università
di genova
facoltà di
ingegneria

dipartimento
informatica
sistemistica
telematica

Una nuova classe di procedure di decisione per logiche modali e terminologiche: teoria, implementazione e testing

Tesi per il conseguimento del titolo di

Dottore di Ricerca in Ingegneria Elettronica ed Informatica

Candidato:

Ing. **Roberto Sebastiani**
DIST - Università di Genova

Relatori:

Ch.mo Prof. **Mauro Di Manzo**
DIST - Università di Genova

Ch.mo Prof. **Fausto Giunchiglia**
DISA - Università di Trento
IRST - Trento

Ringraziamenti

Al termine di questo dottorato, e anche di questi tre anni, ringrazio:

- **Mauro Di Manzo**, per aver reso tutto questo possibile;
- **Fausto**. Per avermi insegnato a "fare scienza"; per avermi sempre seguito; per avermi insegnato "the right attitudes"; per avermi assegnato questa tematica e aver vinto i miei scetticismi iniziali su di essa. Soprattutto, per avermi dato fiducia;
- **Colin**. Per avermi insegnato tutto quello che so su CSP e phase transitions e, soprattutto, per avermi comunicato la giusta "forma mentis" per queste tematiche. Inoltre (insieme a Patricia) per la splendida amicizia a Stanford e nelle occasioni successive. [For teaching me all I know about CSP and phase transitions and, most important, for giving me the "right attitude" towards these topics. Moreover (together with Patricia) for the dear friendship in Stanford and later.]
- **Enrico G.**, per il "la" iniziale, e altre note;
- mio fratello **Fabrizio**, per i fondamentali consigli;
- Tutte le altre persone con cui ho "interagito scientificamente" in questi anni: **Marco Cadoli**, **Enrico Franconi**, **Fabio Massacci**, **David McAllester**, **Bart Selman**, **Carolyn Talcott**, **Luca Viganò**, **Toby Walsh**, **Richard Weyhrauch**, ... Per tutto quello che mi hanno insegnato, direttamente o indirettamente. [All people with whom I have interacted scientifically: ... For all they have tought me;]
- tutti i membri `MRG@DIST`: **Adolfo**, **Alessandro A.**, **Alessandro C.**, **Alessandro M.**, **Claudio**, **Enrico**, **Lele**, **Luca V.**, **Marco**, **Peck**, **Silvio**, **Simone**, **Tac** ..., tutti i membri `MRG@DISA`: **Chiara**, **Massimo**, **Paolo** ..., e tutti i membri `MRG@IRST` (anche "ex"): **Alessandra/o Z.**, **Alessandro C.**, **Camilla**, **Dimitri I** e **II**, **Fausto**, **Gianluca**, **Giovanna**, **Luca S.**, **Luciano**, **Manuela**, **Marco M.**, **Marisa**, **Michela**, **Paolo T.**, **Piergiorgio**, **Toby**... e tutti quelli che ho dimenticato di menzionare esplicitamente. Perchè "*nella scienza si impara soprattutto per osmosi*" [Giunchiglia 90, personal communication] ... e anche nella vita di tutti i giorni;

- le "ragazze" delle varie segreterie DIST: **Alessandra**, **Anna**, **Barbara**, **Bice**, **Daniela**, **Lorella**, **Marina I e II**, **Raffaella**, **Roberta**, **Silvia**, **Tiziana** ... Per avermi condotto con pazienza materna lungo i labirinti a me oscuri della burocrazia. E per i venti minuti quotidiani di buon umore davanti ad un caffè;

- ancora i membri `MRG@DIST`, insieme ai "vicini" del `Biolab`, per avere reso piacevoli e vivaci le ore in laboratorio;

- **Silvio**, perchè due anni di convivenza non sono acqua;

- **Luca S.**, **Luciano** e **Peck**, per la convivenza a Stanford;

- tutti gli altri amici di Genova: **Carlo** & **Margherita**, **Claudia**, **Clementina**, **Cristina**, **Manuela**, **Roberto**, **Simone**, **Stefania**; gli amici della pallavolo: **Carla** & **Francesco**, **Marco**, **Simona**, **Vittoria**, ...; gli amici del CUS: **Alberto**, **Maria**, ...; gli amici scialpinisti della "*Focolaccia*" di Lucca. Per aver contribuito a rendere più piacevoli questi tre anni;

- tutti gli altri amici di Trento e di Padova: **Alessia** e le sue sorelle, **Anna** e **Aldo**, **Betta** e C., **Betti**, **Carlo** & **Paola D.**, **Cristina P.**, **Jack** & **Robi**, **Laura** & **Paolo B.**, **Laura**, **Marzia**, **Max**, **Monica**, **Silvano**, ..., e tutto il clan di "*Icaro*" e della "*Rete*". Per avermi dato occasione di avere nostalgia;

- Tutti gli altri amici, conosciuti chissà dove e ora dispersi nel mondo: **Alita** & **Jeff** e **Karina**, **Colin** & **Patricia**, **Franz**, **Julija**, **Patty**, **Yvonne**. Per aver mantenuto i contatti con me, nonostante la lontananza. [All other friends spread everywhere in the world: ... For keeping in touch, despite the distances.]

