osservando Ω notiamo che gli elementi sulla diagonale non sono utilizzati in quanto è impossibile che il decisore proposizionale costruisca un assegnamento inconsistente (si veda la sezione 4.4) e quindi tale che $\mu(\psi) = 1$ e $\mu(\psi) = 0$. Inoltre la matrice è simmetrica in quanto viene utilizzata per memorizzare la *validità* della formula $\alpha \leftrightarrow \beta$ che è comunque valida se si scambiano α e β. Questo spiega anche perché la procedura $\textsc{Esat}_A$, quando $r = True$ ritorna *False*. La scelta di memorizzare T in corrispondenza di $\vdash_E \alpha \leftrightarrow \beta$ è stata fatta per evitare casi in cui uno stesso test possa essere ripetuto due volte. In altre parole, la struttura di memoizing si fa anche carico della semplice inferenza $\alpha \leftrightarrow \beta \Leftrightarrow \beta \leftrightarrow \alpha$ che è corretta in $\mathcal{E}$ per qualsiasi α e β.

Veniamo alla descrizione delle primitive di memoizing. La procedura *E-store* deve memorizzare il risultato r sia alla locazione (α, β), sia alla locazione (β, α) in modo da garantire la simmetria della matrice. L'accesso alla struttura avviene tramite la funzione *E-get* che si limita a restituire il valore memorizzato nella matrice alla locazione (α, β) richiesta. Ovviamente, il risultato restituito da *E-get* deve essere inteso in maniera consistente con il funzionamento di *E-store*. Infine, supponendo che ogni locazione di Ω sia inizializzata a $\perp$, se non è avvenuto uno *E-store* per una particolare coppia di formule *E-get* restituisce $\perp$. Entrambe le procedure sono presentate in figura 4.15.

La procedure di decisione presentata in figura 4.16 per *EM* non presenta grosse differenze rispetto a quella per E (a parte, ovviamente, il diverso test ricorsivo) e pertanto non ci soffermiamo sulla descrizione del suo funzionamento.

Piuttosto, siamo interessati a mostrare la diversa realizzazione del memoizing. Cominciamo dalla matrice Ω_M presentata in figura 4.17. Gli elementi sulla diagonale risultano ancora inutilizzati per lo stesso motivo visto per E: non è possibile che $\textsc{EMsat}_{DP}$ costruisca un assegnamento inconsistente, cioè tale che $\mu(\psi) = 1$ e $\mu(\psi) = 0$. La matrice Ω_M non è però

Figura 4.15: Primitive per il memoizing in E

---

function $E\text{-}get(\psi_1, \psi_2)$
  return $\Omega(\psi_1, \psi_2)$.

procedure $E\text{-}store(\psi_1, \psi_2, r)$
  $\Omega(\psi_1, \psi_2) := r$;
  $\Omega(\psi_2, \psi_1) := r$.

Figura 4.16: Procedura EMSAT$_A$ con memoizing

---

function EMSAT$_A(\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      $r := EM\text{-}get(\alpha_i, \beta_j)$;
      if $r = \bot$
        then $r :=$ **not** EMSAT$(\alpha_i \wedge \neg\beta_j)$;
            if $EM\text{-}memoizing\text{-}decision(\alpha_i, \beta_j, r)$
              then $EM\text{-}store(\alpha_i, \beta_j, r)$;
      if r **then return** *False*
  return *True*.

---

simmetrica. Questo è ovvio, se si pensa al fatto che il test ricorsivo coinvolge la validità di un'implicazione tra un generico α e un generico β. Infatti risulta $\not\vdash_{EM} \alpha \rightarrow \beta$ se e solo se $\not\vdash_E \neg(\alpha \wedge \neg\beta)$ e cioè se e solo se $\alpha \wedge \neg\beta$ è soddisfacibile in qualche struttura $\mathcal{E}_M$. Anche in questo caso utilizziamo la matrice per memorizzare la validità del test ricorsivo e quindi, sapere che $\vdash_E \alpha \rightarrow \beta$ non dice nulla circa $\vdash_E \beta \rightarrow \alpha$ e viceversa.

La realizzazione delle primitive di memoizing è abbastanza semplice. In particolare, la prima delle due possibili primitive $EM\text{-}store$ proposte in figura 4.18, si limita a memorizzare il valore r del test ricorsivo nelle celle α e β passate come parametro.

Se volessimo, come nel caso di E, affidare a $EM\text{-}store$ una qualche funzione deduttiva, potremmo sfruttare il fatto che se $\vdash_{EM} \psi_1 \rightarrow \psi_2$ e $\vdash_{EM} \psi_2 \rightarrow \psi_3$ allora anche $\vdash_{EM} \psi_1 \rightarrow \psi_3$. Sfruttare questa possibilità è un'operazione intrinsecamente più costosa rispetto a quella di memorizzare lo stesso valore in due celle diverse. Infatti è necessario, ogni volta che facciamo accesso a Ω_M per la memorizzazione di un risultato, controllare

$$
\begin{array}{cccccccc}
 & \psi_1 & \cdots & \psi_k & \cdots & \psi_l & \cdots & \psi_r \\
\psi_1 & - & \cdots & \perp & \cdots & T & \cdots & F \\
\vdots & & & & & & & \\
\psi_k & \perp & \cdots & - & \cdots & F & \cdots & \perp \\
\vdots & & & & & & & \\
\psi_l & T & \cdots & T & \cdots & - & \cdots & \perp \\
\vdots & & & & & & & \\
\psi_r & \perp & \cdots & \perp & \cdots & T & \cdots & -
\end{array}
$$

Figura 4.17: Matrice Ω_M per la logica EM

Figura 4.18: Primitive di accesso a Ω_M

procedure *EM-store*(ψ_1, ψ_2, r)
  $\Omega_M(\psi_1, \psi_2) := r$.

procedure *EM-store*$^*(\psi_1, \psi_2, r)$
  $\Omega(\psi_1, \psi_2) := r$;
  if $r = True$
    then foreach ξ **in** $\Gamma_\Box$ **do**
      if $\Omega_M(\psi_2, \xi) = True$ **then** $\Omega(\psi_1, \xi) := True$.

che tale risultato sia $r = True$, cioè i parametri attuali ψ_1 e ψ_2 siano tali che $\vdash_{EM} \psi_1 \to \psi_2$. Se così è allora bisogna cercare tutte le formule ψ_l tali che $\vdash_{EM} \psi_l \to \psi_1$ per poter poi concludere che $\vdash_{EM} \psi_l \to \psi_2$. Oppure bisogna cercare tutte le ψ_k tali che $\vdash_{EM} \psi_2 \to \psi_k$ in modo da poter concludere che $\vdash_{EM} \psi_1 \to \psi_k$. La seconda tra queste due ricerche è proposta in figura 4.18 con la procedura *EM-store**. Altri metodi, quali ad esempio, il calcolo della chiusura transitiva di tutte le coppie (ψ_1, ψ_2) per cui è noto che $\vdash_{EM} \psi_1 \to \psi_2$, non vengono neanche presi in esame in quanto tali problemi sono almeno NP-hard (per i dettagli, si veda [Garey e Johnson, 1979]).

Soffermiamoci brevemente sull'opportunità di introdurre una procedura più "intelligente" di *EM-store*, in grado cioè di compiere alcune deduzioni. Anzitutto, notiamo che il costo computazionale di *EM-store* è costante mentre la stessa cosa non può dirsi di *EM-store** il cui costo è, al più, proporzionale a $\mid sub(\varphi) \mid$. Questo significa che, ogni volta che si rende necessario un test ricorsivo in EMSAT$_A$ e il risultato di quest'ultimo è *True*, dobbiamo spendere un tempo lineare in $\mid \varphi \mid$ per effettuare la memorizzazione. Ovviamente qualsiasi inferenza aggiuntiva che riusciamo a fare in questa fase può risparmiarci un test ricorsivo, ma d'altronde non è detto che le dimensioni di φ siano tali da rendere conveniente questo approccio.

La realizzazione di *EM-get* è semplice quanto l'analoga *E-get*: è sufficiente una procedura che date due formule restituisca il valore contenuto nella cella avente come coordinate le formule stesse. Per quanto riguarda *EM-memoizing-decision*, la nostra realizzazione dei decisori prevede che tale funzione ritorni sempre *True*.

Continuiamo la nostra discussione circa i decisori analizzando la procedura di decisione per EN. Nella sezione 4.3 avevamo detto che con l'introduzione del memoizing saremmo stati in grado di combinare in maniera efficiente le la procedura di decisione per E (nella versione di figura 4.4) con il test di consistenza relativo all'assioma N. Il risultato di tale combinazione è mostrato nel frammento di pseudo-codice in figura 4.19.

Figura 4.19: Procedura ENSAT_A con memoizing

---

function $\text{ENSAT}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\alpha_i$ **do**
    foreach conjunct $\Box\beta_j$ **do**
      $r := \textit{EN-get}(\alpha_i, \beta_j)$;
      if $r = \bot$
        then $r :=$ **not** $\text{ENSAT}(\alpha_i \wedge \neg\beta_j)$;
            if $\textit{EN-memoizing-decision}(\alpha_i, \beta_j, r)$
              then $\textit{EN-store}(\alpha_i, \beta_j, r)$;
      if r **then** $r := \textit{EN-get}(\beta_j, \alpha_i)$;
            if $r = \bot$
              then $r :=$ **not** $\text{ENSAT}(\neg\alpha_i \wedge \neg\beta_j)$;
                if $\textit{EN-memoizing-decision}(\beta_j, \alpha_i, r)$
                  then $\textit{EN-store}(\beta_j, \alpha_i, r)$;
      if r **then return** *False*;
  foreach conjunct $\Box\beta_j$ **do**
    $r := \textit{N-get}(\beta_j)$;
    if $r = \bot$
      then $r :=$ **not** $\text{ENSAT}(\neg\beta_j)$;
          if $\textit{N-memoizing-decision}(\beta_j, r)$
            then $\textit{N-store}(\beta_j, r)$;
    if r **then return** *False*;
  return *True*.

$$
\begin{array}{cccccccc}
 & \psi_1 & \cdots & \psi_k & \cdots & \psi_l & \cdots & \psi_r \\
\psi_1 & F & \cdots & \perp & \cdots & \perp & \cdots & F \\
\vdots & & & & & & & \\
\psi_k & \perp & \cdots & T & \cdots & F & \cdots & \perp \\
\vdots & & & & & & & \\
\psi_l & T & \cdots & \perp & \cdots & F & \cdots & \perp \\
\vdots & & & & & & & \\
\psi_r & F & \cdots & \perp & \cdots & T & \cdots & F
\end{array}
$$

Figura 4.20: Matrice Ω_N per il memoizing nella logica EN

Spieghiamo il funzionamento di ENsat$_A$. La procedura consta di due parti che corrispondono, rispettivamente, al test di consistenza per E e al test relativo all'assioma N. Nella prima parte ritroviamo i due loop che scandiscono tutte le coppie α e β controllando che per nessuna di esse α e β siano logicamente equivalenti. Ricordiamo dalla sezione 4.3.1 che questo può essere fatto realizzando una disgiunzione lazy tra i due test ENSAT($\alpha \wedge \neg\beta$) e ENSAT($\neg\alpha \wedge \beta$). Prima di eseguire i due test ricorsivi è necessario controllare se vi sia un risultato precedentemente memorizzato e, in caso contrario, eseguire il test e memorizzare il risultato. Notiamo che le funzioni di *EN-store* sono entrambe "non simmetriche" nel senso che il risultato relativo ai due test ricorsivi viene memorizzato indipendentemente. La procedura si conclude con il loop che scandisce tutti i β e verifica che nessuno di essi sia valido in $\mathcal{E}_N$ con il test ricorsivo ENSAT($\neg\beta$). Anche in questo caso, prima di eseguire il test si controlla che non vi sia un'inferenza precedente memorizzata che consenta di evitare la chiamata a ENSAT.

Il funzionamento di ENSAT è abbastanza articolato e, prima di descriverlo a fondo, è necessario spiegare il funzionamento delle primitive di memoizing e della composizione della matrice Ω_N. Quest'ultima è presentata in figura 4.20, mentre le primitive *EN-get* e *EN-store*^{r} sono date in figura 4.21.

In generale Ω_N è una matrice non simmetrica, in quanto pensiamo alle coordinate (ψ_1, ψ_2) come alla "traduzione" della formula $\psi_1 \to \psi_k$ e quindi sapere che $\alpha \wedge \neg\beta$ è (in)soddisfacibile non dice nulla circa la (in)soddisfacibilità di $\neg\alpha \wedge \neg\beta$. Anche per EN seguiamo la convenzione che il valore booleano eventualmente memorizzato in una cella (ψ_1, ψ_2) concerne la validità della formula $\psi_1 \to \psi_2$ Per cui il fatto che $\Omega_N(\psi_K, \psi_l) = \textit{False}$ significa che $\psi_k \to \psi_l$ è stata dimostrata non valida e quindi $\psi_k \wedge \neg\psi_l$ è soddisfacibile. Le celle sulla diagonale sono utilizzate per contenere i valori relativi alla validità delle formule, cioè memorizzano i risultati del test per l'assioma N. Vale sempre la convenzione relativa alla lettura del valore per cui, ad esempio, $\Omega_N(\psi_l, \psi_l) = \textit{False}$ significa che ψ_l è stata dimostrata non valida in EN

Figura 4.21: Primitive per il memoizing in EN

function $EN\text{-}get(\psi_1, \psi_2)$
  return $\Omega(\psi_1, \psi_2)$.

procedure $EN\text{-}store(\psi_1, \psi_2, r)$
  $\Omega(\psi_1, \psi_2) := r$;
  if $r = False$ **then** $\Omega(psi_2, \psi_2) := r$.

($\neg\psi_l$ è soddisfacibile).

Per quanto riguarda le primitive di memoizing, EN-*get* opera nel modo consueto, restituendo il valore contenuto nella cella puntata dalla coppia (ψ_1, ψ_2), mentre EN-*store* presenta una funzionalità estesa rispetto alla semplice memorizzazione del risultato per la coppia di formule passate come parametro. Questo perché avendo scisso il test di consistenza per E, se sappiamo che $\alpha \wedge \neg\beta$ è soddisfacibile, allora sappiamo anche che β non è valida. Viceversa se sappiamo che $\neg\alpha \wedge \beta$ è soddisfacibile, allora sappiamo anche che α non è valida. Per non sprecare questa informazione ed essere costretti a ripetere inutilmente i test di validità sui soli β (anche per altri assegnamenti), EN-*store* è stata dotata di questa capacità di inferenza. Questa modifica, che comporta un minimo costo computazionale, ci consente di combinare in maniera efficiente il test ricorsivo per la logica E con il test di consistenza per l'assioma N.

Figura 4.22: Primitive per il memoizing in EN

function $N\text{-}get(\psi_1)$
  return $\Omega(\psi_1, \psi_1)$.

procedure $N\text{-}store(\psi_1, r)$
  $\Omega(\psi_1, \psi_1) := r$.

procedure $N\text{-}store^*(\psi_1, r)$
  $\Omega(\psi_1, \psi_1) := r$;
  if $r = True$ **then foreach** ξ **in** $\Gamma_\Box$ **do**
    $\Omega(\xi, \psi_1) := r$.

In figura 4.22 presentiamo le primitive che sono utilizzate come supporto al test ENsat($\neg\beta$). Le semplici *N-get* e *N-store* si limitano a nascondere il dettaglio realizzativo per cui i valori relativi al test sono memorizzati lungo la diagonale, altrimenti inutilizzata, della matrice Ω_N. Invece *N-store** è una possibile versione "intelligente" di *N-store* basata sul fatto che la validità di un certo ψ rende automaticamente valide tutte le formule del tipo $\xi \to \psi$ con $\xi \in \Gamma_\Box$. Dato che il costo di una tale inferenza è proporzionale a $\mid sub_\Box(\varphi) \mid$, non è detto a priori che risulti sempre la scelta più opportuna.

Concludiamo questa sezione con la procedura per la logica EMN. Così come accadeva per EN, all'interno della procedura di decisione presentata in figura 4.23 sono distinguibili la parte dedicata al controllo di consistenza per la logica EM a cui si aggiunge l'inferenza per relativa all'assioma N. A parte l'introduzione del memoizing, non ci sono sostanziali differenze rispetto alla procedura presentata nella sezione 4.3.4, per cui non ci soffermiamo sulla descrizione del funzionamento di EMNsat.

Figura 4.23: Procedura EMNsat$_A$ con memoizing

---

function EMNsat$_A$($\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma$)
  if {there is any conjunct $\Box\alpha_i$}
    then foreach conjunct $\Box\alpha_i$ **do**
        foreach conjunct $\Box\beta_j$ **do**
          $r :=$ *EMN-get*(α_i, β_j);
          if $r = \bot$
            then $r :=$ **not** EMNsat($\alpha_i \wedge \neg\beta_j$);
                if *EMN-memoizing-decision*(α_i, β_j, r)
                  then *EMN-store*(α_i, β_j, r);
          if r **then return** *False*;
      return *True*
    else foreach conjunct $\Box\beta_j$ **do**
        $r :=$ *N-get*(β_j);
        if $r = \bot$
          then $r :=$ **not** EMNsat($\neg\beta_j$);
              if *N-memoizing-decision*(β_j, r)
                then *N-store*(β_j, r);
        if r **then return** *False*;
  return *True*.

Per quanto riguarda il memoizing, è ancora possibile realizzare la struttura dati con una matrice e, in particolare, è possibile organizzarla come abbiamo fatto per EN. Per tutte le coppie di coordinate (ψ_1, ψ_2) memorizziamo il valore booleano relativo alla validità della formula $\psi_1 \rightarrow \psi_2$ e utilizziamo le celle sulla diagonale per memorizzare il risultato di EMNSAT($\neg\beta$). L'implementazione di *EMN-store* può tenere conto in maniera efficiente del fatto che se $\alpha \wedge \neg\beta$ è soddisfacibile allora risulta tale anche $\neg\beta$. A tal fine è sufficiente aggiungere un test sul risultato che viene memorizzato e decidere se è il caso di aggiornare il valore corrispondente alla validità di β.

4.5.2 Le procedure di decisione per EC, ECN, R

In questa sezione abbiamo raggruppato tutte le logiche (eccetto K) i cui modelli sono inclusi nella classe $\mathcal{E}_C$ per due importanti ragioni. Il fatto che ECN, R ed EC abbiano un problema di decisione che è PSPACE-hard rende i decisori per tali logiche e le tecniche che adottiamo per renderli più efficienti fondamentalmente diverse dalle procedure esaminate nella sezione 4.5.1. Inoltre, a differenza di quanto accadeva per le logiche il cui problema di decisione è "solo" NP-complete, l'applicazione delle tecniche di memoizing ci costringe a violare il vincolo teorico per cui i decisori per EC, ECN, ed R dovrebbero avere una quantità di memoria allocata che cresce, al peggio, *polinomialmente* nelle dimensioni della formula da decidere. Come vediamo nella discussione che segue, non siamo in grado di garantire che questo si verifichi per *qualsiasi* formula in ingresso. Nonostante questo, dato che siamo comunque interessati a capire come le tecniche applicate con successo nella sezione 4.5.1 si comportano nel caso di decisori più complessi, lasciamo ai casi di studio sperimentali della sezione 5.3.2 l'ultima parola sull'efficienza *reale* dei decisori che introduciamo in questa sezione.

Come abbiamo già detto nella discussione circa il teorema 3.3, la ragione per cui il problema decisionale da noi affrontato aumentava di complessità è dovuta alla particolare natura delle chiamate ricorsive necessarie per controllare la consistenza di un assegnamento. Infatti, e questo è comune a tutte le logiche di questa sezione, la chiamata ricorsiva a ΛSAT è del tipo ΛSAT($\bigwedge_{i\in\Delta} \alpha_i \wedge \neg\beta_j$) come si può vedere dale figure 4.9, 4.10 e 4.11 nella sezione 4.3. Il fatto che il test ricorsivo coinvolga un *insieme* di formule α che viene provato per ciascuna formula β, implica una modifica nella definizione delle primitive di memoizing che avevamo dato nella sezione 4.5.1. Introduciamo allora le seguenti primitive astratte che gestiscono insiemi:

- Λ-*store*$^s(\Delta, \psi, r)$ consente di memorizzare il risultato r di un test riguardante l'insieme di formule Δ e la formula ψ nella particolare logica Λ;

- Λ-*get*$^s(\Delta, \psi)$ restituisce *True* o *False* se un risultato riguardante Δ e ψ è stato memorizzato e $\perp$ nel caso contrario (nessuna inferenza riguardante Δ e ψ è stata memorizzata);

- Λ-*memoizing-decision*$^s(\Delta, \psi, r)$ realizza una qualche forma di decisione euristica dipendente dalla particolare logica Λ e pertanto ritorna T

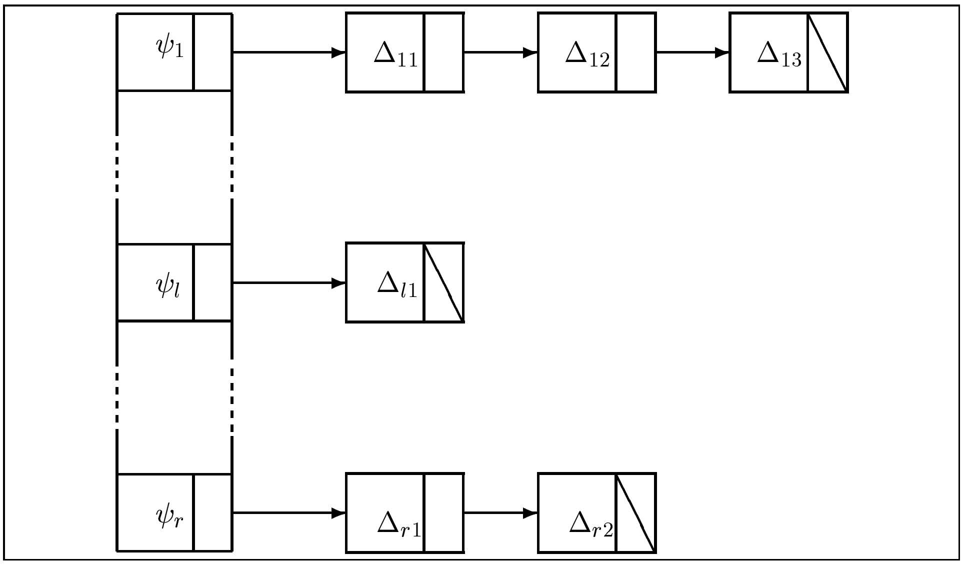

Figura 4.24: Struttura dati per il memoizing in logiche PSPACE

o F a seconda che sia il caso o meno di memorizzare il risultato r per il test di Δ e ψ.

La struttura dati più adatta per il supporto a queste primitive è presentata in figura 4.24. Notiamo che si tratta di una struttura dinamica in quanto, a differenza di quanto accade per le procedure presentate nella sezione 4.5.1, non è assolutamente conveniente allocare in anticipo lo spazio necessario a contenere il risultato di tutti i test. Un semplice calcolo dimostra che i possibili test nella forma $\Lambda\textsc{sat}(\bigwedge_{i\in\Delta}\alpha_i \wedge \neg\beta)$ sono un numero esponenziale nelle dimensioni della formula in ingresso. Consideriamo infatti una generica formula φ e, come al solito, definiamo $\Gamma_\Box \equiv sub_\Box(\varphi)$. Risulta che tutti i possibili $\Delta \subseteq \Gamma_\Box$ sono dati da $2^{|\Gamma_\Box|}$ e quindi tutti i possibili test che coinvolgano un certo insieme Δ e una formula $\psi \in \Gamma_\Box$ sono un numero pari a:

$$|\Gamma_\Box|\, 2^{|\Gamma_\Box|} \leq |\varphi|\, 2^{|\varphi|} \tag{4.26}$$

La scelta di una struttura dinamica non ci consente di affermare che, nel caso pessimo, lo spazio allocato per il memoizing non diventi proibitivo, ma siamo senz'altro garantiti che usiamo solo quello necessario.

L'organizzazione della struttura per il memoizing di figura 4.24 è articolata in due parti. Una tabella allocata staticamente (sulla sinistra della figura) che associa ogni formula dell'insieme $\Gamma_\Box = \{\psi_1 \ldots \psi_r\}$ ad un puntatore ad una lista contenente tutti gli insiemi di formule Δ che sono stati provati con la particolare ψ. Ad esempio, sapere che nella lista associata a ψ_l c'è l'insieme Δ_{l1} significa che è il risultato del test ricorsivo $\Lambda\textsc{sat}(\bigwedge_{\alpha_i\in\Delta_{l1}}\alpha_i \wedge \neg\psi_l)$ è già stato computato. Data la particolare organizzazione della struttura, gli insiemi Δ appartenenti a diverse liste sono completamente isolati, nel senso che è possibile che all'interno delle liste vi siano più copie dello stesso insieme. Questo può comportare uno "spreco" di spazio, ma d'altronde solleva i decisori da una complessa gestione dei puntatori che potrebbe vanificare il vantaggio dato dall'utilizzo del memoizing. Facciamo inotre notare, per le ragioni che chiariamo tra poco, che all'interno di ogni cella allocata dinamicamente troviamo solo l'insieme di formule corrispondente ad un certo test ricorsivo, ma non il risultato del test stesso.

Facciamo un passo indietro e torniamo a considerare il particolare test ricorsivo realizzato dalle procedure ECsat, ECNsat e EMCsat. In particolare ci chiediamo quale informazione possiamo ricavare dal fatto che la formula $\bigwedge_{i \in \Delta} \alpha_i \wedge \neg\beta$ sia (in)soddisfacibile in una fra le logiche oggetto di questa sezione. Consideriamo i seguenti casi:

- la formula $\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta$ è soddisfacibile per un particolare Δ e un particolare β. Da questo possiamo dedurre che per qualsiasi $\Psi \subseteq \Delta$ la formula $\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta$ è soddisfacibile;
- viceversa, se la formula $\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta$ è insoddisfacibile per un particolare Δ e un particolare β possiamo dedurre che per qualsiasi $\Phi \supseteq \Delta$ la formula $\bigwedge_{\alpha_i \in \Phi} \alpha_i \wedge \neg\beta$ è insoddisfacibile;

Ovviamente, se un insieme di formule è vero in un mondo di un modello, tutti i possibili sottoinsiemi saranno comunque veri. Viceversa, se non è possibile trovare un mondo di un modello per cui tutte le formule di un certo insieme sono contemporaneamente verificate, allora non sarà possibile neanche per un sovrainsieme.

Incorporare la capacità di eseguire questi controlli all'interno delle strutture di memoizing ci è sembrato un buon mezzo per ridurre la loro occupazione (media) di spazio in memoria. Per capire la ragione di questo consideriamo l'ottimo nei casi, rispettivamente, della soddisfacibilità e dell'insoddisfacibilità. Nel primo caso, se si riuscisse a dimostrare per un certo $\beta \in \Gamma_\Box$ che $\bigwedge_{\alpha_i \in \Gamma_\Box} \alpha_i \wedge \neg\beta$, allora sarebbe sufficiente un solo elemento nella lista associata a β per riassumere tutte le possibili inferenze di questo tipo. Nel secondo caso, se si riuscisse a dimostrare che per un certo $\beta \in \Gamma_\Box$ esiste un $\alpha \in \Gamma\Box$ tale che $\alpha \wedge \neg\beta$ è insoddisfacibile, allora un'unica cella di memoria riassumerebbe tutte le possibili inferenze con insiemi contenenti quel particolare α.

Per realizzare la struttura e le primitive ad essa associate procediamo allora come segue. Il nostro primo obbiettivo è realizzare una procedura che ci permetta di controllare in maniera efficiente se tra due insiemi di formule esiste una relazione di sottoinsieme (sovrainsieme). Per la discussione che segue è necessario estendere lo pseudo-linguaggio utilizzato nel presentare le procedure con il tipo generico *lista ordinata*. Diciamo tipo generico in quanto è possibile costruire una lista ordinata a partire da qualsiasi tipo di elemento per cui sia data una nozione di *ordine*. Per tutte le liste ordinate valgono le seguenti primitive:

- *head* prende in ingresso una lista ordinata e restituisce il primo elemento della lista;
- *tail* prende in ingresso una lista ordinata e restituisce la lista ottenuta eliminando il primo elemento;
- *empty* ritorna *True* se la lista è vuota e *False* altrimenti.

Data una formula $\varphi \in \mathcal{L}$ vogliamo definire un ordine sulle sottoformule modali di φ. Per fare questo supponiamo di avere una funzione $label : sub(\varphi) \to \mathbf{N}$ tale che $label(\psi) = label(\xi)$ se e solo se $\psi = \xi$ e diciamo che ψ è minore di ξ, indicato con $\psi \prec_{\mathcal{L}} \xi$ se e solo se $label(\psi) < label(\xi)$. Per i dettagli su come sia possibile realizzare praticamente la funzione *label*, rimandiamo all'appendice A e in particolare alle sezioni A.3 e A.8.

Ora che abbiamo una nozione di ordine sulle formule possiamo costruire liste ordinate di formule (ennuple). In particolare tutti i sottoinsiemi di $\Gamma_\Box \equiv sub_\Box(\varphi)$ possono essere rappresentati come liste ordinate. Possiamo allora definire un ordinamento *lessicografico*, e lo indichiamo con "$\prec$", sulle liste ordinate che corrispondono a possibili sottoinsiemi di $\Gamma_\Box$ come segue:

- per qualsiasi lista ordinata Δ tale che $empty(\Delta)$ è falso, risulta $\emptyset \prec \Delta$;
- date due ennuple non vuote $\Psi, \Xi \subseteq \Gamma_\Box$ se $head(\Xi) \prec_{\mathcal{L}} head(\Psi)$, allora $\Xi \prec \Psi$. Viceversa, se $head(\Psi) \prec_{\mathcal{L}} head(\Xi)$ $\Psi \prec \Xi$. Infine, se $\psi = \xi$ allora $\Psi \prec \Xi$ se e solo se $tail(\Psi) \prec tail(\Xi)$.

Si noti che per indicare liste ordinate di formule usiamo le lettere greche maiuscole Γ, Δ, Ψ, ecc., con cui siamo soliti indicare anche gli insiemi di formule. Utilizzando l'ordinamento "$\prec$" possiamo costruire liste di liste (insiemi) di formule. Per indicare liste ordinate (lessicograficamente) di liste ordinate di formule utilizziamo le lettere greche maiscole in mezzo a parentesi quadre (per cui abbiamo, ad esempio, $[\Gamma]$, $[\Delta]$, $[\Psi]$, ecc.).

Date queste definizioni preliminari siamo in grado di costruire una procedura che, date due liste ordinate Ψ e Ξ, distingua i tre casi seguenti:

- esiste una relazione di inclusione tra gli insiemi corrispondenti nel senso che, con abuso di notazione, $\Psi \subseteq \Xi$;
- Ψ non è un sottoinsieme di Ξ e $\Xi \prec \Psi$;
- Ψ non è un sottoinsieme di Ξ e $\Xi \succ \Psi$.

La funzione *SetCmp* presentata in figura 4.25 restituisce uno fra i tre valori $\{$"$\equiv$", "$\prec$", "$\succ$"$\}$ a seconda che si verifichi uno fra i tre casi citati sopra. La struttura della funzione è ottenuta aggiungendo al sempice controllo di inclusione fra liste ordinate, il controllo sull'ordinamento lessicografico. Questo avviene gestendo i seguenti casi:

Figura 4.25: Funzione *SetCmp* per il confronto tra liste ordinate di formule

---

function $SetCmp(\Psi, \Xi)$
  return $SetCmp_r(\Psi, \Xi, \text{“}\subseteq\text{”})$.

function $SetCmp_r(\Psi, \Xi, c)$
  if $empty(\Psi)$
    then return “$\subseteq$”;
  if $empty(\Xi)$
    then if $c =$ “$\subseteq$”
        then return “$\succ$”
        else return c;
  if $head(\Psi) = head(\Xi)$
    then return $SetCmp_r(tail(\Psi), tail(\Xi), c)$;
  if $head(\Psi) \prec_{\mathcal{L}} head(\Xi)$
    then if $c =$ “$\subseteq$”
        then return “$\prec$”
        else return c;
    else return $SetCmp_r(\Psi, tail(\Xi), \text{“}\succ\text{”})$.

[**base1**] se Ψ è vuoto allora è sicuramente un sottoinsieme non stretto di Ξ e il valore di ritorno è “$\subseteq$”;

[**base2**] se Ψ non è vuoto e lo è Ξ, allora Ψ non può essere un sottoinsieme di Ξ. Se il risultato corrente (accumulatore c) è “$\subseteq$” è necessario cambiarlo in “$\succ$” (vedi definizione dell'ordinamento) altrimenti è valido un eventuale ordinamento stabilito in precedenza.

[**equivalenza**] se le “teste” dei due insiemi coincidono è necessario propagare il test sulle “code”. L'accumulatore c passa indenne attraverso questa operazione.

[**fallimento**] se il primo elemento di Ψ è più piccolo del primo elemento di Ξ allora è impossibile che Ψ sia un sottoinsieme di Ξ. Se il risultato corrente è $c =$ “$\subseteq$” bisogna cambiarlo in “$\prec$” per tenere conto del fatto che $\Psi \prec \Xi$. Negli altri casi restituiamo il valore di un ordinamento stabilito in precedenza.

[**propagazione**] se il primo elemento di Ψ è più grande del primo elemento di Ξ è ancora possibile che Ψ sia un sottoinsieme di Ξ, ma sicuramente $\Psi \succ \Xi$. La ricerca è propagata sulla “coda” di Ξ.

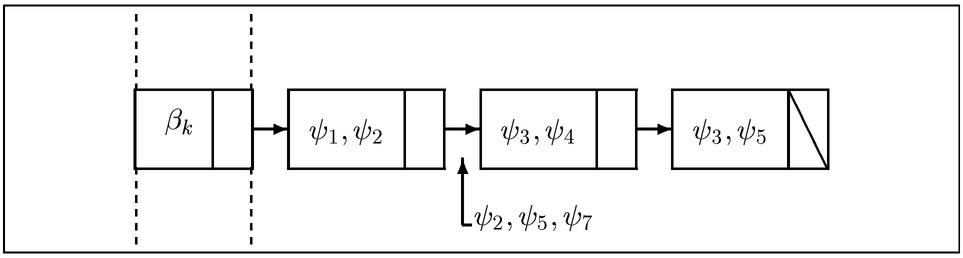

Figura 4.26: Ricerca di un elemento nella struttura Υ_s (se $\psi_2 \wedge \psi_5 \wedge \psi_7 \neg \beta_k$ è soddisfacibile la freccia indica il punto di inserimento)

Con *SetCmp* abbiamo raggiunto l'obbiettivo di costruire una funzione efficiente perché il suo tempo di esecuzione è proporzionale a $\max(\mid \Psi \mid, \mid Xi \mid)$ e quindi risulta lineare nelle dimensioni degli insiemi (liste) dati in ingresso.

A questo punto abbiamo tutti gli strumenti e le definizioni necessarie per esporre il funzionamento del memoizing per sottoinsiemi, così come è stato realizzato nelle nostre procedure di decisione. Supponiamo di avere a disposizione due strutture distinte entrambi simili a quella mostrata in figura 4.24 e ci riferiamo ad esse con i termini Υ_s e Υ_v. Utilizziamo Υ_s per memorizzare risultati relativi alla sola soddisfacibilità e Υ_v per memorizzare risultati relativi alla sola validità. In altre parole, scandendo la lista associata a $\beta \in \Gamma_\Box$ all'interno di Υ_s trovo tutti gli insiemi Δ per cui si è dimostrato che $\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg \beta$ è una formula asoddisfacibile. Viceversa scandendo la lista associata a β all'interno di Υ_v trovo tutti gli insiemi Δ per cui si è dimostrato $\bigwedge_{\alpha_i \in \Delta} \alpha_i \rightarrow \beta$ è una formula valida. Dato che le procedure di decisione sono corrette, non è possibile che uno stesso insieme sia contenuto per uno stesso β in entrambe le strutture, quindi non c'è duplicazione in questo senso. Per quanto riguarda la notazione, chiamiamo $\Upsilon_s(\psi)$ e $\Upsilon_v(\psi)$ la lista ordinata lessicograficamente di liste ordinate associata ad ogni formula $\psi \in \Gamma_\Box$ nelle due strutture.

Il motivo per cui abbiamo sdoppiato il supporto di memorizzazione è la necessità di ottimizzare le consuete operazioni di ricerca e inserimento. A nostro avviso, è possibile ottenere realizzazioni molto più efficienti delle primitive di accesso, costruendo ciascuna struttura nella maniera più opportuna per il particolare tipo di inferenza che deve realizzare. Presentiamo allora due semplici esempi che oltre a chiarire il funzionamento del memoizing, mettono in evidenza le peculiarità delle due strutturè.

Come si può vedere nella figura 4.26 relativa ad un frammento di Υ_s, le liste associate ad ogni formula dell'insieme $\Gamma_\Box$ sono ordinate in senso *ascendente*, cioè andando dal primo puntatore alla coda delle liste si incontrano liste di formule lessicograficamente più grandi rispetto a tutte quelle che le precedono. Consideriamo un esempio di ricerca all'interno di Υ_s, supponendo che sia $\Psi = \{\psi_2, \psi_5, \psi_7\}$ e di dover svolgere il test ricorsivo

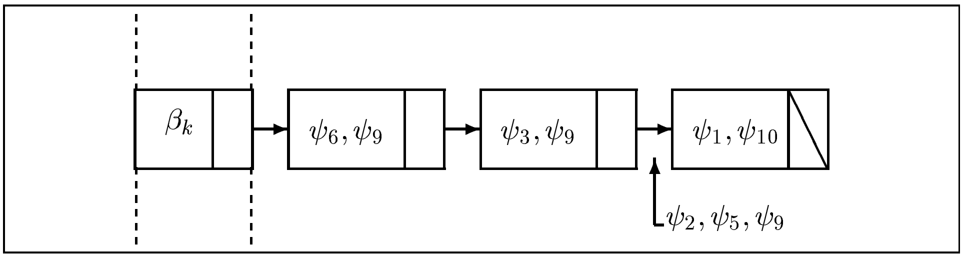

Figura 4.27: Ricerca di un elemento nella struttura Υ_v (se $\psi_2 \wedge \psi_5 \wedge \psi_9 \to \beta_k$ è valida la freccia indica il punto di inserimento)

$\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta_k$. Devo controllare se all'interno di Υ_s non sia memorizzato, in corrispondenza di β_k, un insieme Δ tale che $\Psi \subseteq \Delta$. Incomincio il controllo dal primo elemento della lista associata a β utilizzando la funzione *SetCmp* (vedi figura 4.25). Mi fermo in due casi:

- l'insieme Δ prelevato da $\Upsilon_s(\beta_k)$ è un sovrainsieme di Ψ: questo vuol dire che anche $\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta_k$ è soddisfacibile;
- l'insieme Δ prelevato da $\Upsilon_s(\beta_k)$ è lessicograficamente più grande di Ψ: non è possibile che nella restante parte della lista ci siano sovrainsiemi per Ψ

Quindi la ricerca prosegue sino a che gli insiemi in $\Upsilon_s(\beta_k)$ sono lessicograficamente minori di Ψ (si veda la figura 4.28 per dettagli) per poi fermarsi in uno dei due casi sopra citati. Nel nostro esempio, nessun Δ in $\Upsilon_s(\beta_k)$ è un sovrainsieme per Φ. Dato il particolare ordine della lista, la ricerca termina esattamente nell'eventuale punto di inserimento per Ψ. Nel caso in cui si provasse che $\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta_k$ è soddisfacibile potrebbe essere effettuata un'inserzione ordinata senza nessun costo aggiuntivo.

Consideriamo ora la figura 4.27 relativa ad un frammento di Υ_v in cui le liste associate ad ogni formula dell'insieme $\Gamma_\Box$ sono ordinate in senso *discendente*, cioè andando dal primo puntatore alla coda delle liste si incontrano liste di formule lessicograficamente più piccole rispetto a tutte quelle che le precedono. Per il nostro esempio consideriamo $\Psi = \{\psi_2, \psi_5, \psi_9\}$ a cui corrisponde il test ricorsivo $\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta_k$. Questa volta dobbiamo controllare se all'interno di Υ_s non sia memorizzato, in corrispondenza di β_k, un insieme Δ che risulti essere un sottoinsieme di Ψ. Il controllo inizia dal primo elemento della lista associata a β_k utilizzando sempre la funzione *SetCmp* (è sufficiente invertirne i parametri per ottenere un funzionamento corretto anche per Υ_v). Naturalmente, mi fermo in due casi:

- l'insieme Δ prelevato da $\Upsilon_v(\beta_k)$ è un sottoinsieme di Ψ: e quindi risulta anche $\bigwedge_{\alpha_i \in \Psi} \alpha_i \wedge \neg\beta_k$ è insoddisfacibile;

- l'insieme Δ prelevato da $\Upsilon_v(\beta_k)$ è lessicograficamente più piccolo di Ψ: non è possibile che nella restante parte della lista ci siano sottoinsiemi per Ψ.

Esattamente all'opposto rispetto al caso di Υ_s, la ricerca prosegue sino a che gli insiemi in $\Upsilon_v(\beta_k)$ sono lessicograficamente più grandi di Ψ (si veda la figura 4.29 per dettagli). Nel caso del nostro esempio, nessun Δ in $\Upsilon_v(\beta_k)$ è un sottoinsieme di Φ. Data la particolare organizzazione della lista, la ricerca termina comunque nell'eventuale punto di inserimento per Ψ. Nel caso in cui si provasse che $\bigwedge_{\alpha_i \in \Psi} \alpha_i \to \beta_k$ è valida potremmo inserire Ψ senza nessuna ulteriore ricerca.

Operativamente, per realizzare le primitive di memoizing nelle logiche che trattiamo in questa sezione, abbiamo bisogno di due procedure che siano in grado di gestire le ricerche che abbiamo delineato in maniera informale nei due esempi precedenti. Le procedure *SupSet* e *SubSet* che presentiamo in figura 4.29 e 4.28 servono proprio per fornire tale supporto. Infatti, entrambe eseguono una ricerca appropriata in una lista di liste e ritornano una risposta affermativa o un punto di inserimento per un 'eventuale risultato.

Figura 4.28: Ricerca di sovrainsiemi per Ψ in una lista $[\Delta]$

function $SubSet(\Psi, [\Delta])$
  if $empty([\Delta])$ **then return** $\uparrow [\Delta]$;
  $r := SetCmp(\Psi, head([\Delta]))$
  if $r =$ "$\subseteq$" **then return** Ψ;
  if $r =$ "$\prec$" **then return** $\uparrow [\Delta]$
    else $SubSet(\Psi, tail([\Delta]))$ **.**

La funzione in figura 4.28 controlla se la lista ordinata di formule Ψ è un sottoinsieme di qualcuna delle liste di formule contenute in $[\Delta]$. Affinché *SubSet* funzioni correttamente è necessario che $[\Delta]$ sia ordinato in senso ascendente (come accadeva epr tutte le liste in Υ_s). Sia nel caso in cui il fondo di $[\Delta]$ viene raggiunto senza successo ($empty([\Delta])$), sia nel caso in cui è inutile continuare a scandire $[\Delta]$ alla ricerca di sovrainsiemi per Ψ ($\Psi \prec head([\Delta])$), la funzione ritorna un punto di inserzione nella lista $[\Delta]$ (rappresentato con "$\uparrow$").

Viceversa, la funzione in figura 4.29 controlla se la lista ordinata di formule Ψ è un sovrainsieme di qualcuna delle liste di formule contenute in $[\Delta]$. La struttura di *SupSet* è esattamente simmetrica rispetto a quella di *SubSet*. $[\Delta]$ è ordinato in senso discendente dato che questa è la carateristica tutte le liste in Υ_v. Sia nel caso in cui $[\Delta]$ termina senza successo ($empty([\Delta])$), sia nel caso in cui è inutile continuare a scandire $[\Delta]$ alla ricerca di sottoinsieme

Figura 4.29: Ricerca di sottoinsiemi per Ψ in una lista di $[\Delta]$

---

function $\mathit{SupSet}(\Psi, [\Delta])$
  if $\mathit{empty}([\Delta])$ **then return** $\uparrow [\Delta]$;
  $r := \mathit{SetCmp}(\mathit{head}([\Delta]), \Psi)$
  if $r =$ "$\subseteq$" **then return** Ψ;
  if $r =$ "$\prec$" **then return** $\uparrow [\Delta]$
    else $\mathit{SupSet}(\Psi, \mathit{tail}([\Delta]))$ **.**

---

per Ψ ($\mathit{head}([\Delta]) \prec \Psi$), la funzione ritorna un punto di inserzione nella lista $[\Delta]$ (rappresentato ancora con "$\uparrow$").

Ora che abbiamo sviluppato completamente la struttura dati per il memoizing e le primitive che ne consentono l'accesso, possiamo realizzare le versioni con memoizing dei decisori. Incominciamo con la procedura $\mathrm{EC}\textsc{sat}_A$ presentata in figura 4.30. La procedura che presentiamo è il risultato dell'aggiunta di ben due diverse strutture di memoizing e, di conseguenza, di due gruppi di primitive, alla struttura descritta nella sezione 4.3.5.

Consideriamo allora il primo gruppo dato dalle primitive *EC-get*, *EC-store* e *EC-memoizing-decision* che gestiscono il test ricorsivo $\mathrm{EC}\textsc{sat}(\neg\alpha \wedge \beta)$. Le prime due sono, rispettivamente, la funzione di accesso e la funzione di memorizzazione in una struttura che chiamiamo Ω_C. Data la forma del test ricorsivo supportato da Ω_C la scelta più ovvia per la realizzazione della struttura è una semplice matrice, del tutto simile alle strutture utilizzate per le procedure della sezione 4.5.1. La realizzazione di *EC-get* e *EC-store* segue lo stesso paradigma, una volta che si è stabilito che il valore nella cella di Ω alla coordinata (ψ_1, ψ_2) è relativo alla validità di $\psi_1 \to \psi_2$. Per *EC-memoizing-decision* vale ancora la considerazione che, a priori, il comportamento migliore sembra essere quello per cui tale euristica torna sempre *True*.

Questo primo gruppo di primitive consente di aumentare l'efficienza del test ricorsivo $\mathrm{EC}\textsc{sat}(\neg\alpha \wedge \beta)$ che coinvolge sempre coppie di formule. Le stesse considerazioni che avevamo fatto a proposito, ad esempio, della logica *EM* valgono anche in questo caso e le scelte realizzative non sono diverse, per cui non entriamo nel dettaglio di questa prima parte della procedura. Molto più interessante è invece capire come sia possibile gestire la parte relativa al test ricorsivo $\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta$. Supponiamo di avere a nostra disposizione due strutture Υ_s e Υ_v così come le abbiamo introdotte in precedenza.

Le primitive per il memoizing andranno realizzate secondo i seguente schema concettuale. *EC-get*^{s} riceve in ingresso un certo insieme Δ di formule e un certo β. Il primo passo è quello di ordinare Δ e successivamente

Figura 4.30: Pocedura ECSAT$_A$ con memoizing

---

function ECSAT$_A$($\bigwedge_i \Box\alpha_i \;\wedge\; \bigwedge_j \neg\Box\beta_j \;\wedge\; \gamma$)
  if {there is any conjunct $\Box\alpha_i$}
    then foreach conjunct $\Box\beta_j$ **do**
        $\Delta = \{\alpha_1, \ldots, \alpha_n\}$;
        foreach conjunct $\Box\alpha_i$ **do**
          $r := EC\text{-}get(\alpha_i, \beta_j)$;
          if $r = \bot$
            then $r :=$ **not** ECSAT($\neg\alpha_i \wedge \beta_j$);
                if $EC\text{-}memoizing\text{-}decision(\alpha_i, \beta_j, r)$
                  then $EC\text{-}store(\alpha_i, \beta_j, r)$;
          if $r = False$ **then** $\Delta = \Delta - \{\alpha_i\}$;
        if $\Delta = \emptyset$ **then return** *True*;
        $r := EC\text{-}get^s(\Delta, \beta_j)$;
        if $r = \bot$
          then $r :=$ **not** ECSAT($\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j$);
              if $EC\text{-}memoizing\text{-}decision^s(\Delta, \beta_j, r)$
                then $EC\text{-}store^s(\beta_j, r)$;
        if $r = True$ **then return** *False*;
  return *True*.

---

controllare se Δ è contenuto in qualche insieme (lista) della lista $\Upsilon_s(\beta)$. In caso affermativo $EC\text{-}get^s$ deve ritornare *False* in quanto l'implicazione $\bigwedge_{\alpha_i \in \Delta} \alpha_i \to \beta$ non è valida. In caso contrario $EC\text{-}get^s$ deve conservare il punto di inserzione per un eventuale risultato e passare acontrollare se Δ contiene qualche insieme (lista) della lista $\Upsilon_s(\beta)$. Se così è, allora $EC\text{-}get^s$ deve ritornare *True* in quanto l'implicazione $\bigwedge_{\alpha_i \in \Delta} \alpha_i \to \beta$ non è valida. Se anche questo controllo fallisce, viene restituito $\bot$ e si conserva il punto di inserzione. Chiaramente $EC\text{-}store^s$ viene invocata solo quando $EC\text{-}get^s$ ha ritornato $\bot$ e quindi entrambe i puntatori per l'inserzione nella struttura di memoizing sono pronti. È sufficiente fare un test per conoscere la natura del risultato da memorizzare per collocarlo nella struttura più opportuna. In figura 4.31 presentiamo la coppia di primitive $EC\text{-}get^s$ e $EC\text{-}store^s$ realizzate con il comportamento appena descritto.

Ci sono alcune particolarità nella realizzazione delle primitive per il memoizing su insiemi che vale la pena di chiarire. Notiamo che $EC\text{-}get^s$ ordina l'insieme Δ (utilizzando l'ordine "$\prec_{\mathcal{L}}$" definito prima) e immagazzina il risultato in una variabile globale Δ_g. Questo serve per evitare che $EC\text{-}store^s$ debba ripetere l'operazione in quanto, comunque, le liste di formule aggiunte

Figura 4.31: Primitive per il memoizing in *EC*

---

function $EC\text{-}get^s(\Delta, \psi)$
  $\Delta_g := sort(\Delta)$;
  $s\text{-}point = SubSet(\Delta_g, \Upsilon_s(\psi))$;
  if $s\text{-}point = \Delta_g$ **then return** *False*;
  $v\text{-}point = SupSet(\Delta_g, \Upsilon_v(\psi))$;
  if $v\text{-}point = \Delta_g$ **then return** *True*
                else return $\perp$.

  procedure $EC\text{-}store^s(\psi, r)$
    if r **then** $insert(\Delta_g, \Upsilon_v(\psi), v\text{-}point)$
       else $insert(\Delta_g, \Upsilon_s(\psi), s\text{-}point)$.

---

nelle strutture di memoizing devono essere ordinate in modo da non alterare il risultato di ricerche successive. Ci sono poi altre due variabili condivise da $EC\text{-}get^s$ e $EC\text{-}store^s$ che rappresentano i due punti di inserzione ordinata nelle strutture di memoizing. Anche in questo caso, la condivisione dell'informazione sul punto di inserimento, consente di ottenere un funzionamento efficiente senza che dettagli interni al memoizing debbano essere gestiti dalla procedura $\text{EC}\textsc{sat}_A$.

Un ultimo commento riguarda le primitive *insert* e *sort* che abbiamo utilizzato nel presentare lo pseudo codice per $EC\text{-}store^s$ e $EC\text{-}get^s$. Il compito di *sort* è semplicemente quello di ordinare un insieme di formule in base alla relazione "$\prec_{\mathcal{L}}$" che abbiamo definito in precedenza. Il compito di *insert* è quello inserire una lista di formule (nel nostro caso Δ_g) all'interno di una lista di liste ($\Upsilon_{s,v}$) dato un particolare punto di inserzione (*s,v-point*).

Non è difficile, sulla base di $\text{EC}\textsc{sat}_A$, costruire la parte modale del decisore per *ECN*. In figura 4.32 presentiamo $\text{ECN}\textsc{sat}_A$ che è costruito a partire da $\text{EC}\textsc{sat}_A$ modificando opportunamente la struttura del codice e le primitive di memoizing.

Per quanto riguarda il codice, notiamo che abbiamo rimosso l'iniziale controllo sulla presenza di formule modali assegnate a vero (formule α) e il test intermedio $\Delta = \emptyset$. Questo è esattamente ciò che abbiamo fatto nella sezione 4.3.6 per la procedura senza memoizing e la motivazione è sempre che in tutte le logiche che contengono N come teorema, non è più vero che un assegnamento costituito di soli β è banalmente soddisfacibile.

Per quanto riguarda le strutture di memoizing, ritroviamo sia Υ_s sia Υ_v che supportano il memoizing su insiemi, sia la matrice Ω_{CN} che contiene i risultati del test ricorsivo $\text{ECN}\textsc{sat}(\neg\alpha \wedge \beta)$. Le primitive per l'accesso alle

Figura 4.32: Procedura $\text{ECN}\textsc{sat}_A$

---

function $\text{ECN}\textsc{sat}_A(\bigwedge_i \Box\alpha_i \ \wedge\ \bigwedge_j \neg\Box\beta_j \ \wedge\ \gamma)$
  foreach conjunct $\Box\beta_j$ **do**
    $\Delta = \{\alpha_1, \ldots, \alpha_n\}$;
    foreach conjunct $\Box\alpha_i$ **do**
      $r := \textit{ECN-get}(\alpha_i, \beta_j)$;
      if $r = \bot$
        then $r :=$ **not** $\text{ECN}\textsc{sat}(\neg\alpha_i \wedge \beta_j)$;
            if $\textit{ECN-memoizing-decision}(\alpha_i, \beta_j, r)$
              then $\textit{ECN-store}(\alpha_i, \beta_j, r)$;
      if $r = \textit{False}$ **then** $\Delta = \Delta - \{\alpha_i\}$;
    $r := \textit{ECN-get}^s(\Delta, \beta_j)$;
    if $r = \bot$
      then $r :=$ **not** $\text{ECN}\textsc{sat}(\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j)$;
          if $\textit{ECN-memoizing-decision}^s(\Delta, \beta_j, r)$
            then $\textit{ECN-store}^s(\Delta, \beta_j, r)$;
    if $r = \textit{True}$ **then return** *False*;
  return *True*.

---

prime due strutture hanno dovuto essere modificate per tenere conto del fatto che l'insieme ricevuto in ingresso può essere eventualemte vuoto. In tal caso significa che stiamo eseguendo un test del tipo $\text{EC}\textsc{sat}(\neg\beta)$ e quindi dobbiamo accedere alla struttura predisposta per memorizzare la validità delle singole formule. Come accadeva per EN ed EMN (vedi sezione 4.5.1 il risultato relativo alla validtià del generico $\psi \in sub_\Box(\varphi)$ viene memorizzata nelle celle $\Omega_{CN}(\psi, \psi)$ cioè sulla diagonale della matrice Ω_{CN}.

Le primitive di accesso a Υ_s e Υ_v sono presentate in figura 4.33. Notiamo che oltre alla possibilità di gestire insiemi vuoti, abbiamo aggiunto in $\textit{ECN-store}^s$ la possibilità di memorizzare *False* in corrispondenza della cella $\Omega_{CN}(\psi, \psi)$, quando sappiamo che una formula delt tipo $\ldots \wedge \neg\psi$ è soddisfacibile.

Come accade per EC, anche nel caso di ECN le primitive per il memoizing in Ω_{CN} possono essere facilmente dedotte da quelle, ad esempio, introdotte nella sezione 4.5.1 per EMN. La cosa più importante è che $\textit{ECN-store}$ sia costruita in maniera tale per cui se $\neg\alpha \wedge \beta$ è soddisfacibile, allora la cella $\Omega_{CN}(\alpha, \alpha)$ viene assegnata a *False*.

Chiudiamo questa sezione presentando lo pseudo codice per $\text{R}\textsc{sat}_A$ con memoizing. Questa procedura è molto più compatta rispetto a quelle viste per EC o ECN, in quanto è scomparsa la parte relativa alla gestione del

Figura 4.33: Primitive per il memoizing in ECN

```
function ECN-get^s(Δ, ψ)
  if Δ = ∅
    then Δ_g = ∅;
         return Ω_CN(ψ, ψ);
  Δ_g := sort(Δ);
  s-point = SubSet(Δ_g, ϒ_s(ψ));
  if s-point = Δ_g then return False;
  v-point = SupSet(Δ_g, ϒ_v(ψ));
  if v-point = Δ_g then return True
                   else return ⊥.

    procedure ECN-store^s(ψ, r)
      if Δ_g = ∅ then Ω_CN(ψ, ψ) := r;
                 else if r then insert(Δ_g, ϒ_v(ψ), v-point)
                           else Ω_CN(ψ, ψ) := r;
                                insert(Δ_g, ϒ_s(ψ), s-point)  .
```

test ricorsivo sulla soddisfacibilità delle formule $\neg\alpha \wedge \beta$.