- **Paola R.**, a cui non devo solo ringraziamenti, ma anche scuse;

- tutti coloro che ho dimenticato di menzionare esplicitamente;

- mia madre, mio fratello e sorella, zie/zii, cognati/e ed aspiranti tali: per tutto;

- **Simona**, *"dulcis in fundo"*.

Genova, 27 Febbraio 1997.

Indice

Capitolo 1

Introduzione

"... le idee sono come i figli: non basta concepirle, è necessario farle nascere e crescere ... "
[D. Picouly, "Il campo di nessuno"]

1.1 Motivazioni

In generale, come "problema di decisione" per una logica $\mathcal{L}$ intendiamo il problema di determinare se, data un'espressione (formula, concetto) in $\mathcal{L}$, esista un'interpretazione in $\mathcal{L}$ che la soddisfi. Ad esempio, in logica proposizionale [1] il problema consiste nel determinare se esista un assegnamento di valori di verità ai singoli atomi di una formula φ che renda vera φ. Tale definizione non è restrittiva, in quanto tutti i problemi di decisione per le logiche considerate — come refutazione, sussunzione, determinazione di validità, (in)consistenza e consequenzialità, ecc. — sono riconducibili con costo lineare al problema della soddisfacibilità (si vedano, ad esempio, [Kleene, 1952; Fitting, 1983; Donini *et al.*, 1992]).

La complessità computazionale del problema di decisione nelle logiche da noi considerate varia da NP-compete a PSPACE-complete (in qualche caso fino e EXPTIME-complete) [Ladner, 1977; Halpern

[1] È spesso d'uso, soprattutto in ambienti ingegneristici, chiamare *logica booleana* la logica proposizionale.

e Moses, 1992; Donini *et al.*, 1992] [2]. Si tratta quindi di problemi di elevatissima complessità computazionale, che mettono a dura prova l'efficienza degli algoritmi. Si noti, per confronto, che la soddisfacibilità nella logica proposizionale è "solo" NP-complete [Cook, 1971].

In generale, rifacendoci alla terminologia corrente, parleremo di "ragionamento automatico" intendendo tutte le varie operazioni su espressioni logiche, quali deduzione, inferenza, refutazione, sussunzione, valutazione, verifica di validità/soddisfacibilità, model checking, ecc. Il ragionamento automatico in logica proposizionale e del primo ordine è stato lungamente oggetto di studio, a causa delle applicazioni ai sistemi digitali ed alle basi di dati. Di conseguenza esistono un gran numero di sistemi di ragionamento automatico molto efficienti per tali logiche (ad esempio, [Bryant, 1992; Crawford e Auton, 1996; McCune, 1990]).

Per quanto riguarda lo studio di metodi di ragionamento automatico in altre logiche (ad esempio, logiche modali, terminologiche, dinamiche, temporali), la ricerca di procedure di ragionamento efficienti rappresenta invece un problema ancora aperto. Al riguardo, nella letteratura distinguiamo "storicamente" due fasi. In una prima fase, protratta fino a tempi relativamente recenti, tale studio era relegato a lavori in logica pura, intelligenza artificiale e rappresentazione della conoscenza. In questo senso, la ricerca si era dedicata prevalentemente allo studio di formalismi *generali*, trascurando lo studio dell'efficienza degli algoritmi che ne derivavano [3]. L'ideazione di un gran numero di formalismi/algoritmi basati su tableau [4] [Smullyan, 1968] ("tableau-based" da ora in poi) è riconducibile a questa fase. In una fase successiva, un notevole impulso alla ricerca nel settore è stato dato dallo sviluppo dei metodi

[2] Ricordiamo la gerarchia delle classi di complessità (si veda ad esempio [Garey e Johnson, 1979]): $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$, con il vincolo $P \subset EXPTIME$.

[3] Gli algoritmi di ragionamento automatico sono tipicamente basati su opportuni formalismi logici, secondo il paradigma *"algorithm = logic + control"* [Kowalski, 1979].

[4] Si veda il Capitolo 3.

formali per la verifica di sistemi e protocolli, basati su logiche modali e temporali (ad esempio, [Boyer e Moore, 1983; Camilleri *et al.*, 1987; Halpern e Fagin, 1985; Hunt, 1989; Owre *et al.*, 1995; Manna e Pnueli, 1995]) e dallo sviluppo di basi di conoscenza basate su logiche terminologiche (ad esempio, [Patel-Schneider *et al.*, 1991; Patel-Schneider, 1996]). Questo ha creato la necessità di nuovi efficienti algoritmi per il ragionamento automatico con tali logiche. Nel Capitolo 3 presentiamo una descrizione sommaria dello stato dell'arte per quanto riguarda le logiche modali e terminologiche. [Manna e Pnueli, 1995] è un buon riferimento per quanto riguarda le logiche temporali [5].