Figura 4.34: Pocedura $\textsc{Rsat}_A$ con memoizing

```
function RSAT_A(⋀_i □α_i ∧ ⋀_j ¬□β_j ∧ γ)
  foreach conjunct □β_j do
    B = {α_1, ..., α_n};
    if B = ∅ then return True;
    r := R-get^s(B, β_j);
    if r = ⊥
      then r := not RSAT(⋀_{α_i∈B} α_i ∧ ¬β_j);
           if R-memoizing-decision^s(B, β_j, r)
             then R-store^s(B, β_j, r);
    if r = True then return False;
  return True.
```

Non ci sono particolari osservazioni da fare sul comportamento del memoizing, in quanto l'implementazione delle primitive $R\text{-}get^s$ e $R\text{-}store^s$ e identica a quelle presentata per la logica EC. Anche in questo caso, come per tutte le procedure sinore descritte, l'euristica *R-memoizing-decision* è tale da ritornare sempre *True*

4.5.3 Un decisore per K

Come abbiamo visto nel capitolo 2 la logica modale K rappresenta il punto di congiunzione tra la semantica con strutture $\mathcal{MS}$ e la semantica a mondi possibili che caratterizza tutte le logiche modali normali (o standard). In particolare, almeno per quanto ne sappiamo, è l'unica fra le logiche prese in esame per cui esistono procedure di decisione realizzate in pratica. Pertanto è estremamente interessante confrontare la struttura del decisore da noi proposto con altri decisori realizzati direttamente per K. La procedura che noi proponiamo, essendo il risultato di una progressiva "crescita" all'interno delle logiche classiche, è sviluppata con le tecniche messe a punto nelle sezioni 4.5.1 e 4.5.2. Siamo allora interessati a saggiare la bontà del nostro approccio comparandoci con procedure di decisione che sono nate direttamente per automatizzare la deduzione in logiche modali normali.

Come abbiamo già accennato nella sezione 4.3.8 la nostra procedura di base per K non è altro che una restrizione al caso monoagente della procedura KSAT introdotta per la prima volta in [Giunchiglia e Sebastiani, 1996a]. Nel seguito, per distinguere KSAT dalla nostra realizzazione, utilizziamo per quest'ultima il termine EMCNSAT. Abbiamo deciso di scegliere KSAT come termine di paragone non solo per il fatto che si tratta di un decisore SAT based la cui realizzazione è per molti aspetti molto vicina a quella delle nostre procedure. Oltre a questo c'è il fatto che KSAT è stato dimostrato superiore ad altri sistemi che realizzano algoritmi di decisione per K. In [Giunchiglia e Sebastiani, 1996b] vengono presentati i risultati sperimentali che attestano la superiorità di KSAT nei confronti di sistemi considerati allo stato dell'arte nel campo della deduzione automatica in logiche modali e terminologiche. Dato che la particolare forza di KSAT sembra risiedere essenzialmente nella componente proposizionale del ragionamento e visto che le nostre procedure utilizzano la stessa tecnologia, il confronto sembra spostarsi sulla parte relativa all'inferenza modale.

Cominciamo con la descrizione della procedura KSAT. Tutto il materiale che esponiamo in questa sezione è stato tratto, senza modifiche che ne alterino sostanzialmente il significato, da [Sebastiani, 1997].[4] Data l'estrema somiglianza della parte proposizionale degli algoritmi esponiamo solamente la parte relativa all'inferenza modale, ristretta al caso in cui vi sia un unico agente.

Dato che anche KSAT è un decisore basato su DPLL, aggiungendo la procedura presentata in figura 4.35 al framework descritto nella sezione 4.2

[4] Per "modifiche non sostanziali" abbiamo inteso cambiamenti di notazione per mantenere la coerenza con quella da noi utilizzata

Figura 4.35: Procedura di decisione KSAT_A

---

function $\text{KSAT}_A(\bigwedge_i \Box\alpha_i \ \wedge \ \bigwedge_j \neg\Box\beta_j \ \wedge \ \gamma)$
  $\Delta = \{\beta_1 \ldots \beta_m\}$
  if $\text{KSAT}_F(\bigwedge_i \alpha_i, \top, \Delta) = \emptyset$ **then return** *True*
    else return *False* .

function $\text{KSAT}_F(\alpha, \eta, \Delta)$
  if $\alpha = \top$
    then return $\{\beta \in \Delta \mid \text{KSAT}(\eta \wedge \neg\beta) = \mathit{False}\}$;
  if $\alpha = \bot$
    then return Δ;
  if {a unit clause (l) is in α}
    then return $\text{KSAT}_F(\mathit{assign}(l, \alpha), \eta \wedge l, \Delta)$;
  $l := \mathit{choose\text{-}literal}(\alpha)$;
  $\Delta_1 := \text{KSAT}_F(\mathit{assign}(l, \alpha), \eta \wedge l, \Delta)$;
  if $\Delta_1 = \emptyset$
    then return Δ_1
     else return $\text{KSAT}_F(\mathit{assign}(\neg l, \alpha), \eta \wedge \neg l, \Delta)$.

---

si ottiene sostanzianzialmente lo stesso decisore presentato come "KSAT versione 4" in [Sebastiani, 1997], cioè la versione più potente della procedura KSAT. Sempre in [Sebastiani, 1997], leggiamo che la quarta versione di KSAT incorpora le seguenti caratteristiche:

[**ordinamento**] tutte le formule in ingresso al decisore vengono ordinate in modo da evitare la generazione di assegnamenti contenenti permutazioni di una stessa sottoformula assegnati a valori diversi;

[**fattorizzazione**] è la metodologia descritta in figura 4.35 e deve il suo nome al fatto che si tenta una fattorizzazione del test ricorsivo;

[**test intermedi**] corrisponde a quello che nella sezione 4.2 abbiamo chiamato "early pruning".

Sia l'esecuzione di test intermedi, sia l'ordinamento sono realizzati anche nei nostri decisori. Anche se non lo abbiamo presentato nella sezione 4.2 l'ordinamento è parte integrante delle nostre procedure di decisione per gli stessi motivi addotti in [Sebastiani, 1997] a proposito di KSAT. Il particolare algoritmo di ordinamento da noi utilizzato è discusso nella sezione 5.2.2.

La fattorizzazione dei test ricorsivi è invece assente in tutti i nostri decisori (per i motivi esposti nella sezione 5.3.2) e pertanto ci soffermiamo brevemente per discuterla. Se pensiamo alla versione di KSAT presentata nella sezione 4.3.8 vediamo che KSAT_A invoca ripetutamente KSAT passando come argomenti formule nella forma $\bigwedge_i \alpha_i \wedge \neg\beta_1, \ldots, \bigwedge_i \alpha_i \wedge \neg\beta_m$. Tutte queste formule hanno la congiunzione $\bigwedge_i \alpha_i$ in comune e quindi, all'atto della generica chiamata ricorsiva KSAT cerca un assegnamento per il generico test ricorsivo senza cercare di riutilizzare nessuno degli assegnamenti precedenti.

La procedura KSAT_A in figura 4.35 risolve il problema cercando di trovare un assegnamento η per la formula $\bigwedge_i \alpha_i$ (rappresentata con α) per poi utilizzarlo per eliminare dall'insieme Δ tutte le formule β che sono consistenti con l'assegnamento trovato. La ricerca di assegnamenti continua sino a che Δ è vuoto ($\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ è consistente), oppure nessun altro assegnamento è disponibile ($\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ è inconsistente).

Ai fini della nostra discussione può essere sufficiente la descrizione sommaria di KSAT che abbiamo dato sinora. Per ulteriori approfondimenti e per una descrizione più dettagliata facciamo riferimento a [Sebastiani, 1997].

La parte relativa all'inferenza modale EMCNSAT_A del decisore da noi realizzato per la logica K è presentata in figura 4.36. Oltre all'assenza del meccanismo di fattorizzazione, la procedura si distingue da KSAT_A per il fatto che realizza un meccanismo di memoizing del tutto analogo a quello che abbiamo utilizzato come supporto ai decisori per EC, ECN e ECM.

Figura 4.36: Pocedura EMCNSAT_A con memoizing

---

function $\text{EMCNSAT}_A(\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma)$
  foreach conjunct $\Box\beta_j$ **do**
    $\Delta = \{\alpha_1, \ldots, \alpha_n\}$;
    $r := \mathit{EMCN\text{-}get}^s(\Delta, \beta_j)$;
    if $r = \bot$
      then $r :=$ **not** $\text{EMCNSAT}(\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j)$;
        if $\mathit{EMCN\text{-}memoizing\text{-}decision}^s(\Delta, \beta_j, r)$
          then $\mathit{EMCN\text{-}store}^s(\Delta, \beta_j, r)$;
    if $r = \mathit{True}$ **then return** False;
  return True.

Dal nostro punto di vista, la procedura di decisione EMCNSAT_A è ottenuta da EMCSAT_A tenendo in conto che, essendo valido in K l'asioma N, non è più possibile dichiarare banalmente consistente un assegnamento privo di formule α. Al più, in assenza di atomi modali assegnati a vero, il test ricorsivo $\text{EMCNSAT}(\bigwedge_{\alpha_i \in \Delta} \alpha_i \wedge \neg\beta_j)$ si riduce a $\text{EMCNSAT}(\neg\beta_j)$ per ogni β_j.

Le primitive per il memoizing devono quindi essere simili a quelle per

ECN e cioè tali da saper gestire insiemi Δ eventualmente vuoti. In figura 4.37 presentiamo le primitive $EMCN$-get^s e $EMCN$-$store^s$ che sono in grado di accettare insiemi Δ vuoti e interpetarli correttamente. Un'osservazione è necessaria a proposito della struttura dati utilizzata per memorizzare i valori relativi alla validità di singole formule. Notiamo che, a differenza di quanto accadeva per ECN, si tratta di un *vettore* w_K piuttosto che di una matrice. Questo è ovvio se si considera il fatto che è sufficiente un numero pari a $\mid sub_\Box(\varphi) \mid$ celle di memoria per contenere l'informazione riguardante ogni sottoformula di φ. Nel caso di EN, EMN e ECN non era necessario allocare tael vettore, avendo a disposizione lo stesso spazio, inutilizzato, sulla diagonale delle matrici Ω_N, Ω_{MN} e Ω_{CN}.

Figura 4.37: Primitive per il memoizing in $EMCN$

---

function ECN-$get^s(\Delta, \psi)$
  if $\Delta = \emptyset$
    then $\Delta_g = \emptyset$;
        return $\omega_K(\psi)$;
  $\Delta_g := sort(\Delta)$;
  $s\text{-}point = SubSet(\Delta_g, \Upsilon_s(\psi))$;
  if $s\text{-}point = \Delta_g$ **then return** *False*;
  $v\text{-}point = SupSet(\Delta_g, \Upsilon_v(\psi))$;
  if $v\text{-}point = \Delta_g$ **then return** *True*
            else return $\perp$**.**

  procedure ECN-$store^s(\psi, r)$
    if $\Delta_g = \emptyset$ **then** $\omega_K(\psi) := r$;
          else if r **then** $insert(\Delta_g, \Upsilon_v(\psi), v\text{-}point)$
                else $\omega_K(\psi) := r$;
                  $insert(\Delta_g, \Upsilon_s(\psi), s\text{-}point)$.

Concludiamo questa sezione con un confronto "a distanza" tra le due procedure di decisione presentate e cioè KSAT e EMCNSAT. Ritorniamo nel capitolo 5 e in particolare nella sezione 5.3.2 sulle differenze tra le due procedure quando esponiamo i risultati dei test comparativi su dati sperimentali. nel frattempo possiamo già fare le seguenti osservazioni:

- KSAT è la prima procedura di decisione effettivamente realizzata che sfrutti la tecnologia SAT based. Ha dimostrato di essere molto veloce (vedi [Giunchiglia e Sebastiani, 1996b]) anche se comparata con sistemi allo stato dell'arte, mostrando comportamenti molto interessanti, sia da punto di vista qualitativo sia dal punto di vista quantitativo. Nella versione completa di tutte le ottimizzazioni è il decisore per la logica K più efficiente che sia noto in letteratura. Si tratta di un algoritmo PSPACE nel senso che la memoria occupata cresce polinomialmente con le dimensioni della formula da decidere.

- EMNCsat coincide, nella sua versione base, con la versione base di Ksat e realizza, al pari di quest'ultima, test su assegnamenti incompleti e ordinamento di formule. Non fa uso della fattorizzazione ma utilizza tecniche di memoizing. In particolare i punti di forza di EMCNsat vanno, a nostro avviso, ricercati in:
  - l'utilizzo del memoizing consente di evitare del tutto la ripetizione di test ricorsivi per formule identiche;
  - le strutture di memoizing incorporano possibilità deduttive;
  - l'inferenza compiuta nel passo di early pruning, sempre grazie al memoizing, non è inutile anche in caso di fallimento.

Come abbiamo visto nella sezione 4.5.2 non abbiamo alcuna garanzia che lo spazio allocato per il memoizing non esploda esponenzialmente. Ovviamente, nel caso in cui questo accada in pratica, nessuno dei vantaggi proposti è tale da compensare il fatto che lo spazio allocato diverge rapidamente al crescere della dimensione della formule in ingresso.

Capitolo 5

Risultati sperimentali

Legge di Hofstadter: ci vuole sempre più tempo di quanto si pensi, anche tenendo conto della legge di Hofstadter.
[D.R. Hofstadter - "Gödel, Escher, Bach"]

In questo capitolo presentiamo i risultati relativi alla sperimentazione dei decisori introdotti nel capitolo precedente e la metodologia di testing utilizzata per valutarne le prestazioni. In particolare, nella sezione 5.1 esponiamo le motivazioni e i risultati presenti in letteratura che ci hanno portato ad intraprendere tale analisi. Nella sezione 5.2 descriviamo il generatore di formule e i moduli di *preprocessing* per l'ordinamento e la semplificazione delle formule in ingresso. Sempre nella stessa sezione, discutiamo la scelta dei parametri utilizzati per lo studio del comportamento dei decisori e il metodo di analisi dei risultati. Infine nella sezione 5.3 presentiamo i *case study* relativi alle procedure di decisione ESAT e EMCNSAT.

5.1 L'approccio sperimentale

Nel momento stesso in cui le applicazioni per l'intelligenza artificiale sono portate al di fuori del campo della pura speculazione filosofica o psicologica, è necessario che si confrontino con la realtà dei problemi che intendono risolvere. Qualsiasi modello di un particolare aspetto del ragionamento umano deve, alla prova dei fatti, dimostrare di essere adatto allo scopo per cui era stato concepito. In pratica si evidenzia spesso il problema di decidere, date due o più procedure che automatizzano un certo tipo di ragionamento, capire quale sia la più efficiente. In generale, dato l'alto contenuto di decisioni euristiche tipico delle applicazioni di deduzione automatica, l'unico approccio possibile è quello sperimentale. Se una procedura ha maggiore successo di un'altra nell'affrontare un certo insieme di problemi considerato "tipico" per l'applicazione, allora, da un punto di vista empirico, possiamo affermare che la prima è più efficiente.

Dal punto di vista pratico, questa idea deve fare i conti con il fatto che molti dei problemi rilevanti nel campo dell'intelligenza artificiale, almeno nella loro forma più generale, hanno complessità più che polinomiali (si veda [Garey e Johnson, 1979] per un elenco dettagliato). Come abbiamo mostrato nel capitolo 3, l'argomento da noi affrontato non fa eccezione: il problema della soddisfacibilità in una qualsiasi delle logiche Λ ha complessità NP o addirittura superiore.

L'esplosione combinatoria dei problemi ha da sempre suggerito l'esistenza di un compromesso tra efficienza da un lato e assenza di certi requisiti dall'altro. Nel caso delle procedure di decisione il requisito che più spesso è stato sacrificato per ottenere efficienza è stato la *completezza* del ragionamento realizzato. In letteratura si trovano esempi di decisori incompleti che hanno dimostrato essere molto più efficienti rispetto alle procedure complete. È il caso, ad esempio, di GSAT un decisore incompleto per SAT (descritto tra l'altro in [Sebastiani, 1997]), che è molto più veloce dello stesso algoritmo DPLL, ma non è in grado di dare una risposta se la formula in ingresso non è soddisfacibile.

D'altro canto, come abbiamo mostrato nella sezione 4.4 i nostri decisori sono tutti, oltre che corretti, anche completi rispetto alle logiche Λ. Pertanto, tutte le procedure da noi realizzate hanno tempi di esecuzione che, nel caso pessimo, sono esponenziali nelle dimensioni della formula da decidere.

A questo punto potremmo chiederci quale senso abbia valutare le prestazioni di programmi che sembrano essere segnati alla nascita da problemi di efficienza. La risposta a questa domanda ci viene dai lavori presentati nel campo della ricerca su SAT (si veda, per esempio, [Mitchell *et al.*, 1992]) e più recentemente, dai risultati di [Giunchiglia e Sebastiani, 1996b] che confrontano diversi decisori per la logica K su base essenzialmente sperimentale.

Nelle parole di [Mitchell *et al.*, 1992], ha senso chiederci se sia ragionevole utilizzare procedure di ragionamento completo (in questo caso per SAT) e soprattuto chiederci se i casi veramente difficili siano frequenti oppure siano sempre il risultato di particolari codifiche legate a qualche applicazione specifica. Sempre nello stesso articolo, leggiamo anche che è difficile, in generale, rispondere a queste domande su base *analitica*: anche in questo caso la nostra ultima risorsa sembra essere proprio l'analisi sperimentale.

I lavori sulla logica K seguono questo filone concettuale e propongono una metodologia di testing adatta per molte logiche normali e terminologiche (vedi anche [Giunchiglia *et al.*, 1996] per approfondimenti), sulla base della quale è possibile trarre conclusioni relative all'efficienza dei decisori utilizzando un'analisi empirica delle prestazioni.

Questi approcci rappresentano la nuova tendenza in atto nel campo dell'intelligenza artificiale per cui, dalla ricerca di soluzioni più o meno efficienti e comuque limitate nel caso generale, si è passati all'indagine condotta con i mezzi dell'analisi sperimentale sulla "reale" complessità dei problemi combinatori.

La motivazioni che ci hanno portato a seguire un approccio sperimentale per l'analisi dei nostri decisori sono due. La prima è che tutte le procedure di decisione presentate, eccetto ovviamente EMCNSAT, sono uno strumento di indagine nella complessità del ragionamento non onnisciente. In letteratura sono noti soltanto risultati teorici (si veda [Vardi, 1989]) per la complessità del ragionamento nelle strutture $\mathcal{MS}$ e il nostro scopo è quello di integrarli con l'osservazione sperimentale. In particolare siamo interessati a capire se certi fenomeni osservati nel caso della logica proposizionale si ripresentano anche nelle logiche da noi considerate. Inoltre, vogliamo evidenziare (se esiste) quale sia la differenza in termini sperimentali nel passaggio dalle logiche il cui problema di soddisfacibilità è NP-complete a quelle per cui lo stesso problema è PSPACE-hard.

La seconda motivazione è che la comparazione di diverse procedure di decisione, siano esse basate su diverse tecnologie, siano esse variazioni sul tema di un'unico decisore, deve necessariamente avvenire su base sperimentale. È infatti molto difficile, per non dire impossssibile, stabilire da un punto di vista analitico l'impatto delle decisioni prese in fase realizzativa sulle prestazioni finali dei decisori. Le informazioni provenienti dall'analisi sperimentale possono inoltre fornire una base per successive realizzazioni più efficienti di uno stesso decisore. In questo senso la costruzione di procedure di decisione diventa un ciclo che alterna una fase puramente realizzativa con una fase di sperimentazione e raccolta dati.

5.2 Metodologia di testing

La metodologia che presentiamo in questa sezione è derivata, con alcune modifiche, da quella proposta in [Giunchiglia *et al.*, 1996] per logiche modali normali e terminologiche. L'architettura concettuale dell'ambiente di testing è descritta in figura 5.1. I risultati che presentiamo in questo capitolo sono stati tutti ottenuti utilizzando questa particolare architettura con cui abbiamo automatizzato il procedimento di test e raccolta dei dati.

L'ambiente descritto in figura 5.1 consta di tre moduli fondamentali: un modulo di *controllo*, un *generatore* e uno o più decisori fra quelli presentati nel capitolo 4. Ogni decisore è preceduto da un modulo di *preprocessing* che "filtra" opportunamente le formule in ingresso. Vediamo in dettaglio la descrizione del comportamento dei moduli:

[**Controllo**] È il supervisore dell'intero processo di testing. Riceve in ingresso un "programma" relativo alla serie di esperimenti da svolgere e, sulla base di questo, carica i decisori necessari, inizializza il modulo di generazione formule e, per ogni formula generata data in ingresso ad un certo decisore, raccoglie i dati corrispondenti e li registra su disco.

[**Generatore**] È un generatore programmabile di formule casuali. Riceve in ingresso dal modulo di controllo i parametri sulla base dei quali

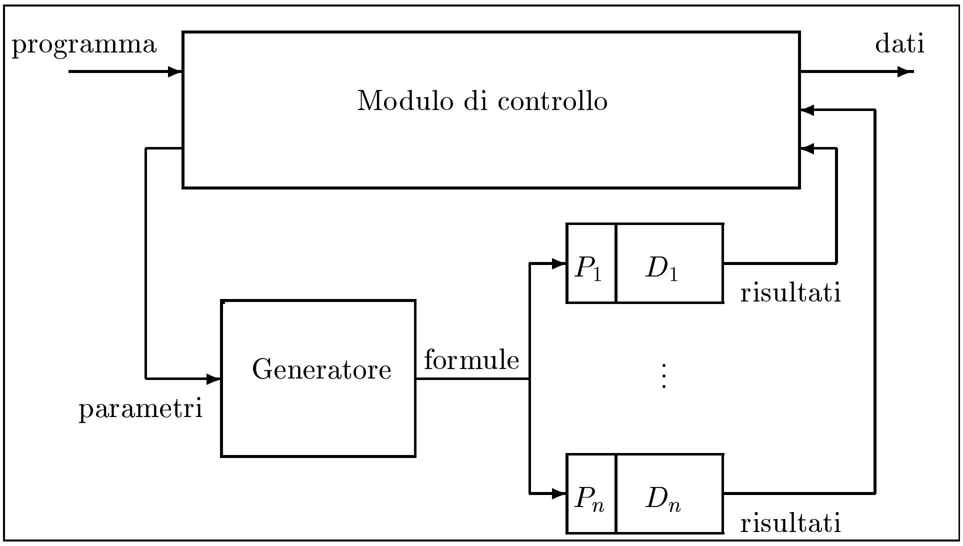

Figura 5.1: Lo schema dell'ambiente di testing ($D_1 \ldots D_n$ sono procedure di decisione, $P_1 \ldots P_n$ sono moduli di preprocessing)

generare le formule che vengono poi smistate ai singoli decisori. In particolare, ogni singolo campione è distribuito a tutte le procedure di decisione, in modo da rendere possibili analisi comparative.

[**Preprocessing**] Si tratta di moduli software che semplificano, ordinano ed eseguono conversioni sintattiche sulle formule in uscita dal generatore.

[**Decisori**] Sono le procedure di decisione, opportunamente modificate per restituire oltre al risultato *True* o *False* altri valori relativi all'efficienza interna quali, ad esempio, numero di chiamate ricorsive, tempi di esecuzione, ecc. Ogni decisore riceve in ingresso una formula campione e terminato il suo compito restituisce il controllo al supervisore.

La costruzione dell'ambiente di testing costituisce una parte importante nella realizzazione di un'approccio sperimentale alla costruzione di decisori efficienti. Lo scopo per cui questo ambiente è stato costruito è quello di eseguire un'analisi sperimentale di tipo *statistico*. Dato il valore di alcuni parametri di controllo e l'eventuale campo di variazione ad essi associato, un certo numero di *campioni* viene generato per ogni punto nel sistema di coordinate stabilito dai parametri variabili. Ad ogni campione corrisponde una formula del linguaggio modale $\mathcal{L}$ e l'insieme delle formule generate con certi parametri rappresenta una particolare *popolazione* rispetto alla quale sono distribuiti i valori in uscita dai decisori. La nostra analisi non è dettagliata rispetto alle singole popolazioni, ma piuttosto ne considera valori di insieme quali, ad esempio, medie e mediane. In particolare, i risultati

formula ::= clause, list-of-clauses
list-of-clauses ::= clause, list-of-clauses | λ
clause ::= literal, list-of literals
list-of-literals ::= literal, list-of-literals | λ
literal ::= atom | $\neg$ atom
atom ::= $\Box$ clause | prop-var
prop-var ::= $P_0, P_1, \ldots P_N$

Figura 5.2: Grammatica delle formule $CNF_{\mathcal{K}}$

sono analizzati comparando la variazione dei valori di insieme rispetto alla variazione dei parametri con cui sono generate le formule.

Per spiegare in dettaglio la metodologia di testing abbiamo deciso di procedere per gradi. Innanzi tutto, la parte 5.2.1 di questa sezione è dedicata alla descrizione della procedura di generazione delle formule e dei parametri che le caratterizzano. La parte 5.2.2 è dedicata alle procedure di "filtro" per i campioni generati, mentre il problema della scelta dei parametri di generazione (fondamentale per la validità dei risultati ottenuti) è discusso in 5.2.3. Infine, le particolari analisi statistiche condotte sui dati in uscita e le motivazioni che ci hanno portato all'utilizzo di certi valori piuttosto che di altri per l'analisi delle nostre procedure sono presentati in 5.2.4.

5.2.1 Generazione di formule

Sino a questo punto, parlando di formule casuali, non siamo entrati nel dettaglio di come sia possibile combinare una generazione probabilistica con il fatto che siamo comunque interessati a controllare certi parametri delle formule generate. Di fatto, come vedremo in 5.2.3 parlando della scelta dei parametri per la generazione, siamo interessati ad avere un controllo ben più sofisticato di quello che una generazione completamente casuale potrebbe fornirci.[1]

Le formule che utilizziamo per la realizzazione degli esperimenti sono molto simili a quelle proposte in [Giunchiglia *et al.*, 1996] la cui struttura può essere formalizzata, restringendoci al caso monoagente, con la grammatica di figura 5.2. Dato che queste formule sono particolarmente adatte per il testing in logiche modali normali e terminologiche, d'ora in avanti ci riferiamo a tali formule con il termine $CNF_{\mathcal{K}}$.

Analizzando la struttura delle formule $CNF_{\mathcal{K}}$, notiamo che ciascuna di esse si presenta come una lista di clausole e ogni clausola è una lista

[1] Per "completamente casuali" intendiamo formule con connettori scelti a caso in numero e posizione.

formula ::= clause, list-of-clauses
list-of-clauses ::= clause, list-of-clauses | λ
clause ::= literal, list-of literals
list-of-literals ::= literal, list-of-literals | λ
literal ::= atom | $\neg$ atom
atom ::= $\Box$ formula | prop-var
prop-var ::= $P_0, P_1, \dots P_N$

Figura 5.3: Grammatica delle formule $CNF_{\mathcal{MS}}$

di letterali. L'interpetazione comunemente assegnata a questa struttura è che le clausole siano tutte in congiunzione tra di loro e i letterali siano tutti in disgiunzione tra di loro. Ogni letterale è un atomo, eventualmente negato, e l'atomo è una sottoformula modale o un atomo proposizionale. Notiamo che le sottoformule modali sono sempre clausole e cioè contengono soltanto disgiunzioni di letterali. In altre parole, l'operatore di congiunzione è presente solo al livello più alto della formula.

Le formule che rispettano la grammatica presentata in figura 5.2 hanno molte interessanti proprietà, tra cui il fatto che esiste una traduzione da una qualsiasi formula del linguaggio modale $\mathcal{L}$ in una corrispondente $CNF_{\mathcal{K}}$ tale da preservarne la soddisfacibilità rispetto alle strutture $\mathcal{K}$ (per i dettagli si veda [Sebastiani, 1997]). Il problema è che questa proprietà non vale per le logiche da noi considerate e cioè non è vero che è possibile tradurre una qualsiasi formula di $\mathcal{L}$ in una corrispondente $CNF_{\mathcal{K}}$. La ragione di questo va ricercata nel fatto che la traduzione proposta in [Sebastiani, 1997] si basa sullo schema $\Box \bigwedge_i \psi_i \leftrightarrow \bigwedge \Box \psi_i$ che sappiamo non essere valido in nessuna delle logiche Λ, eccetto K.

In figura 5.3 presentiamo la grammatica relativa al tipo di formule da noi utilizzato per il testing nelle logiche modali classiche. Chiamiamo le formule in questione $CNF_{\mathcal{MS}}$ per sottolineare il fatto che una qualsiasi formula del linguaggio $\mathcal{L}$ può essere convertita in una equivalente $CNF_{\mathcal{MS}}$ preservando la soddisfacibilità rispetto alle strutture $\mathcal{MS}$ (basta eseguire una conversione CNF per ogni livello modale).

Notiamo che le formule $CNF_{\mathcal{MS}}$ differiscono dalle $CNF_{\mathcal{K}}$ solo per il fatto che all'interno di un atomo modale non vi sono solo clausole ma intere formule. L'impossibilità di "distribuire" la congiuzione rispetto all'operatore epistemico "$\Box$" ci costringe ad utilizzare questa struttura che, a differenza di quanto accadeva con le $CNF_{\mathcal{K}}$, presenta un maggior numero di gradi di libertà. Infatti è necessario gestire opportunamente il numero di congiunzioni presenti in ciascun livello modale oltre a quello più alto.

Parametro	*Descrizione*
R	Numero di letterali contenuti in ogni clausola
T	Probabilità che un letterale sia un atomo negato
B	Probabilità che un atomo sia una variabile proposizionale
N	Numero massimo di variabili proposizionali
d	Massima profondità modale
L_n	Numero di clausole al livello n ($n = 0$ è il livello più alto)

Tabella 5.1: Parametri per la generazione di formule $CNF_{\mathcal{MS}}$

Fissata la struttura per le nostre formule, sono anche determinati i parametri che possono controllare la generazione di serie casuali di campioni. Nella tabella 5.1 è presentato l'elenco di tutti i parametri che utilizziamo a tal fine.

Data la struttura delle formule e i parametri di controllo, il passo successivo è la costruzione di una procedura di generazione che sia programmabile con i parametri esposti nella tabella 5.1. Dato che la struttura delle formule $CNF_{\mathcal{MS}}$ differisce da quella delle formule $CNF_{\mathcal{K}}$ la procedura di generazione descritta in [Giunchiglia *et al.*, 1996] risulta inadatta per i nostri scopi. La realizzazione del generatore di formule è presentata in figura 5.4.

Per generare un campione casuale è sufficiente impostare tutti i parametri e invocare *gen-wff*(0). Il valore iniziale del parametro di accumulazione n è dovuto al fatto che nella tabella 5.1 abbiamo scelto per convenzione il numero 0 per denotare il livello modale più alto. All'interno di *gen-wff* si trova un ciclo che invoca *gen-clause* tante volte quanto è il valore di L_n cioè il numero di clausole per il livello corrente. Stesso comportamento per *gen-clause* la quale invoca per R volte la procedura di generazione per i letterali, *gen-lit*. Quest'ultima restituisce un atomo o, con probabilità T un atomo negato. A sua volta *gen-atom*, dopo aver controllato che il livello modale corrente n sia diverso da quello massimo d, genera con probabilità B un atomo modale o, con probabilità $(1-B)$, un atomo proposizionale. Nel caso in cui $n = d$ non si deve invocare ricorsivamente *gen-wff* per cui il controllo passa comunque a *gen-var*. Quest'ultima estrae un numero da 1 a N sulla base del quale restituisce una certa variabile proposizionale.

La procedura presentata possiede le caratteristiche che ci interessano ai fini della nostra analisi sperimentale. Si tratta di una procedura programma-

Figura 5.4: Generatore di formule $CNF_{\mathcal{MS}}$

---

function *gen-wff*(n)
  for $i := 1$ **to** L_n **do**
    $C_i :=$ *gen-clause*(n);
  return $\bigwedge_i^{L_n} C_i$.

function *gen-clause*(n)
  for $i := 1$ **to** R **do**
    $l_i :=$ *gen-lit*(n);
  return $\bigvee_i^R l_i$.

function *gen-lit*(n)
  $c := \{$a random number in $[0, 1]\}$;
  if $c \leq T$ **then return** $\neg$*gen-atom*(n)
    else return *gen-atom*(n) **.**

function *gen-atom*(n)
  if $n = d$ **then return** *gen-var*;
  $c := \{$a random number in $[0, 1]\}$;
  if $c \leq B$ **then return** $\Box$*gen-wff*$(n + 1)$
    else return *gen-var* **.**

function *gen-var*
  $c := \{$a random number in $[1, N]\}$;
  return P_c.

---

bile, nel senso che variando opportunamente i valori dei parametri è possibile generare formule tra di loro diverse, anche se dotate della medesime struttura. Inotre si tratta di un generatore di campioni casuali, in quanto non è fissato a priori il numero esatto dei letterali negati, degli atomi modali e non si ha un controllo sulle ripetizioni di letterali o sull'utilizzo di essi in punti precisi della formula.

5.2.2 Ordinamento e semplificazione

Le formule generate utilizzando la procedura presentata in figura 5.4 sono tipicamente affette da un elevato "rumore", nel senso che contengono sottoformule ridondanti o ripetizioni inutili di connettori. Questo fatto, unitamente alla natura degli algoritmi di decisione che, in generale, sono molto sensibili a piccole fluttuazioni nelle formule in ingresso, può provocare distorsioni anche considerevoli nei valori finali.

Per eliminare la componente rumorosa delle formule, è sufficiente anteporre ad ogni decisore un semplice modulo di semplificazione come quello presentato in figura 5.5. Notiamo che le operazioni svolte sono, senza alcuna eccezione, molto efficienti e pertanto comportano un dispendio di tempo minimo. In ogni caso, come vediamo in 5.2.4, nelle nostre misure il tempo impiegato per la semplificazione della formula è tenuto distinto dal tempo di decisione vero e proprio.

La procedura di semplificazione *wff-simplify* utilizza un algoritmo di discesa ricorsiva all'interno della struttura delle formule $CNF_{\mathcal{MS}}$. Per ogni formula si gestiscono separatamente i casi in cui essa è una variabile, la negazione di una sottoformula, una congiunzione o un disgiunzione di sottoformule, oppure un atomo modale. Nel caso in cui la formula è una variabile *wff-simplify* ne ritorna semplicemente il valore, mentre in tutti gli altri casi viene invocata un'apposita funzione di semplificazione.

Le sottoformule del tipo $\neg\psi$ sono gestite da *neg-simplify* semplificando ricorsivamente ψ e gestendo tre casi sul valore di ritorno ξ. Se ξ equivale a $\top$ ($\bot$) allora la sua negazione è, ovviamente, $\bot$ ($\top$). Se ξ è ancora una formula negata $\neg\rho$, allora, dato che due negazioni affermano, *neg-simplify* ritorna ρ.

Le congiunzioni e le disgiunzioni vengono gestite, rispettivamente, da *and-simplify* e da *or-simplify*. Entrambe le procedure semplificano ricorsivamente tutti i termini ξ_i, eliminando termini ridondanti, eliminando termini indifferenti e riducendo, quando sia possibile, le congiunzioni a $\bot$ e le disgiunzioni a $\top$. Notiamo che il comportamento delle due procedure è identico a meno della dualità tra gli operatori.

Infine, l'operatore $\Box$ è gestito semplicemente propagando la semplificazione al suo interno e restituendo, in veste di atomo modale, la formula risultante da tale semplificazione. Questa che presentiamo è la versione più generale di *box-simplify* ed è quella valida per tutte le logiche Λ. Per le logiche in cui è valido l'assioma N è possibile utilizzare la versione *box-simplify*$_N$ proposta in figura 5.6.

Il concetto su cui si basa *box-simplify*$_N$ è molto semplice. In tutte le strutture $\mathcal{MS}$ per cui lo schema N risulta valido si ha che $\Box\top \wedge \neg\top$ è assurdo (non ci sono mondi inconsistenti) ed anche $\top \wedge \neg\Box\top$ è assurdo (tutti gli assegnamenti epistemici contengono l'unità per cui $\Box\top \leftrightarrow \top$. Di conseguena è sempre possibile sostituire $\top$ al posto di $\Box\top$.

Anche semplificate, le formule generate casualmente presentano ancora una potenziale sorgente di inefficienza per i nostri decisori. Il problema è connesso con il fatto che agli atomi modali deve essere assegnato un valore di verità e quindi atomi *sintatticamente* uguali devono essere trattati come un'unica variabile proposizionale. È facile vedere che $\Box(\xi \wedge \psi)$ e $\Box(\psi \wedge \xi)$ sono lo stesso atomo modale, ma un test di uguaglianza in tempo lineare (si

Figura 5.5: Semplificazione di formule $CNF_{\mathcal{MS}}$

---

function *wff-simplify*(φ)
  if $\varphi \in \{P_1 \dots P_N\}$ **then return** φ;
  if $\varphi = \neg\psi$ **then return** *neg-simplify*$(\neg\psi)$;
  if $\varphi = \bigwedge_i^L \psi_i$ **then return** *and-simplify*$(\bigwedge_i^L \psi_i)$;
  if $\varphi = \bigvee_i^R \psi_i$ **then return** *or-simplify*$(\bigvee_i^R \psi_i)$;
  if $\varphi = \Box\psi$ **then return** *box-simplify*$(\Box\psi)$; **.**

function *neg-simplify*$(\neg\psi)$
  $\xi :=$ *wff-simplify*(ψ);
  if $\xi = \bot$ **then return** $\top$;
  if $\xi = \top$ **then return** $\bot$;
  if $\xi = \neg\rho$ **then return** ρ **.**

function *and-simplify*$(\bigwedge_i^L \psi_i)$
  for $i := 1$ **to** L **do**
    $\xi_i :=$ *wff-simplify*(ψ_i);
    if $\xi_i := \bot$ **then return** $\bot$;
    if $\xi_i \in \{\xi_1 \dots \xi_{i-1}\}$ **then** $\xi_i := \top$;
  $\Delta = \{\xi_i \mid \xi_i \neq \top\}$;
  return $\bigwedge_{\rho_i \in \Delta} \rho_i$**.**

function *or-simplify*$(\bigwedge_i^R \psi_i)$
  for $i := 1$ **to** R **do**
    $\xi_i :=$ *wff-simplify*(ψ_i);
    if $\xi_i := \top$ **then return** $\top$;
    if $\xi_i \in \{\xi_1 \dots \xi_{i-1}\}$ **then** $\xi_i := \bot$;
  $\Delta = \{\xi_i \mid \xi_i \neq \bot\}$;
  return $\bigvee_{\rho_i \in \Delta} \rho_i$**.**

function *box-simplify*$(\Box\psi)$
  $\xi :=$ *wff-simplify*(ψ);
  return $\Box\xi$**.**

---

pensi alla primitiva `EQUAL` del LISP) non sarà mai in grado di verificarlo. Più in generale, il nostro problema è costituito dal fatto che formule costituite da diverse permutazioni di identiche sottoformule, sfuggono ad un generico test di uguaglianza lineare, dando origine ad una proliferazione nel numero delle variabili. In pratica, all'aumento di $\mid Atom(\varphi) \mid$ si aggiunge il fatto che

Figura 5.6: Semplificazione di atomi modali (per le logiche in cui è valido l'assioma N)

---

function *box-simplify*$_N(\Box\psi)$
  $\xi :=$ *wff-simplify*(ψ);
  if $\xi = \top$ **then return** $\top$;
  return $\Box\xi$**.**

---

i possibili assegnamenti per φ aumentano come $2^{|Atom(\varphi)|}$ determinando un drastico peggioramento nelle prestazioni dei decisori.

Il modo più naturale per trattare questa anomalia dovuta al fatto che si gestiscono formule modali in maniera "atomica", è quello di effettuare una passata di ordinamento sulle formule in ingresso ai decisori. Il particolare tipo di ordine utilizzato non è di fondamentale importanza, purché sia tale da ricondurre tutte le permutazioni di una stessa formula ad un'unica forma. Nel nostro caso si tratta di una modifica del comune ordine lessicografico per adattarlo alla struttura delle formule generate.

Come avevamo anticipato nella sezione 4.5.3, questo tipo di procedimento deve essere considerato una parte del decisore vero e proprio, piuttosto che una forma di preprocessing come la semplificazione. Il motivo per cui abbiamo introdotto la semplificazione è dipendente dal particolare generatore utilizzato mentre il problema delle diverse permutazioni di formule uguali è un aspetto intrinseco alle procedure di decisione da noi realizzate e quindi il problema si ripresenterebbe anche con formule non generate probabilisticamente.

D'altro canto, visto che dal punto di vista realizzativo le procedure di ordinamento sono staccate dai decisori veri e propri, abbiamo preferito trattarle insieme alle altre forme di preprocessing. Anche dal punto di vista della valutazione delle prestazioni, nella sezione 5.2.4 vediamo che il tempo per l'ordinamento e la semplificazione sono contati assieme come tempo di preprocessing. Questo è dovuto anche al fatto che, non avendo necessità di compararci con altre procedure di decisione se non KSAT che sfrutta la stessa tecnologia, il tempo di ordinamento viene spesso trascurato nelle nostre analisi sperimentali. Nel caso generale, considernado che i tempi dell'ordinamento crescono al più in maniera proporzionale a $|\varphi|^2$, mentre i nostri decisori hanno complessità decisamente superiori, non si introducono distorsioni sostanziali considerando l'ordinamento come un preprocessing.

5.2.3 Scelta dei parametri

Ora che abbiamo definito gli aspetti relativi alla generazione e al preprocessing delle formule si pone il problema della scelta dei parametri in base ai quali osservare gli andamenti dei valori forniti in uscita dai decisori. Tutti i parametri messi a disposizione dal generatore (vedi tabella 5.1) definiscono uno spazio multidimensionale all'interno del quale si collocano le formule generate. Dato che l'esplorazione esaustiva di tutto lo spazio non è concepibile e probabilmente anche inutile, ci chiediamo quale possa essere il sottospazio di interesse, cioè quell'insieme di formule su cui compiere la nostra indagine.

Così come abbiamo fatto per la generazione delle formule, ispireremo il nostro modello per l'analisi dei dati a quello proposto in [Giunchiglia *et al.*, 1996] che, d'ora in avanti, chiamiamo "$3CNF_{\mathcal{K}}$". Nel seguito descriviamo tale metodologia e spieghiamo come sia possibile adattarla al testing con logiche non normali.

Innanzi tutto, il metodo "$3CNF_{\mathcal{K}}$"prevede che il numero di letterali in ogni clausola sia fisso e noto a priori. Al riguardo, nell'area di ricerca su SAT, si parla di modello *fixed clause length* (si veda [Mitchell *et al.*, 1992]) proprio perché il valore del parametro R non è fissato su base probabilistica (questo è un'altro modello detto *constant probability*). Il valore scelto per R è, sempre in analogia con il caso proposizionale, $R = 3$ e quindi le formule generate sono $CNF_{\mathcal{K}}$ (vedi la grammatica in figura 5.2) aventi solo tre letterali in ogni clausola. Il valore attribuito al parametro R ha un significato ben preciso in quanto le formule con tre letterali per clausola sono le più semplici per cui il problema della soddisfacibilità è ancora NP-complete (si veda [Garey e Johnson, 1979] al riguardo).

Per quanto riguarda i parametri probabilistici, il metodo "$3CNF_{\mathcal{K}}$"prevede che a B e a T, cioè alle probabilità di generare atomi modali e letterali negati, venga assegnato un valore costante. In particolare, in [Giunchiglia e Sebastiani, 1996a] e in [Giunchiglia e Sebastiani, 1996b] l'analisi comparativa tra KSAT e altri sistemi viene condotta con $B = 0.5$ e $T = 0.5$. In altre parole, per ogni livello modale, è assolutamente equiprobabile la generazione di atomi modali o letterali negati, rispetto alla generazione di atomi proposizionali e letterali positivi. Le ragioni per questa scelta sono, nel caso di T, l'analogia con il caso proposizionale e, nel caso di B, il fatto che è desiderabile ottenere, in media, un bilanciamentto tra il ragionamento proposizionale e quello modale.

Il metodo "$3CNF_{\mathcal{K}}$", per valori di d e N arbitrari ma fissati a priori, prevede la generazione di un numero sufficiente di campioni per ogni valore di L che si muove in un opportuno campo di variazione. In analogia con il caso proposizionale, tale campo viene scelto sperimentalmente in maniera tale che la percentuale delle formule soddisfacibili per ogni punto decresca dal 100% alllo 0%. Nella figura 5.7 presentiamo un esempio relativo al semplice caso proposizionale, ottenuto con il nostro generatore impostando $d = 0$. Si

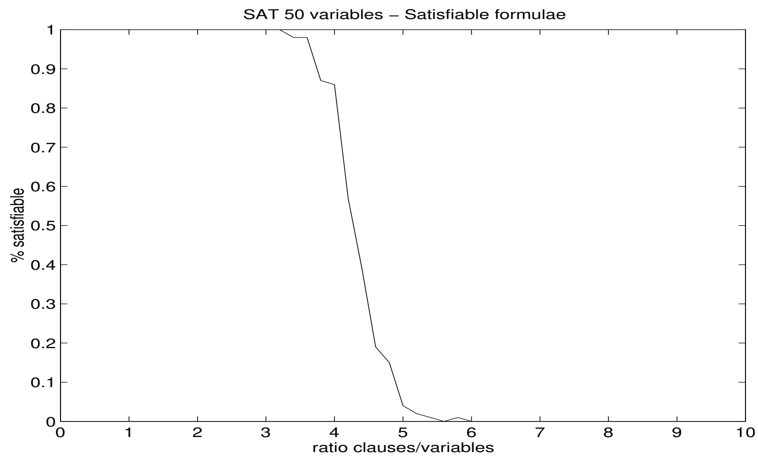

Figura 5.7: Percentuale di formule soddisfacibili per un test SAT ($N = 50$)

noti che l'asse delle ascisse riporta il numero delle clausole L normalizzato sul numero delle variabili N così come prevede il metodo "$3CNF_{\mathcal{K}}$". La ragione per cui si esegue una tale normalizzazione è che il rapporto L/N, almeno nel caso proposizionale, è una misura di quanto una formula è *vincolata*. Infatti, ad un basso valore del rapporto corrispondono formule con poche clausole rispetto al numero delle variabili e quindi formule facilmente soddisfacibili (poco vincolate). Viceversa, all'aumentare del rapporto, otteniamo formule con molte clausole rispetto al numero di variabili e quindi facilmente insoddisfacibili (molto vincolate).

Il metodo "$3CNF_{\mathcal{K}}$"sino a qui descritto presenta tre interessanti proprietà che siamo interessati a conservare:

1. data una qualsiasi logica i cui modelli sono un sottoclasse di $\mathcal{K}$, le formule $3CNF_{\mathcal{K}}$ rappresentano tutte le formule in $\mathcal{L}$ dato che, come abbiamo già fatto notare, esiste una conversione tale da preservare la soddisfacibilità nelle strutture $\mathcal{K}$;

2. l'uso della forma $3CNF_{\mathcal{K}}$ minimizza il numero di parametri da gestire come campo di valori, piuttosto che come valore fissato (in effetti solo L è variabile);

3. come nel caso proposizionale, l'aumento del rapporto L/N corrisponde ad un incremento *monotono* della misura di quanto la formula è vincolata (*constraintness* della formula) ed è tale da consentire una regolazione approssimativa della difficoltà delle formule generate.

Presentiamo adesso la scelta dei parametri di generazione nel nostro caso. Quando diciamo proprietà 1, 2, ecc. intendiamo riferirci alle proprietà appena elencate.

Innanzi tutto, come abbiamo fatto notare in 5.2.1, al fine di conservare la proprietà 1 è necessario modificare la struttura delle formule introducendo la congiunzione di clausole anche ai livelli modali diversi da quello più alto. D'altronde, come è lecito supporre, questa scelta obbligata mette in serio pericolo la conservazione delle proprietà 2 3. È chiaro che, se anche la scelta di fissare i parametri R, T, B, N e d fosse ancora possibile, in ogni caso avere L_n piuttosto che L aggiunge una "coordinata" per ogni livello modale. Quindi se con "$3CNF_{\mathcal{K}}$" dobbiamo gestire sempre e solo la variazione del numero di congiunzioni al livello più alto, nell'elaborare un metodo di test per le logiche non normali dobbiamo tenere conto del fatto che è data una molteplicità di livelli per cui gestire la variazione del numero delle clausole. Questo implica l'impossibilità di ottenere la proprietà 2 a meno di non rinunciare alla 1.

Un'altra difficoltà, anche se per ragioni meno evidenti, è data dalla proprietà 3. Si consideri, ad esempio, il caso in cui le formule vengano generate aumentando il valore di L_0 e contemporaneamente il valore di L_n per tutti gli $n > 0$. Supponiamo anche che tutti gli altri parametri vengano scelti come nel metodo "$3CNF_{\mathcal{K}}$" e in particolare, che sia fissato il numero di variabili e la massima profondità modale. Consideriamo allora un possibile assegnamento μ generato da ΛSAT dato da $\mu = \Box\alpha \wedge \neg\Box\beta$. La struttura di μ è scelta in modo che per qualsiasi logica Λ si renda necessario un test di consistenza ricorsivo. In particolare in quasi tutti gli algoritmi che abbiamo visto, eccezion fatta per qualche caso in EC, i test ricorsivo è sempre nella forma $\alpha \wedge \neg\beta$. Noi sappiamo che la formula β ha una struttura ben precisa e cioè:

$$\beta = \bigwedge_{i}^{L_n} \bigvee_{j}^{R} \beta_{ij} \tag{5.1}$$

e quindi siamo in grado di ricavare la negazione di β:

$$\beta = \bigvee_{i}^{L_n} \bigwedge_{j}^{R} \neg\beta_{ij} \tag{5.2}$$

per cui il test ricorsivo diventa:

$$\Lambda\text{SAT}(\bigwedge_{i}^{L_n} \bigvee_{j}^{R} \alpha_{ij} \wedge \bigvee_{i}^{L_n} \bigwedge_{j}^{R} \neg\beta_{ij}) \tag{5.3}$$

Osserviamo che per valori di $L_n > R$ la formula corrispondente a $\neg\beta$ è meno vincolata di quella corrispondente ad α in quanto è più facile soddisfare una disgiunzione rispetto ad una congiunzione. L'aumento di L_n non fa che aumentare il divario: α diventa sempre più banalmente insoddisfacibile e il contrario per β che diventa sempre meno vincolata e quindi sempre più facilmente soddisfacibile. Questo esempio mostra come, in generale, una variazione del parametro L_n per $n > 0$ non ci permette di essere sicuri *a priori* del fatto che stiamo modificando in maniera monotona la constraintness della formula. Per di più, date particolari scelte dei parametri B e T (valori prossimi a 1), o particolari relazioni tra il parametro L_0 e gli L_n per $n > 0$ è addirittura possibile, in certe logiche, che all'aumento di L/N possa corrispondere una diminuzione, in media, del livello di constraintness delle formule generate. In altre parole, la proprietà 3 potrebbe non essere più verificata per tutte le scelte dei parametri di generazione.

La conclusione della discussione precedente è che, in tutte le logiche Λ (eccetto R e K), non è possibile effettuare una scelta di parametri tali da costruire un metodo che abbia tutte le proprietà del metodo "$3CNF_{\mathcal{K}}$". Partendo da questa considerazione cerchiamo di capire quali sono i compromessi che possiamo accettare senza invalidare l'analisi sperimentale. In realtà quindi, più che un metodo, definiamo una famiglia di metodi che chiameremo in generale "$3CNF_{\mathcal{MS}}$", di cui il metodo "$3CNF_{\mathcal{K}}$" è un caso particolare.

Cominciamo dal valore per R che decidiamo di mantenere costante e, come nel caso del metodo "$3CNF_{\mathcal{K}}$", pari a $R = 3$. Questo ci consente di mantenere le formule il più compatte possibile senza perdere l'intrinseca "durezza" dei problemi della soddisfacibilità nelle logiche Λ, sia per quelle che hanno un test NP-complete, sia per quelle che presentano un test PSPACE-hard. Il valore per le probabilità B e T è fissato e, in generale, usiamo i valori $B = 0.5$ e $T = 0.5$.

Nella nostra terminologia chiamiamo *problema* un esperimento condotto avendo fissato un valore per d e per N. In figura 5.8 vediamo il grafico relativo ai valori di soddisfacibilità percentuali per formule modali nella logica E con $d = 1$ e $N = 3$. Notiamo che il grafico è tridimensionale in quanto, fissata $d = 1$, possiamo decidere di variare indipendentemente il valore di L_1 e L_0. La rappresentazione dei valori avviene sempre normalizzando il numero delle clausole per ogni livello rispetto al numero delle variabili. Ovviamente, se $d > 1$ perdiamo la rappresentazione grafica relativamente a tutti i valori restituiti dai decisori e questo, in generale, ci costringe a restringere ulteriormente lo spazio delle formule generate e analizzate.

Nell'eseguire gli esperimenti che presentiamo nella sezione 5.3, abbiamo utilizzato principalmente le seguenti metodologie:

"$3CNF^1_{\mathcal{MS}}$" Tutti i valori sono fissati come nell'esempio di figura 5.8. In particolare, essendo questo approccio fattibile al massimo per $d = 1$, si varia indipendentemente il numero delle clausole di livello 0 (L_0) e il numero delle clausole di livello 1 (L_1) detti anche *top level* e *bottom level*. Per ogni coordinata (L_0, L_1) si genera un certo numero di campioni il cui valore di insieme può essere rappresentato in un grafico tridimensionale con assi normalizzati $(L_0/N, L_1/N)$.

"$3CNF^n_{\mathcal{MS}}$" Un modo per ottenere rappresentazioni significative anche per

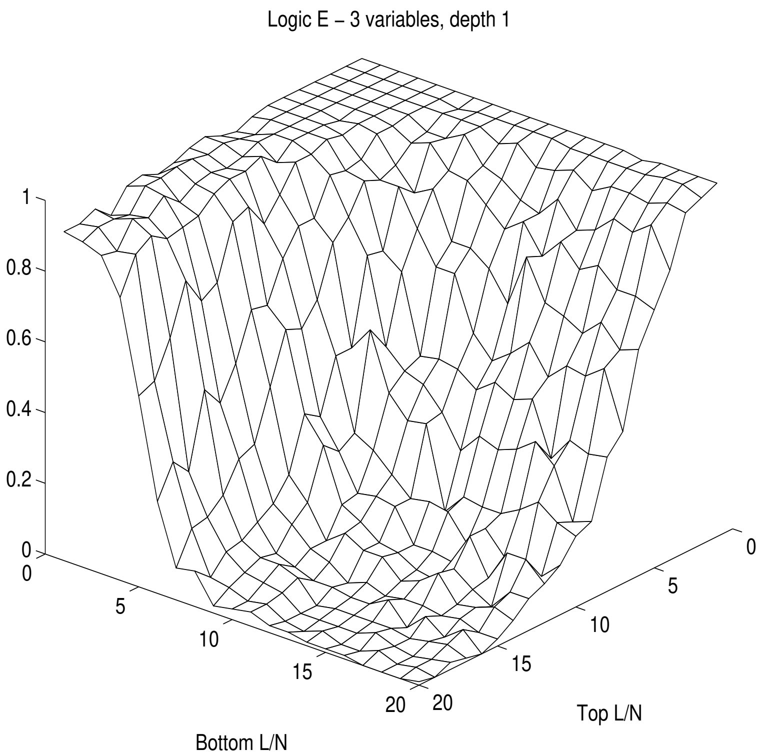

Figura 5.8: Percentuale di formule soddisfacibili per la logica E con $N = 3$ $d = 1$. Sugli assi sono indicati i rapporti L_0/N (Top Level L/N) e L_1/N (Bottom Level L/N). Gli altri parametri di generazione sono $R = 3$, $B = 0.5$, $T = 0.5$, 100 campioni per punto.

$d > 1$ è quello di agganciare la variazione del numero di clausole dei livelli più bassi alla variazione di L_0. Formalmente si tratta di realizzare la regola di generazione $\forall n.L_n = L_0$. In sostanza, considerato lo spazio multidimensionale definito da $L_0 \ldots L_d$, attraversiamo "in diagonale" l'insieme delle formule possibili.

"$3CNF^k_{\mathcal{MS}}$" un altro modo per mantenere la rappresentazione grafica per $d > 1$ è quello di fissare, indipendentemente da L_0, il valore di L_n per tutti gli $n > 0$. Notiamo che, se il valore scelto è $k = 1$, allora il metodo coincide con "$3CNF_{\mathcal{K}}$". I test che eseguiamo con questo metodo sono rappresentati su di un asse L/N dove $L = L_0$ e pertanto risultano parametrici nella dimensione di k.

In assenza di una metodologia valida in generale, nel corso della nostra indagine sperimentale abbiamo dovuto di volta in volta porci il problema di quale fosse la metodologia più appropriata per il particolare test che intendevamo eseguire.

In particolare "$3CNF^1_{\mathcal{MS}}$" si è dimostrata un supporto molto valido per capire l'andamento globale di certi valori, come ad esempio la percentuale di formule soddisfacibili. La limitazione più forte di questa tecnica è l'impossibilità di scalare a profondità maggiori di uno: l'assenza di tecniche analitiche per la valutazione dei risultati in più dimensioni che siano dimostrate estendere il caso monodimensionale impedisce di fatto l'analisi dei dati raccolti.

La metodologia "$3CNF^n_{\mathcal{MS}}$" è sicuramente un ottimo banco di prova per l'efficienza dei decisori e a differenza di "$3CNF^1_{\mathcal{MS}}$" può essere estesa a profondità modali superiori a uno senza perdere la rappresentazione grafica. Peraltro, come abbiamo discusso in precedenza, qeusta metodologia non è assente da difetti. Il difetto più grave è che per molte delle logiche Λ non siamo garantiti che un incremento di L/N propagato a tutti i livelli incrementi monotonicamente la constraintness della formula. A questo si aggiunge il fatto che le dimensioni delle formule generate "esplodono" molto velocemente con l'aumentare di L e della profondità d. Abbiamo comunque utilizzato "$3CNF^n_{\mathcal{MS}}$" per alcuni test sulla logica E, dove era particolarmente interessante capire le prestazioni del decisore su formule particolarmente "dure".

La metodologia "$3CNF^k_{\mathcal{MS}}$" è quella che abbiamo utilizzato per la maggior parte dei test. Pur non coprendo esaustivamente tutti i valori per L, le formule generate con $L_n = k$ per $n > 0$ ci garantiscono che al crescere del rapporto L/N il grado di constraintness della formula aumenta essendo fortemente dipendente dal numero di clausole generate al livello più alto. Questo ci ha permesso di verificare le prestazioni dei nostri decisori seguento il pradigma generale proposto in SAT (si veda [Mitchell *et al.*, 1992]) e, per analogia, in K (si veda [Giunchiglia *et al.*, 1996]).

Per apprezzare la differenza tra la metodologia "$3CNF^k_{\mathcal{MS}}$" e la "$3CNF^n_{\mathcal{MS}}$" consideriamo un calcolo che mette in luce la differenza nella crescita delle formule generate all'aumentare del numero di clausole. Chiamiamo $Prop_n$ il numero di atomi presenti in una formula generata con il metodo "$3CNF^n_{\mathcal{MS}}$" e $Prop_k$ la stessa quantità ottenuta con il metodo "$3CNF^k_{\mathcal{MS}}$". Ponendo $X(i) = R^{i+1}B^i(1-B)$ risulta:

$$Prop_n(L) = \sum_{i=0}^{d} k^i L X(i) \tag{5.4}$$

$$Prop_k(L) = \sum_{i=0}^{d} L^{i+1} X(i) \tag{5.5}$$

Ci chiediamo ora quale sia l'incremento delle due quantità se il numero delle clausole aumenta di uno. Formalmente vogliamo confrontare il valore delle differenze:

$$\begin{aligned} &Prop_k(L+1) - Prop_k(L) = \\ &= \textstyle\sum_{i=0}^{d} k^i (L+1) X(i) - \sum_{i=0}^{d} k^i L X(i) = \sum_{i=0}^{d} k^i X(i) \end{aligned} \tag{5.6}$$

$$\begin{aligned} &Prop_n(L+1) - Prop_n(L) = \\ &= \textstyle\sum_{i=0}^{d} k^i (L+1)^{i+1} X(i) - \sum_{i=0}^{d} k^i L^{i+1} X(i) \geq \sum_{i=0}^{d} L^i X(i) \end{aligned} \tag{5.7}$$

Si noti che, per semplicità, abbiamo introdotto una minorazione della differenza $Prop_n(L+1) - Prop_n(L)$ piuttosto che il valore esatto. Quanto abbiamo ricavato ci è comunque sufficiente per concludere che per $L > k$ l'espansione di una formula generata in accordo con il metodo "$3CNF^n_{\mathcal{MS}}$" supera l'espansione della corrispondente formula generata seguendo il metodo "$3CNF^k_{\mathcal{MS}}$". Formalmente:

$$\Delta Prop_n(L) \geq \sum_{i=0}^{d} L^i X(i) > \sum_{i=0}^{d} k^i X(i) = \Delta Prop_k(L) \tag{5.8}$$

Ora è anche evidente il motivo per cui abbiamo affermato che le formule generate con il metodo "$3CNF^n_{\mathcal{MS}}$" diventano molto più dure all'aumentare di L. Dal risultato 5.8 notiamo che l'espansione delle formule "$3CNF^n_{\mathcal{MS}}$", passando da L a $L+1$ clausole, dipende da L e quindi maggiore è L maggiore è l'espansione. In pratica questo si traduce in un aumento esponenziale delle dimensioni dela formula a fronte di quello lineare ottenuto con il metodo "$3CNF^k_{\mathcal{MS}}$". Concludiamo osservando che, anche in quest'ultimo caso, è necessario che k sia abbastanza piccolo affinché le formule generate non abbiano dimensioni proibitive.

5.2.4 Valutazione delle prestazioni

In qusta sezione vogliamo entrare nei dettagli del problema della scelta dei valori in base ai quali valutare le prestazioni dei decisori. Nella tabella 5.2

Gruppo	*Valori in uscita*
Memoizing	Numero di ricerche (divise fra totali e con successo)
	Spazio allocato e spazio effettivamente utilizzato
	Numero di risultati positivi e di risultati negativi
Parte SAT	Soddisfacibilità della formula
	Chiamate totali alla procedura ΛSAT_{DP}
	Numero di backtracking effettuati
	Numero di letterali assegnati (detto anche numero di *tries*)
Parte Modale	Chiamate ricorsive (numero di volte per cui viene invocato il test $\Lambda\text{SAT}(\Gamma_\alpha \circ \Gamma_\beta)$)
	Numero di assegnamenti banalmente consistenti
	Numero di early pruning (divisi fra totali e con successo)
Tempi di CPU	Tempo totale(incluso eventuale preprocessing delle formule)
	Tempo di esecuzione
	Tempo di esecuzione (escluso il tempo per il labeling degli atomi modali)

Tabella 5.2: Valori in uscita dai decisori

presentiamo, divisi per gruppi, un insieme di valori che possono essere restituiti in uscita dai decisori. Nella discussione che segue presentiamo le finalità per cui particolari valori sono utili al fine di confrontare diverse procedure di decisione. Si noti che con "diverse" intendiamo procedure di decisione che differiscono anche per una singola caratteristica realizzativa (per esempio ECsat con o senza memoizing). In questo caso il confronto serve per confermare sperimentalmente l'ipotesi che una certa modifica aumenta le prestazioni del decisore o, viceversa, per ricavare l'evidenza del contrario.

Vediamo, per ogni gruppo di valori presentato nella tabella 5.2, quali sono le motivazioni che rendono rilevanti tali quantità e in che contesto.

Il primo gruppo raccoglie un insieme di valori che, nel loro complesso, ci consentono di valutare l'efficienza delle delle strutture di memoizing e delle primitive che ne disciplinano l'accesso (introdotte nella sezione 4.5).

Avendo a disposizione sia il numero totale delle ricerche, cioè il numero totale di volte in cui utilizziamo la primitiva Λ-*get*, sia il numero di ricerche che hanno successo, cioè il numero di volte in cui Λ-*get* ritorna un risultato diverso da $\perp$, siamo in grado di calcolare la *hit ratio* della struttura di memoizing, definita come:

$$\text{Hit ratio} \equiv \frac{\text{Ricerche con esito}}{\text{Ricerche totali}} \tag{5.9}$$

Una hit ratio alta indica una scelta ottimale delle primitive che realizzano il memoizing. In generale siamo interessati sia ai valori medi, sia ai valori estremi per questo parametro, in quanto i primi ci danno una visione influenzata dal fatto che certi campioni (formule) potrebbero essere banalmente soddisfacibili o insoddisfacibili, tanto da non richiedere nessun utilizzo delle primitive di accesso. Viceversa, i valori estremi consentono di ricavare un'efficienza media in caso di utilizzo.

Sempre del gruppo relativo al memozing, fanno parte i valori relativi alla quantità di spazio allocato e alla quantità di spazio usato. Precisiamo che la distinzione tra questi valori ha senso per i decisori come ESAT, EMSAT e, in generale, per tutti quelli che allocano staticamente una matrice per poi eventualmente riempirla con i risultati computati. Per i decisori come ECSAT e simili, che allocano lo spazio *on demand*, l'unico valore che ha senso è quello relativo allo spazio utilizzato. Per il primo gruppo di procedure di decisione diventa interessante il calcolo dell'efficienza in termini di spazio della struttura di memoizing, cioè il rapporto tra lo spazio allocato e lo spazio utilizzato. Formalmente:

$$\text{Efficienza} \equiv \frac{\text{Spazio utilizzato}}{\text{Spazio allocato staticamente}} \tag{5.10}$$

Questo valore ci serve per capire se la particolare realizzazione della struttura di memoizing è tale da sprecare spazio o meno. La conoscenza dello spazio utilizzato è fondamentale per l'analisi delle prestazioni di quei decisori che, per la particolare complessità del problema che affrontano, possono utilizzare una quantità esponenziale di memoria.

Gli ultimi valori relativi al memoizing sono il numero di risultati positivi e il numero di risultati negativi memorizzati. Come abbiamo visto nella sezione 4.5, le primitive Λ-*get* e Λ-*store* sono costruite per memorizzare e recuperare risultati in termini di validità. I valori relativi al tipo di risultato memorizzato possono aiutarci a capire se questa sia , in generale, una scelta efficiente, oppure se la sua efficacia dipenda dalla particolare logica, da qualche particolare condizione sui parametri, ecc.

Passiamo ora alla descrizione del gruppo di valori relativo alla parte proposizionale dei decisori. In questo gruppo abbiamo anche incluso il risultato relativo alla soddisfacibilità della formula modale data in ingresso alla procedura di decisione in quanto, da un punto di vista realizzativo, è la parte proposizionale che amministra tale risultato.

Con numero di chiamate a ΛSAT_{DP}, intendiamo il numero totale di volte in cui viene invocata la parte puramente proposizionale dei decisori, sia per eseguire un test ricorsivo necessario per valutare la consistenza di un assegnamento, sia quando la procedura si invoca nel tentativo di generare un assegnamento per una generica sottoformula. Questo valore è quindi importante perché ci dà un'idea delle dimensioni totali dello spazio di ricerca, inteso sia dal punto di vista proposizionale, sia dal punto di vista modale.

Il numero di backtracking è il numero totale di volte in cui ΛSAT ritorna *False*. Un elevato valore di backtracking suggerisce che la ricerca di un assegnamento per la formula sia risultata particolarmente ostica. Se, a parità di altri fattori, un'euristica si dimostra capace di ridurre il backtracking, significa che le scelte compiute in base ad essa sono in qualche modo più efficaci e richiedono meno "ripensamenti" per correggere scelte iniziali sbagliate.

Infine, il numero di *tries* rappresenta il numero totale di volte in cui si è tentato di costruire un assegnamento attribuendo un valore ad un letterale. In particolare, calcoliamo il numero di tries distinto nella quantità di letterali assegnati in seguito a unit propagation e nella quantità di letterali assegnati in seguito a splitting. La somma dei due valori cositituisce una stima dello spazio di ricerca puramente proposizionale, mentre il confronto tra numero di unit propagation e numero di splitting ci dà una stima della difficoltà (sempre in termini proposizionali) della formula da decidere.

Per quanto riguarda la parte modale, il valore di gran lunga più importante è rappresentato dal numero di chiamate ricorsive effettuate al decisore. Come abbiamo visto nelle sezioni 4.3 e 4.5, il controllo di consistenza di un assegnamento avviene invocando una serie di test ricorsivi su particolari sottoformule dell'assegnamento. Il numero di volte per cui questo avviene costituisce una misura dello spazio di ricerca depurato della componente proposizionale. Infatti, se l'assegnamento è puramente proposizionale nessuno dei decisori esegue alcuna chiamata a ΛSAT e quindi se le formule in ingresso non contengono operatori modali, il numero di chiamate ricorsive è comunque nullo. In generale, in presenza di formule modali, il numero di chiamate ricorsive è una misura empirica della complessità del ragionamento modale. Il grande pregio di questo valore parametro, come di tutti quelli che involvono un mero conteggio, è quello di essere indipendente dalla piattaforma (hardware, sistema operativo, linguaggio) su cui è calcolato.

Il numero di assegnamenti banalmente consistenti è il numero di volte in cui, per assenza di parte modale, per mancanza di atomi assegnati a falso o per mancanza di atomi assegnati a vero (solo per le logiche che non hanno nei loro teoremi lo schema N), un assegnamento può essere dichiarato consistente senza alcuna chiamata ricorsiva. Se questo numero è troppo elevato rispetto alle chiamate ricorsive, significa che le formule date in ingresso non consentono una analisi corretta dei decisori, in quanto stiamo sfruttando poco la parte puramente modale.

Il test su assegnamenti incompleti che avevamo chiamato early pruning nella sezione 4.2, è parte di tutti i nostri decisori. Pertanto, siamo interessati a conoscere il numero totale di test intermedi, e, tra questi, il numero di volte in cui riusciamo ad effettuare backtracking e cioè il numero di volte in cui gli assegnamenti incompleti risultano anche modalmente inconsistenti. Capire per quali scelte dei parametri di generazione delle formule otteniamo una elevata efficacia dell'early pruning può consentirci di elaborare un decisione euristica Λ-*likely-unsatisfiable* più sofisticata di quella che attualmente è realizzata nei nostri decisori che prevede sempre l'invocazione del controllo su assegnamenti incompleti.

I tempi di CPU sono considerati il parametro che meglio sintetizza le prestazioni di un qualsiasi sistema. Questo è particolarmente vero quando diversi sistemi sono comparati "a scatola chiusa", cioè senza preoccuparsi dei dettagli realizzativi ma soltanto della loro prestazione complessiva. Ovviamente il principale difetto di queste misure è quello di essere dipendenti da tutta una serie di fattori che vanno dal particolare compilatore utilizzato sino al particolare hardware su cui sono stati eseguiti i test. Nella nostra analisi utilizzeremo i tempi di CPU soprattutto come riscontro e per convalidare i risultati ottenuti con altre misure. In particolare, teniamo conto tre diversi tempi e cioè il tempo totale di esecuzione (compreso eventuale preprocessing delle formule) il tempo totale per decidere una formula e lo stesso tempo depurato della componente relativa all' etichettatura degli atomi modali. La differenza tra questi tempi ci permette di capire se le procedure che implementano il preprocessing e il labeling si mantengono efficienti all'aumentare della dimensione delle formule. Invece, comparando le procedure di decisione da noi costruite, possiamo omettere i tempi di preprocessing e labeling per concentrarci sull'efficienza del nocciolo costituito dai particolari algorimti di decisione.

5.3 Presentazione e analisi dei risultati

Lo scopo di questa sezione è quello di presentare e discutere i risultati sperimentali che abbiamo ottenuto applicando i possibili metodi della famiglia "$3CNF_{\mathcal{MS}}$". Nella parte 5.3.1 ci concentriamo sui risultati ottenuti per la logica E e, in particolare, sulle questioni riguardanti l'utilizzo del memoizing. Nella parte 5.3.2 presentiamo il confronto sperimentale tra la procedura EMCNSAT e la procedura KSAT descritta in [Sebastiani, 1997] e i risultati ottenuti confrontando il decisore per E con il migliore fra i due decisori per K.

Tutti gli esperimenti sono stati condotti su un `Pentium 133Mhz/64Mb` e un `Pentium 100Mhz/40Mb` entrambi equipaggiati con il sistema operativo

`Linux`; i decisori e l'ambiente di testing sono stati scritti in LISP e compilati utilizzando `GCL 2.2`. Tutte le elaborazioni statistiche sui dati in uscita dai decisori e i grafici degli esperimenti sono stati ottenuti con `MATLAB 4.0`.

5.3.1 La logica E

Come abbiamo fatto presente nella sezione 1.2 non sono note in letteratura procedure di decisione con cui confrontare la procedura ESAT. Nonostante questo, è stata evidente sino dalle prime fasi realizzative la necessità di sapere se la procedura che stavamo costruendo era più efficiente di qualsiasi altro possibile approccio noto anche solo in teoria.

Un primo banco di prova per la procedura ESAT è stato il confronto con la procedura di decisione ottenuta utilizzando il decisore KSAT e la traduzione da E a $K(2)$[2] definita in [Gasquet e Herzig, 1996]. Come abbiamo detto in 1.2 questo è l'unico modo di realizzare una procedura di decisione per E con i soli risultati noti in letteratura. In particolare, la traduzione t è definita come:

- se φ è una variabile, $t(\varphi) = \varphi$
- se $\varphi = \neg\psi$, $t(\varphi) = \neg t(\psi)$
- se $\varphi = \psi \wedge \xi$, $t(\varphi) = t(\psi) \wedge t(\xi)$
- se $\varphi = \Box\psi$, $t(\varphi) = \Diamond_1(\Box_2 t(\psi) \wedge \Box_1 \neg t(\psi))$

Sempre in [Gasquet e Herzig, 1996] viene presentata una dimostrazione del fatto che $\vdash_E \varphi \iff \vdash_{K(2)} t(\varphi)$. Questo risultato ci permette di costruire un decisore per E semplicemente aggiungendo a KSAT un modulo di preprocessing che, utilizzando t, converta le formule generate casualmente.

Il risultati che presentiamo sono stati ottenuti con la metodologia "$3CNF^n_{\mathcal{MS}}$" e cioè variando di pari passo il numero di clausole per ogni livello modale. Siamo garantiti che per la scelta dei parametri effettuata la constraintness delle formule cresce monotonicamente al crescere del rapporto L/N. Questo si può vedere facilmente osservando il grafico di figura 5.8 dal quale è chiaro che l'aumento contemporaneo di L_0/N e L_1/N diminuisce la percentuale di formule soddisfacibili. Naturalmente è sempre vero che la dimensione delle formule cresce esponenzialmente al crescere del numero di clausole e, infatti, questa prima serie di test è limitata ad un rapporto $L/N = 10$.

In figura 5.9 e in figura 5.10 presentiamo, rispettivamente, il confronto tra chiamate ricorsive e il confronto tra tempi di CPU per il problema $N = 4$ $d = 1$ risolto dalla nostra procedura di decisione (con memoizing) e da quella basata su KSAT (KSAT^t d'ora in poi). I valori degli altri parametri utilizzati per la generazione delle formule sono indicati nelle didascalie. In particolare,

[2]In generale, con $K(m)$ si intende la logica K con m agenti

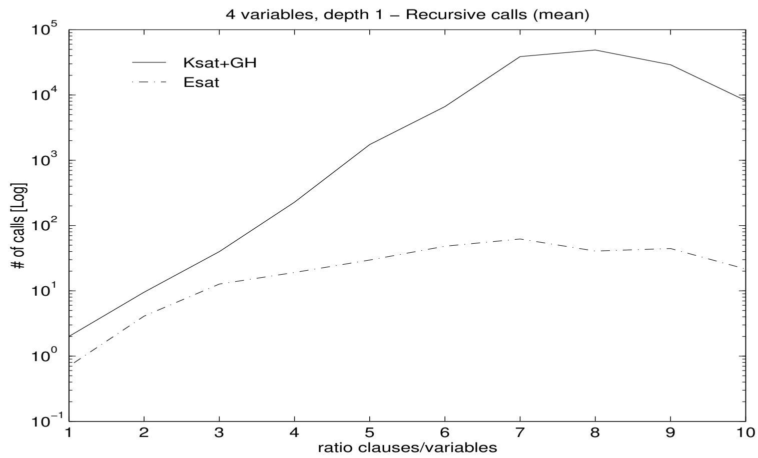

Figura 5.9: Confronto tra il numero di chiamate ricorsive effettuate da ESAT e quelle effettuate da KSAT^t per $N = 4$ e $d = 1$ (altri parametri sono $R = 3$, $B = 0.5$, $T = 0.5$ e 50 campioni per punto).

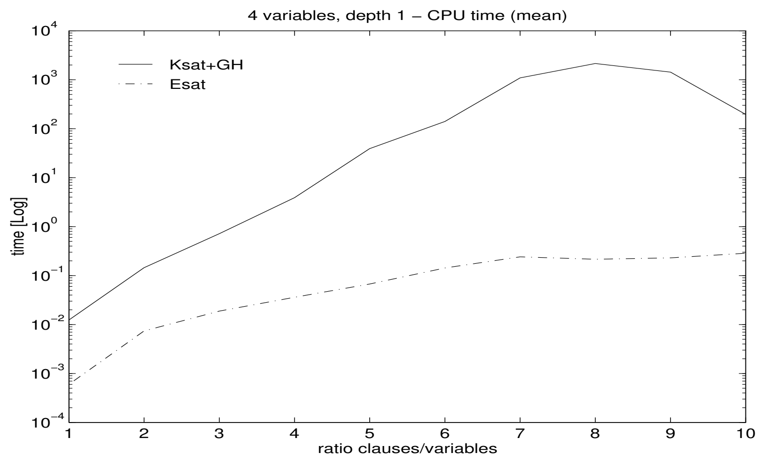

Figura 5.10: Confronto tra i tempi di CPU di ESAT e di KSAT^t al netto dei tempi di preprocessing (ordinamento, semplificazione, etichettatura e, solo per KSAT^t, traduzione) (altri parametri sono $R = 3$, $B = 0.5$, $T = 0.5$ e 50 campioni per punto)

osserviamo che il tempo di CPU considerato è quello privo di tutte le forme di preprocessing, ivi compreso il tempo di traduzione da E a K.

Come si può vedere dalle figure, il risultato di questo esperimento è senz'altro a favore di ESAT. Il divario è infatti molto elevato sia in termini di spazio di ricerca sia in termini di prestazioni complessive. Le chiamate ricorsive effettuate da KSAT^t (figura 5.9) sono sino a quattro ordini di grandezza superiori a quelle effettuate da ESAT e a questo corrisponde una forbice in termini di tempi di esecuzione quasi altrettanto elevata. Aggiungiamo che nella curva presentata in figura 5.10 non è contato il tempo per la traduzione che, a rigore, dovrebbe far parte della valutazione di KSAT^t, in quanto è proprio la traduzione che mette in grado KSAT di decidere formule per E. Tenuto conto di tutto questo, è evidente il fatto che ESAT risulta di gran lunga superiore a KSAT^t.

A parte la conferma del fatto che ESAT, cioè una procedura di decisione costruita appositamente per E, è più efficiente di una traduzione, siamo anche interessati a capire le motivazioni di questo fatto. Le ragioni della scarsa efficienza di KSAT^t sono due:

- la traduzione t può, nel caso pessimo, esplodere *in spazio* rispetto alle dimensioni della formula da tradurre;
- gli algoritmi di decisione in E sono NP-hard, quelli per la decisione in K sono PSPACE-hard (vedi teoremi 3.1 e 3.7 nel capitolo 3;

In definitiva, nonostante KSAT sia un decisore molto efficiente, il peso della traduzione è tale, a nostro avviso, da non rendere praticabile questo metodo per la costruzione di procedure di decisione.

Negli esperimenti che abbiamo appena considerato, i risultati per ESAT sono relativi alla procedura che utilizza il memoizing e cioè che realizza quanto presentato nella sezione 4.5. In pratica, è molto interessante capire se questa si dimostra sempre la scelta più efficiente. A priori, infatti, non è assolutamente detto che il carico di lavoro aggiuntivo per la gestione del memoizing sia più che compensato dal guadagno di prestazioni.

Consideriamo allora un'altro esperimento, questa volta eseguito fissando ad un valore costante il numero di clausole ai livelli diversi da quello più alto. In particolare i grafici presentati in figura 5.11 e in figura 5.12 sono stati ottenuti considerando il problema $N = 3$ e $d = 1$, avendo fissato a $k = 15$ il numero delle clausole. Questo ci ha consentito di ottenere formule abbastanza "dure" senza che l'esplosione in spazio ci fermasse a bassi rapporti di L/N. In tutti i grafici che presentiamo da ora in poi, salvo diversa indicazione, i valori R, T e B si intendono fissati come in figura 5.9. Per quanto riguarda il numero di campioni per punto, sempre salvo diversa indicazione, sono sempre fissati a 100.

I risultati di figura 5.11 confortano la tesi per cui la procedura con il memoizing è più efficiente della corrispondente con il memoizing disabilitato. Questo si spiega con il fatto che tutte le volte che riusciamo a recuperare un risultato nella tabella di memoizing risparmiamo una chiamate ricorsiva. Da solo, questo risultato, conferma che la struttura è efficace nel senso che molte volte riusciamo a ritrovare un risultato già computato. Di fatto, ci sono sino a due ordini di grandezza di differenza tra le curve presente nei

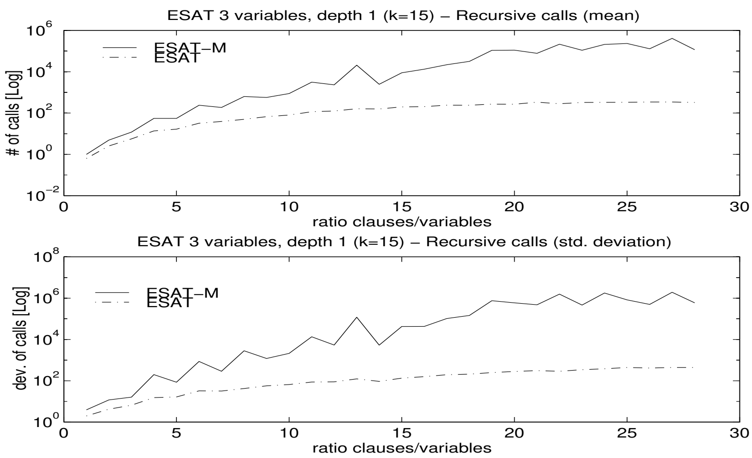

Figura 5.11: Confronto nel numero di chiamate ricorsive tra ESAT con memoizing e la stessa procedura con il memoizing disabilitato per $N = 3$, $d = 1$ e $k = 15$. Il grafico sottostante riporta il valore della deviazione standard per le popolazioni di ogni punto.

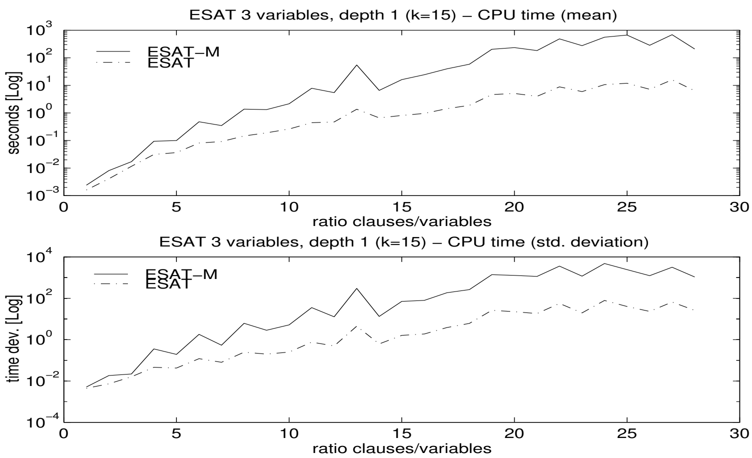

Figura 5.12: Confronto tra tempi di CPU (senza preprocessing) tra ESAT con memoizing la stessa procedura con il memoizing disabilitato per $N = 3$, $d = 1$ e $k = 15$. Il grafico sottostante riporta il valore della deviazione standard per le popolazioni di ogni punto.

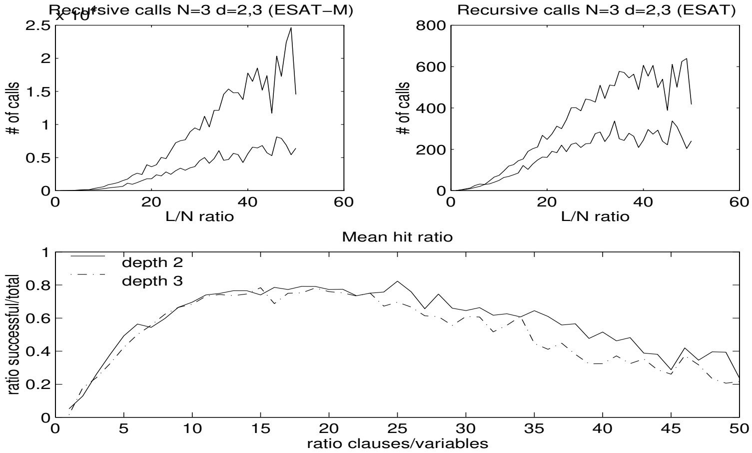

Figura 5.13: ESAT con e senza memoizing per differenti profondità modali con $N = 3$ e $k = 1$. I grafici in alto riportano il numero di chiamate ricorsive: la curva soprastante è, in entrambe i casi, quella per $d = 3$. Il grafico in basso riporta la hit ratio media distinta per $d = 2$ e $d = 3$. Le scale in ordinata sono lineari.

grafici che si traducono in una differenza, in termini di tempo di CPU, di poco più di un ordine di grandezza (figura 5.12). Questo significa che, complessivamente, l'introduzione del memoizing comporta un vantaggio in termini di prestazioni che compensa ampiamente il corrispondente *overhead.*

A questo punto potremmo chiederci quale sia l'andamento delle chiamate ricorsive e dei tempi di CPU, nel momento in cui *scaliamo* verso formule a profondità maggiore. In altre parole, vogliamo capire quanto le due versioni di ESAT (con e senza memoizing) sono reattive ad un incremento della profondità modale delle formule in ingresso.

In figura 5.13 presentiamo i risultati di un esperimento volto a determinare la differenza di comportamento tra ESAT con e senza memoizing passando dalla profondità modale $d = 2$ a $d = 3$. In particolare, tutte le formule hanno $N = 3$ variabili e sono generate tenendo costante il valore delle clausole ai livelli più bassi ($k = 1$). All'interno della figura notiamo tre diversi grafici. I due più in alto confrontano il numero di chiamate ricorsive nel caso con memoizing e nel caso senza memoizing. Notiamo che la procedura senza memoizing è tale che all'aumento della profondità modale corrisponde un aumento da tre a quattro volte nel numero delle chiamate ricorsive, mentre, nel caso di ESAT con memoizing l'aumento è molto più contenuto (circa due volte). Il grafico in basso rappresenta l'andamento della hit ratio per il memoizing (vedi la definizione nella sottosezione 5.2.4). Notiamo che una maggiore profondità modale implica una leggera diminuzione nell'efficienza del memoizing. Questo può dipendere sia dal fatto che,

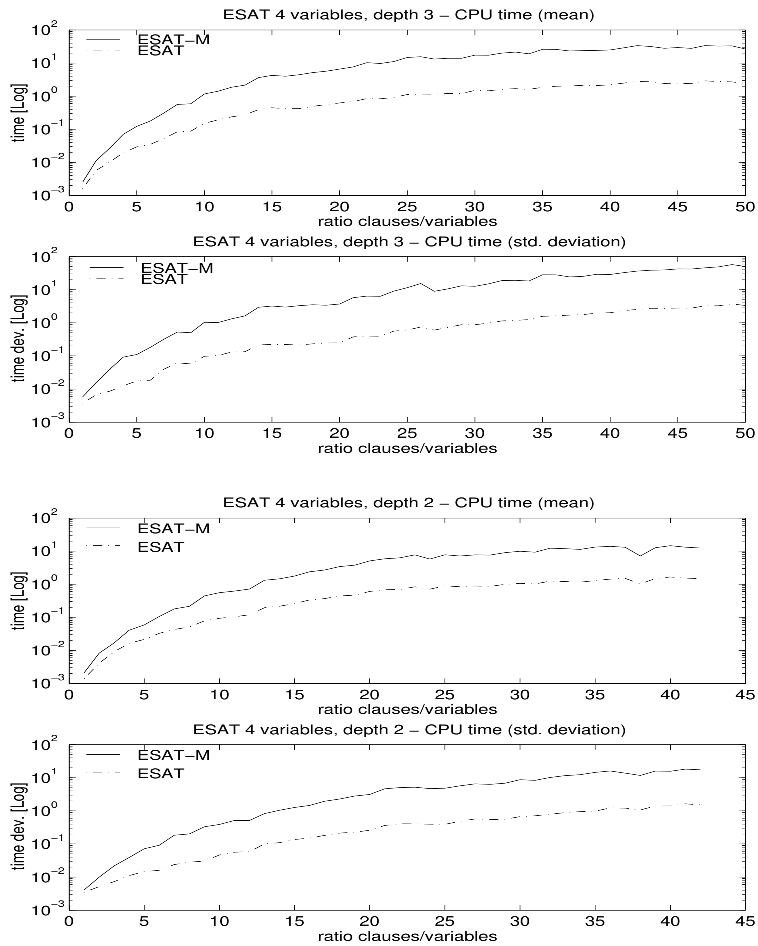

Figura 5.14: Tempi di CPU (senza preprocessing) per ESAT con e senza memoizing per differenti profondità modali con $N = 4$. I due grafici più in alto riportano chiamate ricorsive e deviazione standard per $d = 3$, i due più in basso, gli stessi valori per $d = 4$.

in media, il memoizing viene utilizzato di meno, sia dal fatto che con profondità maggiori è più difficile che vi siano formule uguali, soprattutto ai livelli modali più alti.

Altre conferme all'efficienza delle versioni con il memoizing ci vengono dagli esperimenti con $N = 4$ e $d = 2, 3$ di cui presentiamo i grafici relativi ai tempi di CPU in figura 5.14. Notiamo che in entrambe i casi la differenza di prestazioni si attesta intorno all'ordine di grandezza, ma è senz'altro maggiore nel caso con $d = 3$, rispetto al caso con $d = 2$. L'andamento dei grafici sinora esaminati ci autorizza a supporre che, non solo ESAT è più efficiente nella versione con memoizing, ma anche che la differenza di

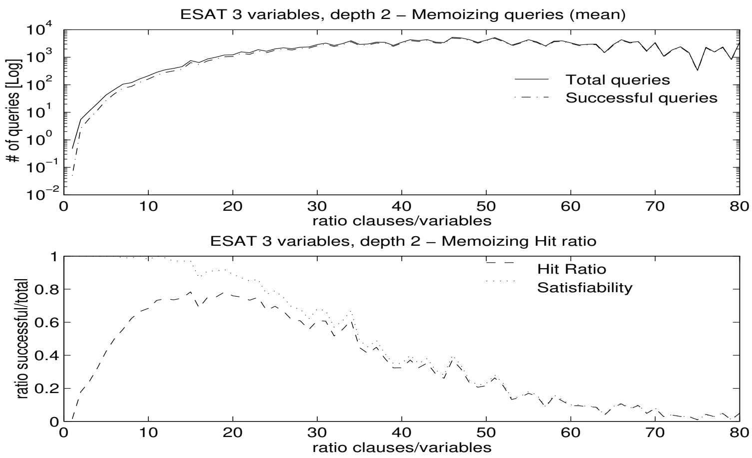

Figura 5.15: Efficienza del memoizing per ESAT Nel grafico più in alto sono riportati il numero delle ricerche totali distinto dal numero di ricerche con esito. Nel grafico inferiore confrontiamo la hit ratio (curva a tratti) con la percentuale di formule soddisfacibili (curva a puntini).

prestazioni rispetto alla versione senza memoizing è destinata a crescere.

Prima di passare ad altri esperimenti, vogliamo soffermarci brevemente per un commento sui grafici nelle figure 5.11, 5.12 e 5.14. Notiamo che ciascuno di essi è accompagnato dal grafico della deviazione standard e che, in tutti i casi, i valori di tale parametro sono nell'ordine di grandezza dei dati stessi. Questo comportamento non stupisce piú di tanto, considerando che lo stesso fenomeno è stato studiato per il caso proposizionale (vedi ad esempio [Mitchell *et al.*, 1992]). Quello che accade è che singole formule, dette *outliers*, sono tali da risultare in tempi di decisione altissimi, fino a vari ordini di grandezza superiori ad altre formule generate con gli stessi parametri. Questo fenomeno, ben lungi dal costituire una "distorsione" nella procedura di testing, sembra invece essere dovuto ad una distribuzione dei campioni per problemi NP-hard intrinsecamente non gaussiana. Le curve di deviazione standard da noi ottenute, sembrano confermare questo comportamento anche per il problema della soddisfacibilità in E.

Concentriamo adesso la nostra attenzione sulla procedura ESAT con memoizing. Innanzi tutto siamo interessati a comprendere meglio l'andamento di alcuni valori legati al memoizing, quali hit ratio, efficienza in spazio, ecc. In figura 5.15 abbiamo riportato due grafici riguardanti le modalità di accesso al memoizing. In particolare si tratta di un problema con $N = 3$, $d = 2$ e k fissato ($k = 1$). Il grafico in alto confronta il numero totale di ricerche nella struttura di memoizing, con il numero di ricerche che hanno successo (per cui il valore restituito è diverso da $\perp$). Come si può vedere dal grafico, dopo una fase iniziale in cui il numero dei successi cresce più rapidamente

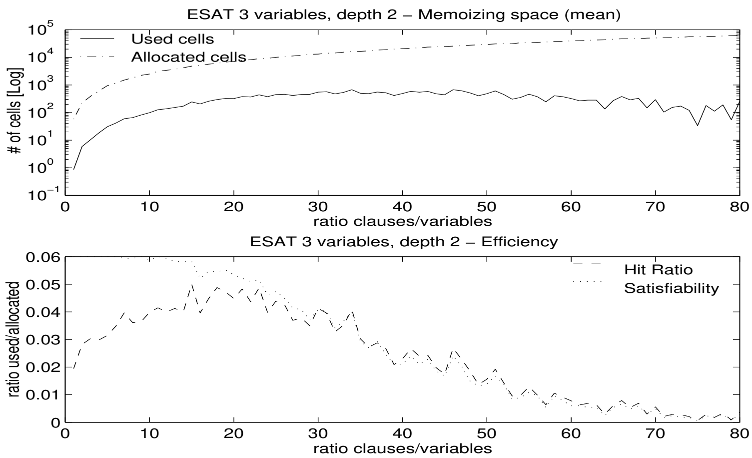

Figura 5.16: Statistiche relative all'allocazione di spazio per ESAT. Il grafico più in alto riporta il totale delle celle di memoria allocate per il memoizing, distinte fra quelle totali e quelle effettivamente utilizzate. Il grafico inferiore mette a confronto l'efficienza in termini di spazio con la percentuale di formule soddisfacibili.

del numero delgi accessi, le due quantità proseguono di pari passo. Questo significa che il memoizing è più efficiente con formule relativamente "grosse". Sempre in figura 5.15 troviamo un grafico che mette a confronto la hit ratio del memoizing con la percentuale di formule soddisfacibili. Il particolare andamento della hit ratio può essere giustificato considerando che nella prima parte del grafico la crescita del numero degli accessi coronati da successo è più veloce rispetto a quella del numero degli accessi totali. Notiamo che la diminuzione della hit ratio media nella seconda parte del grafico va di pari passo con la diminuzione nella percentuale di formule soddisfacibili. Il motivo è che molti campioni utilizzano poco il memoizing in termini di accessi totali (vedi flessione nel numero di ricerche). Questo, e non una minore efficienza del memoizing con formule molto vincolate, è la giustificazione della particolare forma del grafico della hit ratio.

Nella figura 5.16 prendiamo in esame i parametri relativi all'efficienza in termini di spazio del memoizing. In particolare nel grafico più in alto mettiamo a confronto (sempre per $N = 3$ e $d = 2$) lo spazio allocato staticamente con la parte di questo effettivamente utilizzata per memorizzare risultati. È interessante notare che la quantità di spazio allocato cresce *logaritmicamente* su di un grafico in scala logaritmica. Di conseguenza, in accordo con quanto avevamo anticipato nella sezione 4.5, lo spazio occupato da ESAT cresce *polinomialmente* all'aumentare delle dimensioni della formula in ingresso. Viceversa, la quantità di spazio effettivamente utilizzato, decresce all'aumentare delle dimensioni delle formule. Questo è ancora dovuto al fat-

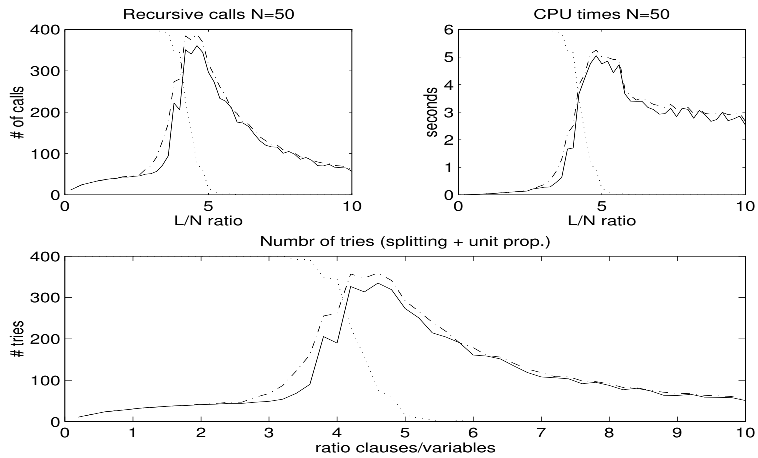

Figura 5.17: Easy-Hard-Easy pattern per SAT. In alto a sinistra, il numero delle chiamate ricorsive, in alto a destra, i tempi di esecuzione e, in basso, il numero di tries hanno il tipico andamento che si ottiene aumentando L/N nel modello fixed-clause-length. Tutti i grafici presentano media (curva a tratti), mediana (curva solida) e percentuale di formule soddisfacibili (curva punteggiata). ($R = 3$, $T = 0.5$ e 100 campioni per punto)

to che, in media, meno formule utilizzano il memoizing, ad esempio perché sono banalmente insoddisfacibili dal punto di vista proposizionale. Notiamo anche che l'efficienza in termini di spazio è comunque molto ridotta, tanto da suggerire l'eventuale applicazione di strutture dati a *matrice sparsa* per la gestione del memoizing. Il più grosso punto a sfavore di questo approccio è dato dal fatto che l'accesso alla struttura deve essere il più veloce possibile.

L'ultima serie di risultati che intendiamo discutere riguarda il particolare comportamento dimostrato da ESAT che sembra ci autorizzi a supporre che alcuni fenomeni ben noti per SAT valgano anche in E. Consideriamo la figura 5.17 in cui riportiamo tre diversi grafici relativi ad un problema SAT con $N = 50$. Come è ben noto da risultati in letteratura (vedi [Mitchell *et al.*, 1992]) l'andamento del grafico di soddisfacibilità presenta una brusca transizione nell'intorno di $L/N = 4.3$ a cui si accompagna un brusco aumento di tutti i valori quali tries, CPU time e numero di chiamate ricorsive. Al limite, per un numero molto grande di variabili, la transizione nella soddisfacibilità tende ad un gradino, mentre tutti gli altri i valori tendono ad un impulso. Questo particolare comportamento prende il nome di *phase transition* per la soddisfacibilità e si dice che gli altri valori mostrano andamenti di tipo *easy-hard-easy*. Questo fenomeno, molto noto e studiato per SAT, è stato dimostrato per una discreta classe di problemi NP-completi (vedi [Sebastiani, 1997] per una bibliografia sull'argomento).

Dal nostro punto di vista è interessante capire se il fenomeno della phase

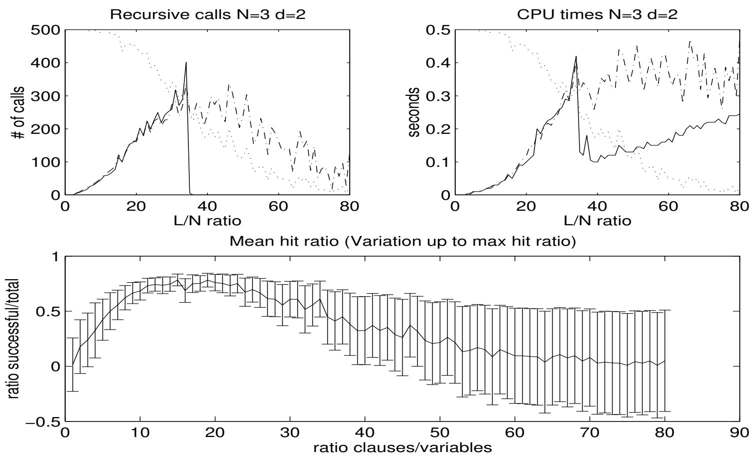

Figura 5.18: Esperimento a favore dell'esistenza di easy-hard-easy pattern per E. I grafici in alto riportano, rispettivamente, il numero di chiamate ricorsive e i tempi di CPU. Nei grafici riportiamo i valori mediani (curva solida), medi (curva tratteggiata), e la percentuale di formule soddisfacibili (curva a punti).

transition si verifica anche per la logica E. L'evidenza sperimentale che abbiamo raccolto, se da un lato non ci permette di affermare che il fenomeno si presenta sicuramente, dall'altro ci consente di congetturarne l'esistenza.

In figura 5.18 vediamo rappresentati i risultati di un esperimento che sembrano confermare le nostre supposizioni. In particolare nei due grafici più in alto che rappresentano, da sinistra verso destra, il numero di chiamate ricorsive e i tempi di CPU, notiamo comportamenti molto simili a quelli evidenziati per SAT. Notiamo che la transizione nella percentuale di formule soddisfacibili è molto più "dolce" rispetto al caso SAT e le campane dei valori sono molto più allargate. L'andamento più tipico è evidente soprattutto considerando i valori mediani (linee solide) mentre i valori medi sono fortemente influenzati dalla presenza degli outliers. Notiamo anche che il picco stabilito dalle curve delle chiamate ricorsive e dei tempi di CPU è, grossomodo, centrato intorno al 50% di formule soddisfacibili, in leggero anticipo rispetto al punto di transizione.

Concludiamo questa parte relativa a ESAT con alcune considerzioni riassuntive riguardanti i dati sperimentali:

- l'analisi preliminare sulla fattibilità di altri metodi di decisione (KSAT^t) che utilizzano risultati presenti in letteratura ha dato chiaramente esito negativo;

- gli esperimenti provano che la corrente realizzazione di ESAT utiliz-

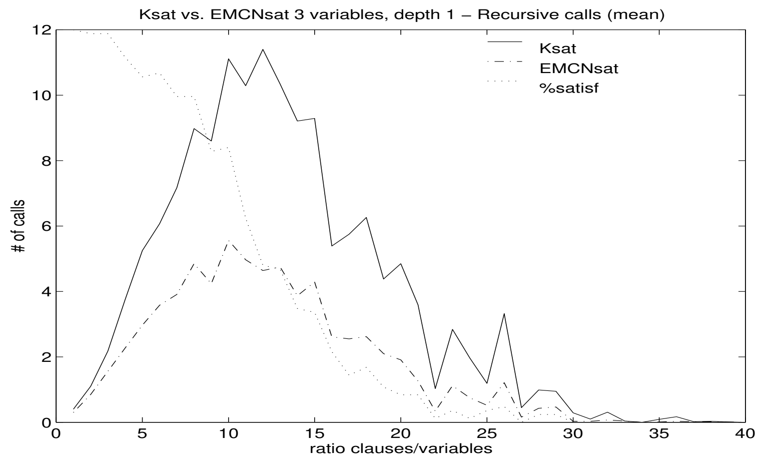

Figura 5.19: Confonto nel numero di chiamate ricorsive tra KSAT e EMCNSAT. La curva "%satisf" rappresenta il numero di formule soddisfacibili (scala 1 : 12).

za tecniche efficaci e poco sensibili all'aumento delle dimensioni del problema;

- in base ai dati raccolti, possiamo congetturare l'esistenza di un fenomeno di phase-transition per E.

5.3.2 La logica K

Come abbiamo preannunciato nella sezione 4.5.3, la logica K risulta particolarmente interessante perché ci consente il confronto con una procedura di decisione considerata allo stato dell'arte. Sempre nella stessa sezione, abbiamo discusso dei particolari punti a favore dell'una e dell'altra procedura che ora intendiamo chiarire dal punto di vista sperimentale.

Gli insiemi di problemi su cui abbiamo confrontato KSAT e EMCNSAT sono $N = 3, 4$ e $d = 1, 2, 3$. Altri parametri sono fissati similmente a quanto abbiamo fatto per E e cioè $T = 0.5$, $B = 0.5$ $R = 3$, ecc. Anche in questo caso i valori di insieme (medie, mediane) sono relativi a 100 campioni per punto. Il numero di clausole ai livelli modali diversi dal più alto è in ogni caso fissato a $k = 1$.

Cominciamo con $N = 1$ e $d = 1$, per cui presentiamo in figura 5.19 il confronto tra chiamate ricorsive e, in figura 5.20, il confronto tra tempi di CPU. In entrambe i grafici abbiamo riportato anche la percentuale di formule soddisfacibili per evidenziare come entrambe gli algoritmi presentino un andamento qualitativo simile. Per inciso, è in base a tale andamento che in [Sebastiani, 1997] si congettura l'esistenza di un fenomeno di phase transition per

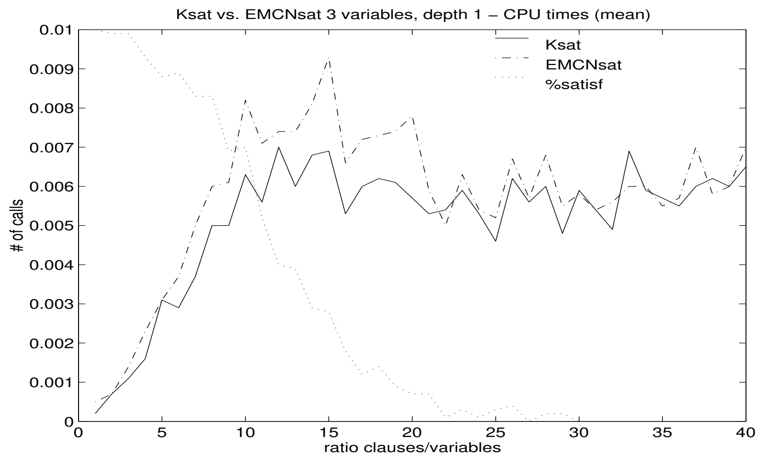

Figura 5.20: Confronto nei tempi di CPU privi di preprocessing tra KSAT e EMCNSAT. La curva "%satisf" rappresenta la percentuale di formule soddisfacibili (scala 1 : 0.01).

la logica K. Dal punto di vista puramente quantitativo, EMCNSAT esegue un numero minore di chiamate ricorsive rispetto a KSAT e questo corrisponde alle nostre aspettative circa la funzionalità del memoizing. A differenza di quanto succedeva con ESAT, i tempi di CPU sembrano smentire il risultato ottenuto considerando le chiamate ricorsive. EMCNSAT risulta complessivamente, più lento di KSAT. Questo significa che con $d = 1$ la gestione del memoizing è ancora troppo onerosa e i benefici troppo ridotti per rendere l'applicazione di tale tecnica conveniente. Un simile risultato è un suggerimento per la realizzazione di un'euristica *EMCN-memoizing-decision* più avanzata di quella utilizzata per questi esperimenti.

A questo punto siamo interessati a capire cosa succede sia all'aumentare delle variabili, sia all'aumentare della profondità. Con questi esperimenti vogliamo verificare che vi sono dei casi in cui il memoizing si dimostra una tecnica efficiente. Altrimenti saremmo costretti a rigettare l'ipotesi iniziale per cui l'introduzione del memoizing può, in media, aumentare l'efficienza dei decisori.

Cominciamo con l'aumentare la profondità modale considerando il problema con $N = 3$ e $d = 2$. Il grafico relativo alle chiamate ricorsive è presentato in figura 5.21. Da quest'ultimo risulta ancora più netta la superiorità di EMCNSAT nell'evitare chiamate ricorsive ridondanti. Inoltre, analizzando il grafico di figura 5.22, possiamo vedere che a questo corrisponde anche un guadagno in termini di prestazioni. Notiamo che la differenza di prestazioni è sempre a vantaggio di EMCNSAT, per cui concludiamo che l'efficacia del memoizing sembra andare di pari passo con l'aumento della profondità modale. Questo concorda con il fatto che che una chiamata ri-

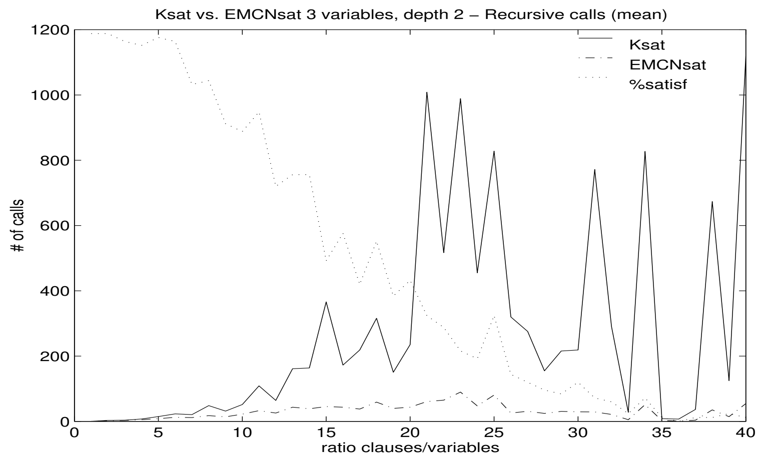

Figura 5.21: Confonto nel numero di chiamate ricorsive tra KSAT e EMCNSAT. La curva "%satisf" è in scala (1 : 1200).

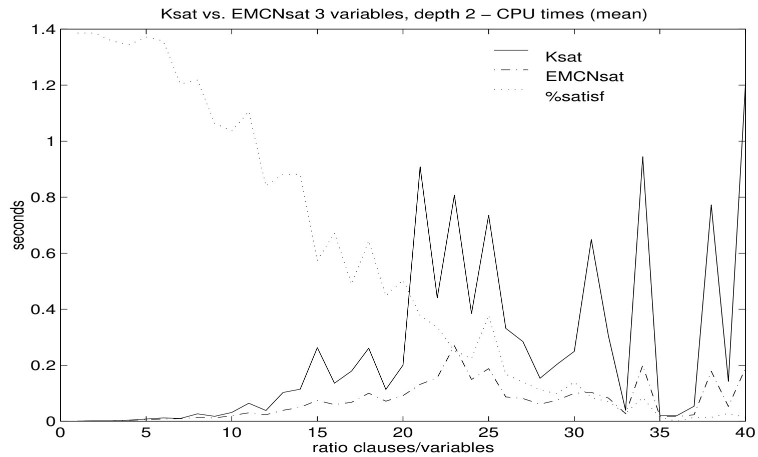

Figura 5.22: Confonto nei tempi di CPU senza preprocessing tra KSAT e EMCNSAT. La curva "%satisf" è in scala (1 : 1.4)

corsiva dovrebbe essere tanto più costosa, in media, quanti più livelli modali sono presenti nella formula.

Proviamo aumentando il numero di variabili e, in particolare, consideriamo il problema per $N = 4$ e $d = 1$. In figura 5.23 presentiamo il grafico relativo ai tempi di CPU da cui si vede chiaramente che, anche in questo caso, il memoizing ci costa più di quello che è il possibile risparmio in termini di tempo.

L'ultima parola al riguardo può venire dai grafici per $N = 4$ e $d = 2$ che

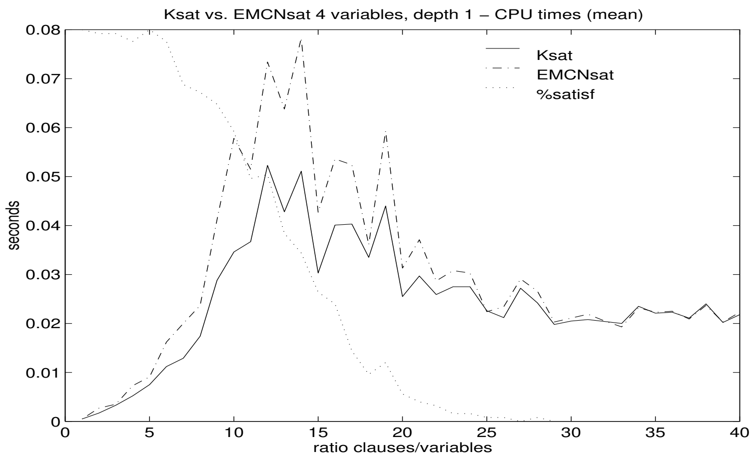

Figura 5.23: Confonto nei tempi di CPU senza preprocessing tra KSAT e EMCNSAT. La curva "%satisf" è in scala (1 : 0.08)

presentiamo in figura 5.24. Ad una differenza che arriva sino a due ordini di grandezza nel numero delle chiamate ricorsive, corrisponde una diminuzione meno che proporzionale nei tempi di esecuzione, ma comunque EMCNSAT risulta in ogni caso più veloce di KSAT.

I risultati che abbiamo ottenuto confrontandoci con KSAT ci consentono di affermare che le metodologie derivate dai decisori per le logiche modali classiche ben si confrontano con una procedura allo stato dell'arte per K. D'altronde, i risultati sperimentali evidenziano la necessità di adotttare tecniche di memoizing più sofisticate di quelle utilizzate oppure di introdurre funzioni di controllo euristico più efficaci di quelle correntemente in uso.

Un'ultima osservazione riguarda lo spazio allocato dalla procedura EMCNSAT che, come avevamo visto nella sezione 4.5 può, nel caso pessimo, esplodere esponenzialmente. Consideriamo a tal proposito la figura 5.25 dove possiamo osservare la dimensione dello spazio allocato per $N = 4$ e profondità modali che vanno da $d = 1$ sino a $d = 3$. Anche se ogni singola curva non risulta in un andamento esplosivo, possiamo notare che la differenza tra le curve raddoppia ad ogni incremento della profondità modale. Combinando questa osservazione con i risultati precedenti possiamo concludere che l'applicazione del memoizing per EMCNSAT è un delicato bilanciamento tra l'aumento di efficacia dato dall'aumento della profondità modale e la corrispondente più rapida saturazione dello spazio di memoria.

Come avevamo anticipato nella sezione 1.3, vogliamo concludere questa sezione presentando un tentativo di confronto empirico tra due procedure di decisione che affrontano problemi di diversa complessità. In particolare, utilizzeremo proprio EMCNSAT e ESAT, come archetipi, rispettivamente, di tutti gli algoritmi PSPACE-hard e di tutti gli algoritmi NP-hard.

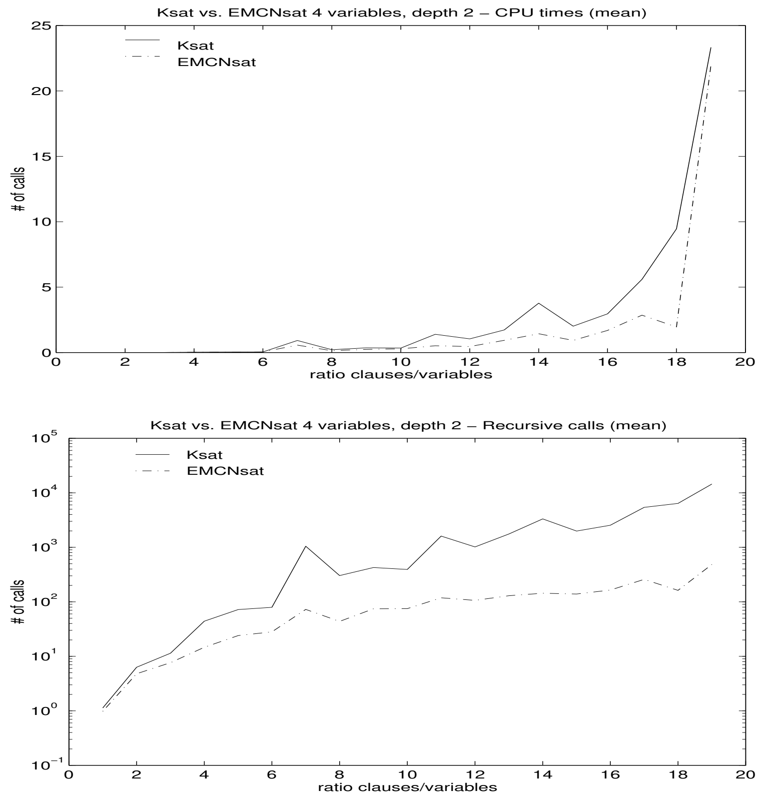

Figura 5.24: Confonto tra Ksat e EMCNsat ($R = 3$, $B = 0.5$, $T = 0.5$ e 100 campioni per punto)

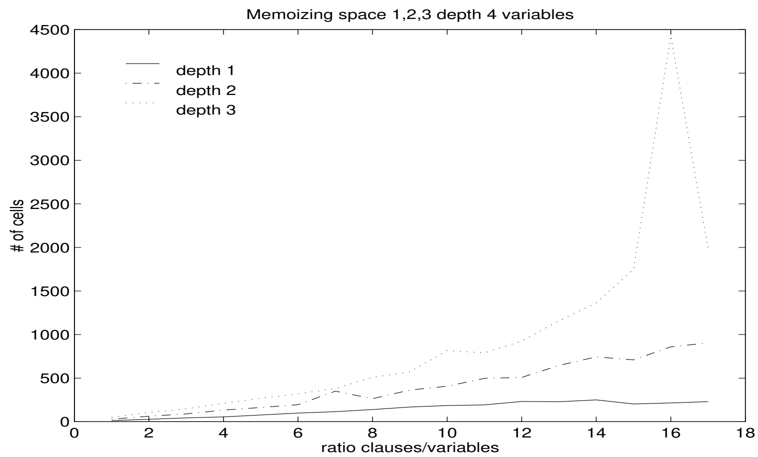

Figura 5.25: Spazio allocato per $N = 4$ $d = 1, 2, 3$

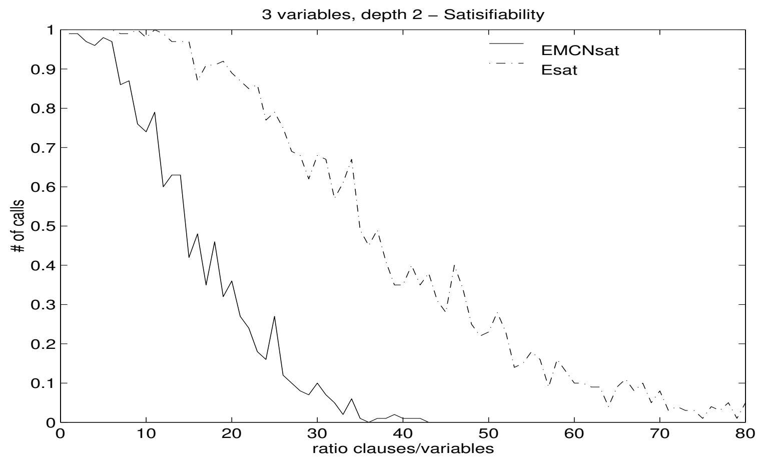

Figura 5.26: Percentuale di formule soddisfacibili per ESAT e EMCNSAT.

Per il confronto utilizziamo il problema $N = 3$ e $d = 2$ (per cui EMCNSAT ha dimostrato di essere superiore a KSAT). Ci aspettiamo che le curve di soddisfacibilità abbiano un diverso comportamento per le due logiche perché non tutto ciò che è soddisfacibile in E è tale anche in K ($\mathcal{K} \subset \mathcal{MS}$). Infatti, notiamo nel grafico di figura 5.26 come la curva corrispondente alla soddisfacibilità in K scenda verso lo 0% molto più rapidamente (per $L/N = 40$ i campioni sono tutti insoddisfacibili in K), mentre la curva per E e spostata molto più verso destra (ancora per $L/N = 80$ ci sono formule soddisfacibili in E). Questo è un primo problema nel confronto fra E e K, in quanto i due decisori incontrano la zona "hard" in due punti diversi del grafico. Infatti EMCNSAT ha un picco intorno a $L/N = 20$ mentre ESAT è più in difficoltà per $L/N = 40$.

La situazione non è destinata a migliorare se si considerano i grafici che mettono a confronto le chiamate ricorsive o i tempi di CPU. Come si può vedere dalla figura 5.27, sorprendentemente, il numero di chiamate ricorsive e i tempi di esecuzione, sono superiori in ESAT rispetto a EMCNSAT. Diciamo sorprendentemente, perché dal punto di vista teorico dovrebbe valere l'esatto contrario, cioè dovrebbe essere EMCNSAT più lenta o soggetta ad un maggior numero di chiamate ricorsive.

La spiegazione di questo fatto può essere ricercata sia nel fatto che il confronto è falsato dalla diversa curva di soddisfacibilità presente nelle due logiche, sia nel fatto che le formule utilizzate per il confronto, anche se banalmente insoddisfacibili in K potrebbero risultare molto ostiche in E. In sostanza, in assenza di ulteriori riscontri basati su diverse metodologie sperimentali o sul confronto tra altri decisori, il problema di capire la reale differenza tra un'algoritmo PSPACE e uno NP, rimane ancora un problema aperto.

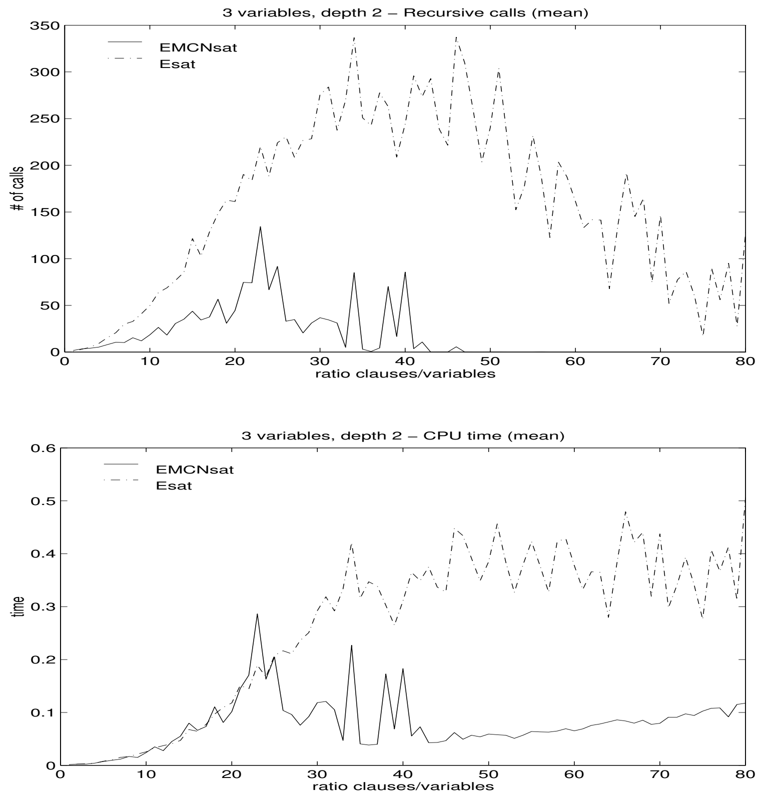

Figura 5.27: Confonto (chiamate ricorsive e tempi di CPU) tra KSAT e EMCNSAT)

Capitolo 6

Conclusioni e futuri sviluppi

Questa tesi presenta una serie di nuovi risultati, che possono essere riassunti come segue:

1. Dimostra che il formalismo SAT-based è applicabile alle logiche modali classiche.
2. Dimostra l'applicabilità pratica del formalismo, realizzando otto diversi decisori DPLL-based. Le procedure di decisione costituiscono il primo tentativo di meccanizzare la deduzione in logiche modali classiche.
3. Presenta in dettaglio l'elaborazione di un metodo sperimentale per le logiche modali classiche.
4. Restingenodosi ad alcuni casi di studio interessanti, compie indagini sperimentali volte a costruire decisori più efficienti, a comparare diversi sistemi o a indagare sulla reale complessità dei problemi decisionali.

Il codice da noi realizzato rappresenta un primo tentativo verso la realizzazione di procedure di decisione modulari, dove la scelta della particolare logica viene lasciata all'utente del sistema.

Il lavoro presentato suggerisce inoltre futuri sviluppi, lungo le seguenti direzioni:

1. Estensione delle procedure di decisione ad altre logiche modali non normali e a teorie logiche.
2. Estensione e consolidamento del metodo sperimentale per una larga classe di logiche modali (normali e non).
3. Costruzione di realizzazioni efficienti con l'ausilio dell'analisi sperimentale.
4. Utilizzo dei decisori e del metodo sperimentale come strumento di indagine nella complessità del ragionamento nelle logiche modali

Appendice A

I decisori in LISP

In questa appendice riportiamo, debitamente commentate, parti selezionate del codice LISP relativo all'implementazione dei decisori oggetto di questa tesi. La funzione delle sezioni seguenti è quella di offrire un utile punto di riferimento per le discussioni contenute nel capitolo 4 e di mostrare come i meccanismi descritti ad alto livello possano di fatto essere realizzati. La prima sezione è dedicata alla strutturazione del codice in moduli comuni a tutti i decisori, più una serie di moduli specializzati per l'inferenza in tutte le logiche Λ. Le sezioni dalla A.2 alla A.5 sono dedicate alla presentazione di alcune parti dei moduli comuni che rivestono una particolare importanza nella struttura generale e cioè la costruzione delle formule, la traduzione di formule in formule etichettate, la procedura di decisione proposizionale e la funzione di valutazione parziale. Le sezioni A.6 ed A.8 sono dedicate alla discussione di due inferenze modali "tipo" e cioè la procedura per la logica E (modello per tutti i decisori NP) e la procedura per la logica $EMCN$ (modello per tutti i decisori PSPACE). In tutti i codici sorgente che presenteremo sono assenti, per motivi di chiarezza, i particolari costrutti per il controllo delle prestazioni sfruttati dall'ambiente di testing e i flag utilizzati per modificare il comportamento dei decisori (abilitazione/disabilitazione di particolari euristiche o procedimenti).

A.1 La strutturazione del codice

In figura A.1 è rappresentata l'architettura completa del sistema che realizza i decisori. L'approccio che seguiamo nel descrivere l'architettura è lo stesso utilizzato nello sviluppo del codice. Pertanto, ogni rettangolo rappresenta un modulo software separato dagli altri e interagente tramite funzioni di interfaccia che ne schermano i dettagli relativi all'implementazione. Le frecce tra moduli rappresentano il fatto che le funzionalità di un modulo sono ncessarie ad un altro per lo svolgimento del suo compito. In particolare la freccia ha la punta rivolta verso l'utente delle funzionalità.

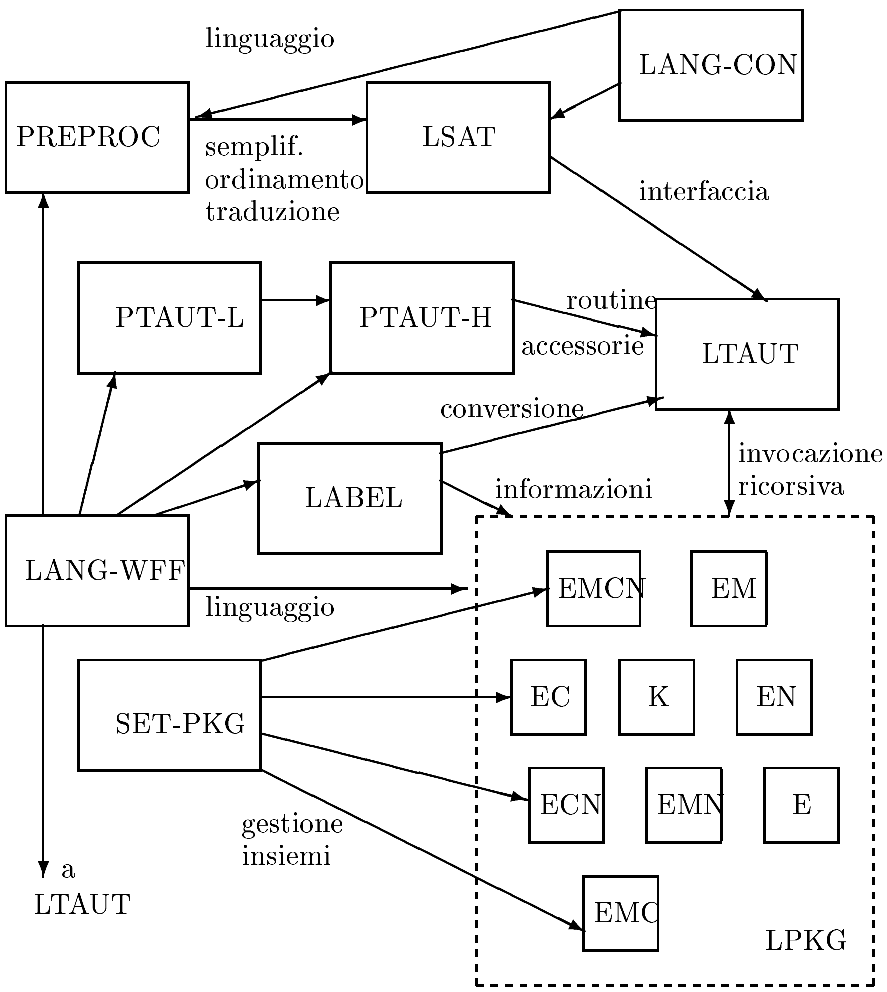

Figura A.1: I moduli software con cui sono realizzate le procedure di decisione (la freccia $A \rightarrow B$ significa che il modulo B utilizza le funzioni messe a disposizione dal modulo A)

Nel seguito, per ogni modulo presentato in figura A.1 diamo una breve descrizione che permette di inquadrarlo all'interno del comportamento globale del codice. Inoltre, per ciascuno di essi diamo l'elenco delle funzioni di interfaccia più importanti esportate all'esterno. Alcune di queste funzioni si trovano descritte in dettaglio nella sezioni seguenti di questo capitolo.

L'architettura del codice può essere considerata composta di tre "blocchi" principali:

[**Interfaccia**] Sono tutti i moduli che si occupano di interfacciare le procedure di decisione verso l'esterno. In questo gruppo annoveriamo `LANG-CON`, `LSAT` e `PREPROC` che forniscono, rispettivamente, un insieme di termini con cui con cui esprimere le formule di $\mathcal{L}$, un decisore "fantoccio" che funge da interfaccia per `LTAUT` e le funzioni di preprocessing(ordinamento, semplificazione e conversione di formule).

[**Decisore SAT**] Sono tutti i moduli comuni ai decisori per le diverse logiche modlai classiche. In questo gruppo inseriamo `LANG_WFF` e `LABEL` (struttura dati e altre utility per formule), `PTAUT-L`, `PTAUT-H` e `LTAUT` (il "nocciolo" del decisore proposizionale).

[**Moduli specifici**] Sono tutti i moduli che contengono la parte specifica relativa ad ogni logica (inferenza modale, memoizing, ecc.). Nella nostra terminologia ci riferiamo ad un generico modulo di questa categoria con il termine *package*. Oltre ai package per le single logiche (contenuti nel rettangolo tratteggiato `LPKG`) annoveriamo in questa categoria anche un modulo di gestione di insiemi come liste ordinate (`SET-PKG`).

Cominciamo la descrizione dal gruppo relativo all'interfaccia verso l'esterno e precisamente dai moduli:

`LANG-CON` esporta un insieme di *costruttori*, *distruttori* e *predicati* che nel loro complesso consento di gestire al generazione induttiva di termini che rappresentano formule di $\mathcal{L}$. La particolarità di questa sintassi, che é legggermente più ricca di quella messa a disposizione da `LANG-WFF`, è quella di gestire i connettivi multipli per "$\wedge$" e per "$\vee$" come liste anziché come alberi di coppie. Questa particolarità rende più compatta la rappresentazione delle formule rispetto a quella utilizzata internamente dai decisori.

`LSAT` esporta la funzione `l-sat-con` che, presi in ingresso una formula descritta con `LANG-CON` e un parametro che seleziona una particolare logica, torna in uscita `T` oppure `NIL` a seconda che la formula sia *soddisfacibile* o meno nella logica considerata. Il suo compito principale è quello di utilizzare le funzioni di `PREPROC` per ordinare e semplificare e convertire nel linguaggio di `LANG-WFF`. La formula così ottenuta è data in ingresso al decisore vero e proprio esportato dal modulo `LTAUT`.

`PREPROC` esporta tre funzioni e cioè `con-sort`, `con-simplify` e `con2prefwff` che eseguono, rispettivamente, ordinamento, semplificazione e conversione delle formule. In particolare `con-sort` e `con-simplify` lavorano su formule rappresentate utilizzando `LANG-CON` in quanto più facili da gestire.

Vediamo ora in dettaglio la descrizione dei moduli che compongono il decisore proposizionale:

`LABEL` esporta diverse utilità, sia per la parte proposizionale sia per la parte modale. La funzione più importante è `label-wff`, descritta nella sezione A.3, che riceve in ingresso una formula nel linguaggio di `LANG-WFF` e la restituisce "etichettata", cioè con tutti gli atomi modali contrassegnati da un numero naturale. Il modulo esporta alcune funzioni accessorie interessanti, tra le quali:

- `agents-list-get` che ritorna la lista di tutti gli agenti utilizzati nella formula;
- `lab-wffs-num-get` restituisce il numero di diverse formule modali etichettate;
- `label-strip`, `conj-mak`, `lab-boxlist` e altre che servono per gestire la compiere varie operazioni con formule etichettate.

`PTAUT-L` esporta tutte le funzionalità di basso livello per `PTAUT-H` (si tratta per la maggior parte di costrutti `defmacro` del LISP).

`PTAUT-H` esporta `MOD-PEVAL` la funzione di valutazione parziale per formule e le funzioni `mod-stats-mak` e `stat2subs` che implementano la ricerca di letterali da assegnare per unit propagation e splitting. `MOD-PEVAL` è descritta in maggiore dettaglio nella sezione A.5.

`LTAUT` esporta la funzione `L-TAUT` che serve per decidere la *validità* di formule modali rappresentate in base a `LANG-WFF`. La routine `L-TAUT-DP` descritta nella sezione A.4 è l'asse portante del modulo, in quanto i compiti di `L-TAUT` sono limitati, essenzialmente, alla etichettatura degli atomi modali e, quando necessaria, all'inizializzazione del memoizing.

Infine, concludiamo la descrizione del sistema con in moduli dedicati all'inferenza modale. Il compito di ciascuno di essi è quello di realizzare le procedure presentate nella sezione 4.5 del capitolo 4. In A.6 esponiamo in dettaglio tutta la struttura del package per la logica E facente capo alla funzione `E-CONSIST`. Nella sezione A.7 presentiamo l'intero mdoulo di gestione insiemi: le sue funzionalità sono necessarei per realizzare la struttura di memoizing necessaria per le logiche EC, ECN, EMC e $EMCN$. Il package per quest'ultima logica è presentato interamente nella sezione A.8

A.2 La descrizione del linguaggio modale

In questa sezione mostriamo le parti di codice funzionale che gestiscono la costruzione della formula. Per ogni simbolo del linguagio modale $\mathcal{L}$ definito nella sezione 2.2, più le abbreviazioni standard, esiste un corrispondente *costruttore* e cioè un'opportuna funzione `mksimbol`. Ogni formula di $\mathcal{L}$ viene mappata in un corrispondente termine costruito induttivamente mediante l'utilizzo dei costruttori (per esempio, da ψ a $\neg\psi$). Per ogni costruttore, esiste il corrispondente *distruttore* e cioè la funzione che dato un termine lo scompone negli argomenti dell'operatore più esterno (per esempio, da $\psi_1 \wedge \psi_2$ a ψ_1). Infine, ad ogni costruttore è associato il *test* `issymbol` che è garantito ritornare `T` ogni volta che l'operatore più esterno del termine corrisponde al costruttore (ad esempio, $\neg\psi$ è una negazione). La lista completa dei costruttori, distruttori e test per la costruzione dei termini `wff` è elencato di seguito:

```
;; Truth constant
; mktruth : wff
(defmacro mktruth () `(quote ,'TRUE))
; istruth : wff -> bool
(defmacro istruth  (w) `(eq ,w 'TRUE))

;; Falsity constant
; mkfalsity : wff
(defmacro mkfalsity () `(quote ,'FALSE))
; isfalsity : wff -> bool
(defmacro isfalsity  (w) `(eq ,w 'FALSE))

;; Variable definition
; mkvar : nat -> wff
(defun mkvar (i)
  (intern (concatenate 'string "A"
                      (write-to-string i))))
; isvar : wff -> bool
(defmacro isvar (w) (list 'atom w))

;; Negation of a formula w
; mkneg : wff -> wff
(defmacro mkneg (w) `(list 'not ,w))
; isneg : wff -> bool
(defmacro isneg (w) `(and (not (isvar ,w))
                          (eq (car ,w) 'not)))
(defmacro negbody (w) (list 'cadr w))
```

```
;; Conjunctions of two formulae w1 and w2
; mkconj : wff * wff -> wff
(defmacro mkconj (w1 w2) `(list 'and ,w1 ,w2))
; isconj : wff -> bool
(defmacro isconj (w) `(and (not (isvar ,w))
                           (eq (car ,w) 'and)))
(defmacro lfconj (w) (list 'cadr w))
(defmacro rtconj (w) (list 'caddr w))

;; Disjunction of two formulae w1 or w2
; mkdisj : wff * wff -> wff
(defmacro mkdisj (w1 w2) `(list 'or ,w1 ,w2))
; isdisj : wff -> bool
(defmacro isdisj (w) `(and (not (isvar ,w))
                           (eq (car ,w) 'or)))
(defmacro lfdisj (w) (list 'cadr w))
(defmacro rtdisj (w) (list 'caddr w))

;; Coimplication between two formulae w1 iff w2
; mkdisj : wff * wff -> wff
(defmacro mkiff (w1 w2) `(list 'iff ,w1 ,w2))
; isiff : wff -> bool
(defmacro isiff (w) `(and (not (isvar ,w))
                          (eq (car ,w) 'iff)))
(defmacro lfiff (w) (list 'cadr w))
(defmacro rtiff (w) (list 'caddr w))

;; Implication between two formulae w1 implies w2
; mkimp : wff * wff -> wff
(defmacro mkimp (w1 w2) `(list 'imp ,w1 ,w2))
; isimp : wff -> bool
(defmacro isimp (w) `(and (not (isvar ,w))
                          (eq (car ,w) 'imp)))
(defmacro hypot (w) (list 'cadr w))
(defmacro concl (w) (list 'caddr w))

;; Box operator (Multiagent systems)
; mkbox-i : nat * wff -> wff
(defmacro mkbox-i (i w) `(list 'box ,i ,w))
(defmacro isbox-i (i w)
 `(and (not (isvar ,w))
       (eq (car ,w) 'box)
       (eql ,i (cadr ,w))))
(defmacro isbox-iany (w)
```

```
 `(and (not (isvar ,w))
       (eq (car ,w) 'box)
       (numberp (cadr ,w))))
(defmacro box-ibody (w) (list 'caddr w))
(defmacro box-iagent (w) (list 'cadr w))
```

A.3 Etichettatura di atomi modali

In questa sezione presentiamo la funzione di traduzione dai termini del linguaggio modale definito nella sezione precedente, nei termini di un nuovo linguaggio, isomorfo al precedente, in cui l'operatore modale rispetta la seguente definizione:

```
;; Labeled boxed atoms (multiagent systems)
; mklab-box : nat * nat * wff -> lwff
(defmacro mklab-box (lab a w)
  `(list 'box ,lab ,a ,w))
(defmacro islab-box (w)
   `(and (not (isvar ,w))
         (eq (car ,w) 'box)
         (numberp (cadr ,w))
         (numberp (caddr ,w))))
(defmacro islab-box-a (w a)
   `(and (not (isvar ,w))
         (eq (car ,w) 'box)
         (numberp (cadr ,w))
         (eql ,a (caddr ,w))))
(defmacro lab-boxbody (w) (list 'cadddr w))
(defmacro lab-boxlab (w) (list 'cadr w))
(defmacro lab-boxa (w) (list 'caddr w))
```

Chiameremo i termini costruiti utilizzando `mklab-box` al posto di `mkbox-i` formule *etichettate*. L'etichettatura si rende necessaria sia perché il decisore proposizionale deve essere in grado di trattare le formule modali come atomi, sia perché, per avere un'implementazione efficiente del memoizing, è necessario potersi riferire alle formule modali con una rappresetnazione più compatta della stessa formula. Ovviamente a formule modali uguali, andranno assegnate etichette uguali e, viceversa, dovremo essere garantiti che atomi modali diversi ricevano diverse etichette. Dato che il nostro linguaggio è, di fatto, multiagente dovremo anche prevedere che formule uguali per egenti diversi ricevano comunque etichette diverse.

```
;; The interface to the labeling function
; label wff -> lwff
(defun label-wff (wff)
```

```
  (progn
    ; Inizializes agents' list and counter
    (agents-list-init)
    ; Initializes the associative table
    (label-table-init)
    ; Reset the label counter
    (reset-label-num)
    ; Label wff
    (rec-label-wff wff)))

;; The labeling function
(defun rec-label-wff (wff)
  (cond
   ;; Leaving variables alone
   ((isvar wff)
    wff)
   ;; Labeling "respects" the structure of the formula
   ((isneg wff)
    (mkneg (rec-label-wff (negbody wff))))
   ((isdisj wff)
    (mkdisj (rec-label-wff (lfdisj wff))
            (rec-label-wff (rtdisj wff))))
   ((isconj wff)
    (mkconj (rec-label-wff (lfconj wff))
            (rec-label-wff (rtconj wff))))
   ((isiff wff)
    (mkiff (rec-label-wff (lfiff wff))
           (rec-label-wff (rtiff wff))))
   ((isimp wff)
    (mkimp (rec-label-wff (hypot wff))
           (rec-label-wff (concl wff))))
   ;; A multiagent box was found, we need to label it
   ((isbox-iany wff)
    ; test if (body,agent) was labeled yet
    (let ((label (label-get (box-ibody wff)
                             (box-iagent wff))))
      (if (NULL label)
          ; get a new label ...
          (let ((next-label (next-label-get)))
            (progn
              ; store the new association
              (label-store next-label
                           (box-ibody wff)
                           (box-iagent wff))
```

```
              ; recursively label the body
              (mklab-box
               next-label
               (box-iagent wff)
               (rec-label-wff (box-ibody wff)))))
        (mklab-box
         ; use the retrieved label and go on
         label
         (box-iagent wff)
         (rec-label-wff (box-ibody wff))))))))
```

A.4 Il decisore L-TAUT-DP

In questa sezione presentiamo il codice di L-TAUT-DP il decisore tautologico proposizionale alla base di tutti i decisori. Si tratta di una realizzazione dell'algoritmo DPLL per il controllo di validità nella logica proposizionale modificato per gestire formule non CNF. La procedura è, con minori modifiche, la stessa descritta nell'articolo [Armando e Giunchiglia, 1993]. In particolare è parametrizzata rispetto alla particolare logica modale ed è in grado di gestire formule modali come atomi. Si noti che per ottenere la corrispondente realizzazione di $\Lambda\textsc{sat}_{DP}$ è sufficiente scirvere la seguente funzione LISP:

```
(defun L-SAT-DP (wff logic)
  (NOT (L-TAUT-DP (mkneg wff) NIL logic)))
```

in quanto sappiamo che per tutte le logiche Λ una formula è soddisfacibile se e solo se $\not\vdash_\Lambda \neg\varphi$.

```
;; DPLL for propositional tautology.
;; The main structure works like this:
;;
;; 1 - Partially evaluate wff
;; 2 - Work out statistics
;;     (e.g. number of occurences for each literal)
;; 3 - Work out possible substitutions
;;     (e.g. most frequent literal)
;; 4 - Decide which of the following is the case:
;;            - Backtracking
;;            - Calling L-CONSIST
;;            - Solving an unit clause
;;            - Early prune the search
;;              by invoking L-CONSIST
;;            - Solving by splitting on a literal
;; Note that the assignment works like a
```

```
;; stack: (CAR assgn) is the last
;; couple (literal . truth value) added.
;; Here we put the original structure of the
;; algorithm: the real one is
;; somewhat more complex because we added
;; performance monitoring parameters

(defun L-TAUT-DP (wff assgn logic)
 ; Partial evaluation of wff.
 (let ((nwff (MOD-PEVAL wff (car assgn))))
   ; Get statistics.
   (let ((stats (l-mod-stats-mak nwff logic)))
     ; Get substitutions and find unit clauses
     (let ((subs (l-stats2subs stats logic)))
       (cond ((DTLDO? subs)
            ;; Backtrack 1.
            T)
           ((NOT (UNDEF-SUBS? subs))
            ;; Unit clause.
            (L-TAUT-DP nwff
                      (cons (car subs) assgn) logic))
           ((NULL stats)
            (if (istruth nwff)
                ;; Backtrack 2.
                T
              ;; Call for final modal consistency check.
              (NOT (L-CONSIST assgn logic))))
           ((AND (L-LIKELY-UNCONSIST assgn logic)
                 ;; Early pruning
                 (NOT (L-CONSIST assgn logic)))
            ;; Backtrack 3.
            T)
           (T
            ;; Splitting.
            (let ((pc (l-pc2sub-get stats logic)))
             (AND (L-TAUT-DP nwff
                            (cons (sub-mak pc (mktruth))
                            assgn))
                  (L-TAUT-DP nwff
                            (cons (sub-mak pc (mkfalsity))
                            assgn))))))))))
```

A.5 Valutazione parziale di formule

La procedura `L-TAUT-DP` descritta nella sezione precedente necessita di una procedura di valutazione parziale che gestisca formule non CNF e formule modali etichettate. Presentiamo il codice LISP per la procedura `MOD-PEVAL` che esegue tale compito. In particolare la formula viene semplificata utlizzando una discesa ricorsiva lazy nella struttura del temine che la rappresenta. Nel codice che segue, ad ogni semplificazione eseguita dalla procedura è associato un commento sulla particolare inferenza effettuata.

```
;; This is the propositional partial evaluator.
;; It has been modified in order to handle just one
;; substitution at a time and to handle labeled boxes.
(defun MOD-PEVAL (wff sub)
  (if (isvar wff)
    ;; If w is a variable and is to be assigned
    ;; then assign it
    (if (EQUAL (sub-get-subed sub) wff)
      (sub-get-subing sub)
     wff)
   (if (islab-box wff)
     ;; If w is a boxed atom and is to be assigned
     ;; then assign it
     (if (AND (islab-box (sub-get-subed sub))
              (EQUAL (lab-boxlab (sub-get-subed sub))
                     (lab-boxlab wff)))
       (sub-get-subing sub)
       wff)
    (if (isneg wff)
      (let ((w (MOD-PEVAL (negbody wff) sub)))
      ;; negation of truth is falsity
      (if (istruth w) (mkfalsity)
        ;; negation of falsity is truth
        (if (isfalsity w) (mktruth)
         ;; return the negation of the evaluated body
         (mkneg w))))
     (if (isdisj wff)
       (let ((w1 (MOD-PEVAL (lfdisj wff) sub)))
       ;; if A is false then AvB = B
       (if (isfalsity w1) (MOD-PEVAL (rtdisj wff) sub)
         ;; if A is true then AvB = true
         (if (istruth w1) (mktruth)
           (let ((w2 (MOD-PEVAL (rtdisj wff) sub)))
           ;; if B is false then AvB = A
```

```
        (if (isfalsity w2) w1
          ;; if B is true then AvB = true
          (if (istruth w2) (mktruth)
            ;; otherwise return the disjunction
           (mkdisj w1 w2)))))))
   (if (isconj wff)
     (let ((w1 (MOD-PEVAL (lfconj wff) sub)))
     ;; if A is false the A&B is false
     (if (isfalsity w1) (mkfalsity)
       ;; if A is true then A&B = B
       (if (istruth w1) (MOD-PEVAL (rtconj wff) sub)
         (let ((w2 (MOD-PEVAL (rtconj wff) sub)))
         ;; if B is false the A&B is false
         (if (isfalsity w2) (mkfalsity)
           ;; if B is true then A&B = A
           (if (istruth w2) w1 (mkconj w1 w2)))))))
    (if (isimp wff)
      (let ((w1 (MOD-PEVAL (hypot wff) sub)))
      ;; if A is false then A->B is true
      (if (isfalsity w1) (mktruth)
        ;; if A is true then A->B = B
        (if (istruth w1) (MOD-PEVAL (concl wff) sub)
          (let ((w2 (MOD-PEVAL (concl wff) sub)))
          ;; if B is false then A->B = -A
          (if (isfalsity w2) (mkneg w1)
            ;; if B is true then A->B is true
            (if (istruth w2) (mktruth) (mkimp w1 w2)))))))
     (if (isiff wff)
       (let ((w1 (MOD-PEVAL (lfiff wff) sub)))
       ;; if A is false then A<->B = -B
       (if (isfalsity w1) (MOD-PEVAL
                             (mkneg (rtiff wff)) sub)
         ;; if A is true then A<->B = B
         (if (istruth w1) (MOD-PEVAL (rtiff wff) sub)
           (let ((w2 (MOD-PEVAL (rtiff wff) sub)))
           ;; if B is false then A<->B = -A
           (if (isfalsity w2) (mkneg w1)
             ;; if B is true then A<->B = A
             (if (istruth w2) w1 (mkiff w1 w2)))))))
      (PRINC "MOD-PEVAL : Fatal error !!!")))))))))
```

A.6 Memoizing e inferenza modale per E

In questa sezione presentiamo la realizzazione in LISP della procedura ESAT_A con memoizing. In particolare, presentiamo le primitive per il controllo della memorizzazione, la struttura dati utilizzata, e le procedura per il controllo dell'inferenza. La struttura per il memoizing è un array statico bidimensionale. Le dimensioni con cui l'array è inizializzato corrispondono al numero di sottoformule modali $\mid \varphi \mid$ e possono ricavate con una funzione accessoria del modulo di labeling (vedi A.1). Questo perché, nel momento in cui una formula viene tradotta da `wff` a `lwff` è possibile tenere traccia del numero totale di etichette assegnate e quindi del numero totale di possibili elementi per la matrice di memoizing. Vediamo allora il codice relativo all'inizializzazione della struttura e alle primitive di accesso:

```
;;The memoizing structure
(defvar %E-memo% NIL)

;; Initializes the memoizing matrix (we make sure
;; the structure is clean and it is implemented in
;; an efficient way).
(defun E-memo-init (dim)
   (setf %E-memo% (make-array (list dim dim)
                              :initial-element NIL
                              :adjustable NIL
                              :fill-pointer NIL
                              :displaced-to NIL)))

;; Returns the value memoized for key1,key2.
;; E-memo-get : lab x lab -> {Y, N, NIL}
(defun E-memo-get (key1 key2)
    (aref %E-memo% key1 key2))

;; Stores the value corresponding to key1,key2 -
;; this function exploits  the fact that if the formulae
;; tested are in the form (a iff b)
;; ordering between the two keys doesn't matter.
;; E-memo-put2: lab x lab x bool -> bool
(defun E-memo-put2 (key1 key2 item)
    (progn
      (setf (aref %E-memo% key1 key2) (memo-trans item))
      (MEMO-UNTRANS
        (setf
          (aref %E-memo% key2 key1) (memo-trans item)))))
```

```
;; Translates T to Y and NIL to N
(defun memo-trans (bool) (if bool 'Y 'N))

;; Untranslates Y to T and N to NIL)
(defun MEMO-UNTRANS (item) (if (EQ item 'Y)
                               T
                               NIL))
```

Vediamo ora la cascata di funzioni che presiede all'inferenza modale. A parte l'estensione al caso multiagente che non è stata descritta nei capitoli precedenti, le procedure riproducono lo schema di inferenza presentato nella sezione 4.5.1 del testo.

```
;; Extract an agent list using agent-list-get
(defun E-CONSIST (cm)
  (let ((agents-list (agents-list-get)))  ; <label.lsp>
    (E-CONSIST1 cm agents-list)))

;; For every agent in the list, apply the
;; consistency test on cm restricted to the agent
(defun E-CONSIST1 (cm agents-list)
  (if (NULL agents-list)
      T
    (AND (E-CONSIST2 cm (car agents-list))
         (E-CONSIST1 cm (cdr agents-list)))))

(defun E-CONSIST2 (cm agent)
  ;; box-neg-wffs extracts boxed atoms assigned to false
  ;; box-pos-wffs extracts boxed atoms assigned to true
  (let ((box-neg-wffs (box-neg-wffs-get cm agent))
        (box-pos-wffs (box-pos-wffs-get cm agent)))
    (if (OR (NULL box-neg-wffs)
            (NULL box-pos-wffs))
        ;; Assignments w/o alpha(s) or beta(s) are
        ;; trivially consistent in E
        T)
        ;; If there are alpha(s) and beta(s) go on
      (E-CONSIST3 box-pos-wffs box-neg-wffs))))

;; This functions iterates the E-consistency
;; test on positive boxed atoms.
(defun E-CONSIST3 (box-pos-wffs box-neg-wffs)
  (if (NULL box-pos-wffs)
       T
      (AND (E-CONSIST4 (car box-pos-wffs)
```

```
                        box-neg-wffs)
           (E-CONSIST-MEMO (cdr box-pos-wffs)
                           box-neg-wffs))))


;; The modal consistency check with
;; "test & memoizing" (iterating on positive boxed atoms).
(defun E-CONSIST-MEMO (box-alpha box-neg-wffs)
  (if (NULL box-neg-wffs)
      T
      (AND (let ((ris (E-memo-get
                         (lab-boxlab box-alpha)
                         (lab-boxlab box-beta))))
             (if (NULL ris)
                 ;; Test alpha iff beta and
                 ;; memoize the result using labels as keys.
                 (NOT
                  (E-memo-put2
                   (lab-boxlab box-alpha)
                   (lab-boxlab box-beta)
                   (L-TAUT1
                     (mkiff (lab-boxbody box-alpha)
                            (lab-boxbody box-beta)) 'E)))
              ;; Return the result already memoized
              (NOT (MEMO-UNTRANS ris))))
           (E-CONSIST-MEMO box-alpha (cdr box-neg-wffs)))))
```

A.7 Gestione di insiemi

```
;; Recursive initialization of a memoizing structure
;; Automatic initialization fails to be an acceptable
;; method because every cell points to the same element
;; and we want cells to be distinct structures.
(defun rec-init (array cur dim)
  (if (EQL cur dim)
      NIL
    (progn
      (setf (aref array cur) (cons 'dummy NIL))
      (memo-uspsp-inc 1)
      (rec-init array (+ cur 1) dim))))


;; This predicate tests the particular structure that
```

```
;; is used to terminate a list instead of NIL.
(defun DUMMYP (node)
  (AND (EQ (car node) 'dummy)
       (EQ (cdr node) NIL)))

;; A three valued subset/order testing routine;
;; Returned values are:
;; - 0 A is a subset of B
;; - 1 A is not a subset of B and A<B (lexicographically)
;; - 2 A is not a subset of B and A>B (lexicographically)
(defun subset (A B)
  (subset-acc A B 0)) ; by default that A is a subset of B

(defun subset-acc (A B acc)
  (if (NULL A)
      0
    (if (NULL B)
        ;; A subseteq B up to this point
        ;; but this reveals that A>B
        (if (EQL acc 0) 2 acc)
      (if (EQL (car A) (car B))
          ;; test (cdr a) supseteq (cdr B)
          (subset-acc (cdr A) (cdr B) acc)
        (if (< (car A) (car B))
            ;; A subseteq B up to this point
            ;; but this reveals that A<B
            (if (EQL acc 0) 1 acc)
            ;; A<B bt there is still hope that A subset B
          (subset-acc A (cdr B) 2))))))


;; Returns (list A) whenever A is a subset of at least
;; one set in B-pool, an insertion point for a new
;; element otherwise. A is a sorted list, while B
;; is a lexicographically sorted list  of sorted lists;
;; lexicographic order is to be ascendant.
(defun subset-POOL (A B-pool)
  (if (DUMMYP B-pool)
      ;; Insertion point comes at the end
      B-pool
    (let ((res (subset A (car B-pool))))
      (cond ((EQL res 0)
             ;; A is a subseteq of an element of B
             (list A))
```

```
            ((EQL res 1)
             ;; A is not a subset of (car B-pool) and A<B:
             ;; this is an insertion point.
             B-pool)
            ((EQL res 2)
             ;; A is not a subset of (car B-pool) and A>B:
             ;; there is hope that we can find a
             ;; supset of A in (cdr B-pool)
             (subset-POOL A (cdr B-pool)))))))

;; Returns (list A) whenever A is a supset of at least
;; one set in B-pool, an insertion point for a new
;; element otherwise. A is a sorted list, while B
;; is a lexicographically sorted list of sorted lists;
;; lexicographic order is to be descendant.
(defun SUPSET-POOL (A B-pool)
  (if (DUMMYP B-pool)
      ;; Insertion point comes at the end
      B-pool
    (let ((res (subset (car B-pool) A)))
      (cond ((EQL res 0)
             ;; A is a supseteq of an element of B
             (list A))
            ((EQL res 1)
             ;; A is not a supset of (car B-pool) and A>B:
             ;; this is an insertion point.
             B-pool)
            ((EQL res 2)
             ;; A is not a supset of (car B-pool) and A<B:
             ;; there is hope that we can find a
             ;; subset of A in (cdr B-pool)
             (supset-POOL A (cdr B-pool)))))))
```

A.8 Memoizing e inferenza modale per $EMCN$

Presentiamo, innanzi tutto la gestione del memoizing.

```
;;; SECTION 3 - MEMOIZING STRUCTURE AND HANDLING

;; The structure is the same as the one for logic EC
(defun EMCN-memo-init (dim)
  (progn (EMCN-memo-init-np dim)
         (EMCN-memo-init-psp dim)))
```

```
;; The memoizing structure for the test (not beta)
(defvar %EMCN-memo-np% NIL)

;; Initializes the memoizing matrix
;; (we make sure the structure is clean and
;; it is implemented in an efficient way).
(defun EMCN-memo-init-np (dim)
  (setf %EMCN-memo-np% (make-array dim
                                   :initial-element NIL
                                   :adjustable NIL
                                   :fill-pointer NIL
                                   :displaced-to NIL))

;; Translates T to Y and NIL to N
(defun memo-trans (bool) (if bool 'Y 'N))

;; Untranslates Y to T and N to NIL)
(defun MEMO-UNTRANS (item) (if (EQ item 'Y)
                                T
                                NIL))

;; The memoizing structure for the test
;; (AND_i alpha_i and not beta)
(defvar %EMCN-memo-psp-SAT% NIL)
(defvar %EMCN-memo-psp-VAL% NIL)

;; Some global variables used in
;; the search-insertion process
; stores the sorted alpha list (avoids sorting twice)
(defvar %EMCN-sorted-alpha%)
; stores the insertion point in the validity structure
(defvar %EMCN-tmp-val%)
; stores th einsertion point in the satisf. structure
(defvar %EMCN-tmp-sat%)

;; This function initializes the memoizing structure
(defun EMCN-memo-init-psp (dim)
  (progn
    ;; This structure holds satisfiability results and is
    ;; searched for superset inclusion
    (setf %EMCN-memo-psp-SAT%
                       (make-array dim
                                   :initial-element NIL
                                   :adjustable NIL
```

```
                                    :fill-pointer NIL
                                    :displaced-to NIL))
    (rec-init %EMCN-memo-psp-SAT% 0 dim)
    ;; This structure holds validity results and is
    ;; searched fo subset inclusion
    (setf %EMCN-memo-psp-VAL%
                        (make-array dim
                                    :initial-element NIL
                                    :adjustable NIL
                                    :fill-pointer NIL
                                    :displaced-to NIL))
    (rec-init %EMCN-memo-psp-VAL% 0 dim))))


;; This function has a lot of tasks. Fisrt of all
;; it sorts the alpha list and
;; stores it in a global (this is for efficient updates).
;; Then the structure for satisfiability
;; is searched for supsets of the current
;; alpha list (Y is returned on match).
;; If none is found then the list for
;; validity is searched for subsets (N is returned on match).
;; If there is no response at all NIL is returned.
;; Insertion points are stored in global
;; variables (this is for efficient updates).
(defun EMCN-memo-get-psp (alpha-labels beta-label)
  (if (NULL alpha-labels)
      (progn
        (setq %EMCN-sorted-alpha% NIL)
        (MEMO-UNTRANS (aref %EMCN-memo-np% beta-label)))
    (progn
      (setq %EMCN-sorted-alpha% alpha-labels)
      (setq %EMCN-tmp-sat%
         (subset-pool %EMCN-sorted-alpha%
                      (aref %EMCN-memo-psp-SAT%
                            beta-label)))
      (if (EQUAL %EMCN-tmp-sat%
                 (list %EMCN-sorted-alpha%))
          'N
        (progn
          (setq %EMCN-tmp-val%
             (supset-pool %EMCN-sorted-alpha%
                          (aref %EMCN-memo-psp-VAL%
                                beta-label)))
```

```
          (if (EQUAL %EMCN-tmp-val%
                     (list %EMCN-sorted-alpha%))
              'Y
            NIL))))))

;; This function updates the lists that have previously
;; been searched. If the result is T then the validity
;; structure is updated, otherwise the satisfiability
;; structure is updated. in the last case the information
;; regarding the satisfiability of (not beta) is recorded too.
(defun filtered-EMCN-memo-put-psp (beta-label result)
  (if (NULL %EMCN-sorted-alpha%)
      (progn (setf (aref %EMCN-memo-np% beta-label)
                   (memo-trans result))
             result)
      (if result
          ;; Store the first element in a temp cell
          (let ((tmpca (car %EMCN-tmp-val%)))
            ;; Replace the first element
            ;; with the current alpha list
            (rplaca %EMCN-tmp-val% %EMCN-sorted-alpha%)
            ;; Replace the cdr with the old list
            (rplacd %EMCN-tmp-val%
                    (cons tmpca (cdr %EMCN-tmp-val%))))
        (let ((tmpca (car %EMCN-tmp-sat%)))
          (rplaca %EMCN-tmp-sat% %EMCN-sorted-alpha%)
          (rplacd %EMCN-tmp-sat%
                  (cons tmpca (cdr %EMCN-tmp-sat%)))
          ;; Look: we can update information
          ;; for N-consistency!
          (setf (aref %EMCN-memo-np% beta-label)
                    (memo-trans result)))
      ;; Return result as it is
      result)))
```

Vediamo il codice realizzato

```
(defun EMCN-CONSIST (cm)
  (let ((agents-list (agents-list-get)))
    (EMCN-CONSIST1 cm agents-list)))

(defun EMCN-CONSIST1 (cm agents-list)
  (if (NULL agents-list)
      T
```

```
   (AND (EMCN-CONSIST2 cm (car agents-list))
        (EMCN-CONSIST1 cm (cdr agents-list)))))

(defun EMCN-CONSIST2 (cm agent)
  (let ((box-neg-wffs (box-neg-wffs-get cm agent))
        (box-pos-wffs (box-pos-wffs-get cm agent)))
    (if (NULL box-neg-wffs)
        T
      (EMCN-APPLY-PSPACE-TEST box-pos-wffs box-neg-wffs))))

;; This function selects raw pspace-test versus pspace-test
;; plus memoizing
(defun EMCN-APPLY-PSPACE-TEST (box-pos-wffs box-neg-wffs)
 (if (NULL box-neg-wffs)
     T
   ;; NOTE: conj-mak strips labels away and
   ;;       forms a conjunction of wffs!!
   ;; NOTE: lab-boxlist strips wffs away and
   ;;       forms a label list!!
   (EMCN-PSPACE-TEST-MEMO (conj-mak box-pos-wffs)
                          (sort (lab-boxlist box-pos-wffs) '<)
                          box-neg-wffs)))

;; This test succeeds if (AND_i alpha and not beta)
;; is satisfiable for every beta in box-neg-wffs.
;; The same as before but with memoizing aids
(defun EMCN-PSPACE-TEST-MEMO (box-pos-conj
                              box-pos-labs box-neg-wffs)
  (if (NULL box-neg-wffs)
      T
    (AND
     (let ((result
            (EMCN-memo-get-psp
              box-pos-labs
              (lab-boxlab (car box-neg-wffs)))))
       (if (NULL result)
           ;; Recursively test satifiability
           ;; (AND_i alpha and not beta)
           (NOT
            (filtered-EMCN-memo-put-psp
             (lab-boxlab (car box-neg-wffs))
             (L-TAUT1
              (mkimp box-pos-conj
                     (lab-boxbody (car box-neg-wffs))) 'EMCN)))
```

```
        ;; Return the result already memoized
        (NOT (MEMO-UNTRANS result))))
    (EMCN-PSPACE-TEST-MEMO
        box-pos-conj box-pos-labs (cdr box-neg-wffs)))))
```

Bibliografia

[Armando e Giunchiglia, 1993] A. Armando e E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[Benerecetti *et al.*, 1996] M. Benerecetti, A. Cimatti, E. Giunchiglia, F. Giunchiglia, e L. Serafini. Formal Specification of Multi-Agent Systems. Technical Report 9605-05, IRST, 1996. To appear in the proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), ECAI-96, Budapest, Hungary.

[Chellas, 1980] B. F. Chellas. *Modal Logic – an Introduction.* Cambridge University Press, 1980.

[Davis *et al.*, 1962] M. Davis, G. Longemann, e D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[Davis e Putnam, 1960] M. Davis e H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Fagin *et al.*, 1995] R. Fagin, J.Y. Halpern, Y. Moses, e M. Y. Vardi. *Reasoning about knowledge.* MIT Press, 1995.

[Fitting, 1983] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishg, 1983.

[Garey e Johnson, 1979] M. R. Garey e D. S. Johnson. *Computers and Intractability.* Freeman and Company, New York, 1979.

[Gasquet e Herzig, 1996] Olivier Gasquet e Andreas Herzig. From classical to normal modal logics. In *Proof Theory of Modal Logic.* Kluwer Academic Publishers, 1996.

[Giunchiglia *et al.*, 1993] F. Giunchiglia, L. Serafini, E. Giunchiglia, e M. Frixione. Non-Omniscient Belief as Context-Based Reasoning. In *Proc. of the 13th International Joint Conference on Artificial Intelligence*, pp. 548–554, Chambery, France, 1993. Also IRST-Technical Report 9206-03, IRST, Trento, Italy.

[Giunchiglia *et al.*, 1996] F. Giunchiglia, M. Roveri, e R. Sebastiani. A new method for testing decision procedures in modal and terminological logics. In *Proc. of 1996 International Workshop on Description Logics - DL'96*, Cambridge, MA, USA, November 1996. Also IRST-Technical Report 9609-05.

[Giunchiglia e Giunchiglia, 1996] F. Giunchiglia e E. Giunchiglia. Ideal and Real Belief about Belief: Some intuitions. In *Agents Breaking Away, Proc. MAAMAW'96*, number 1038 in Lecture Notes in Artificial Intelligence, pp. 1–12. Springer Verlag, 1996.

[Giunchiglia e Sebastiani, 1996a] F. Giunchiglia e R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. of the 13th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, New Brunswick, NJ, USA, August 1996. Springer Verlag. Also DIST-Technical Report 96-0037 and IRST-Technical Report 9601-02.

[Giunchiglia e Sebastiani, 1996b] F. Giunchiglia e R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996. Also DIST-Technical Report 9607-08 and IRST-Technical Report 9601-02.

[Giunchiglia e Serafini, 1994] F. Giunchiglia e L. Serafini. Multilanguage hierarchical logics (or: how we can do without modal logics). *Artificial Intelligence*, 65:29–70, 1994. Also IRST-Technical Report 9110-07, IRST, Trento, Italy.

[Hintikka, 1962] J . Hintikka. *Knowledge and Belief.* Cornell University Press, Ithaca, NY, 1962.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, e H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pp. 459–465, 1992.

[Russell, 1956] B. Russell. *Theory of Knowledge.* George Allen and Unwin Ltd., 1956.

[Sebastiani, 1997] R. Sebastiani. *Una nuova classe di procedure di decisione per logiche modali e terminologiche: teoria, implemetazione e testing.* PhD thesis, DIST - Università di Genova, 1997. In Italian.

[Uribe e Stickel, 1994] T. E. Uribe e M. E. Stickel. Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. In *Proc. of the 1st International Conference on Constraints in Computational Logics*, 1994.

[Vardi, 1986] M. Vardi. On epistemic logic and logical omniscience. In *Teoretical Aspects of Reasoning about Knowledge*, pp. 293–305, 1986.

[Vardi, 1989] M. Vardi. On the complexity of epistemic reasoning. In *Proc. of the 4th IEEE Symp. of Logic in Computer Science*, 1989.