1.2 Finalità

Scopo del lavoro è quello di descrivere una nuova tecnica per sviluppare procedure di decisione *efficienti* per logiche modali e terminologiche, e di presentare una metodologia generale per estendere questa tecnica ad altre logiche. Il nostro approccio, che chiamiamo "SAT-based" [6], è fondato sulle seguenti intuizioni.

1. Il ragionamento automatico nelle logiche modali e terminologiche (ma anche in logiche dinamiche e temporali) può essere visto come una composizione appropriata di distinte fasi di ragionamento automatico proposizionale all'interno di teorie/modelli distinte (si vedano, ad esempio, [Giunchiglia e Serafini, 1994; Giunchiglia *et al.*, 1993]).

2. Tale idea, anche se mai descritta, è implicitamente presente anche nei metodi tableau-based (si veda, ad esempio, [Fit-

[5]Amir Pnueli è stato insignito nel 1996 del *Turing Award* per i suoi studi sulle logiche temporali e sui sistemi di verifica basati su di esse.

[6]"SAT" è l'abbreviazione comunemente usata per "propositional *SAT*-isfiability", cioè soddisfacibilità proposizionale. In generale per procedure SAT si intendono quelle procedure usate normalmente nelle comunità SAT, che presentiamo in dettaglio nella Sezione 3.1. In tale senso, chiameremo "DPLL-based" le procedure basate sull'algoritmo DPLL, "OBDD-based" le procedure basate su OBDD, e così via.

ting, 1983]). Il ragionamento proposizionale, in questo caso, è effettuato mediante l'applicazione delle regole tableau proposizionali, comuni a tali metodi.

3. Una delle grosse cause di inefficienza dei metodi tableau-based è proprio l'uso di tali regole proposizionali. Questa inefficienza si propaga e si espande con la profondità modale (terminologica) delle formule analizzate. Tale espansione è esponenziale nel caso peggiore.

4. La componente proposizionale del ragionamento modale/terminologico può essere effettuato in modo notevolmente più efficiente usando un decisore proposizionale allo stato dell'arte, opportunamente "inserito" nella procedura. Ad esempio, è possibile utilizzare la procedura Davis-Putnam-Longemann-Loveland (DPLL d'ora in poi) [Davis e Putnam, 1960; Davis *et al.*, 1962], OBDD [Bryant, 1992], KE [D'Agostino e Mondadori, 1994], o perfino decisori non completi come GSAT [Selman *et al.*, 1992].

1.3 Contenuti

È necessario premettere che alcuni contenuti di questa tesi sono già stati descritti in [Giunchiglia e Sebastiani, 1996a; Giunchiglia e Sebastiani, 1996b; Giunchiglia e Sebastiani, 1996c; Giunchiglia *et al.*, 1996; Giunchiglia *et al.*, 1997]. Inoltre alcuni contenuti sono correlati a quanto descritto in [Sebastiani, 1994; Bundy *et al.*, 1996a; Bundy *et al.*, 1996b].

1.3.1 Teoria, implementazione e testing

Il lavoro svolto può essere suddiviso in tre parti:

(i) una parte *teorica*,
(ii) una parte *implementativa*,
(iii) una parte *sperimentale*.

Nella parte teorica, abbiamo elaborato i fondamenti teorici su cui si basa la costruzione di una nuova classe di procedure di decisione. Abbiamo introdotto le nuove definizioni di *atomo*, *assegnamento di verità* e *soddisfacibilità proposizionale* per logiche diverse dalla logica proposizionale, illustrandone le proprietà significative; abbiamo proposto e dimostrato una serie di teoremi che legano la soddisfacibilità di formule con la loro soddisfacibilità proposizionale, e che legano la soddisfacibilità di formule con quella dei loro assegnamenti; abbiamo descritto un sistema formale generalizzato per le logiche modali normali $\mathcal{N}$ e dimostrato la sua correttezza e completezza, presentando un algoritmo di decisione generalizzato per tali logiche; abbiamo infine mostrato come estendere (banalmente nel caso delle logiche terminologiche) tale metodologia alle logiche per cui esista un formalismo tableau-based.

Per verificare empiricamente la bontà dell'approccio, nell'impossibilità pratica di implementare una procedura e condurre un test estensivo per tutte le logiche per cui ciò è possibile, abbiamo scelto la logica modale normale K(m) come caso di studio.

Nella parte implementativa, restringendoci a K(m), abbiamo ideato e sviluppato KSAT, un algoritmo basato su un'implementazione del decisore proposizionale DPLL, partendo da una versione di base ed introducendo incrementalmente alcune ottimizzazioni. KSAT è implementato in Common LISP [7]. Abbiamo inoltre ideato ed implementato una nuova procedura di test, chiamata $3\mathrm{CNF}_{\Box m}$, che estende al caso modale una procedura di test standard per decisori proposizionali, pur mantenendone tutte le prerogative. La metodologia $3\mathrm{CNF}_{\Box m}$ è estensibile ad un gran numero di logiche.

Infine, nella parte sperimentale, descriviamo e commentiamo i risultati di vari test empirici estensivi effettuati su KSAT e procedure tableau-based, quali TABLEAU e KRIS [8]. In particolare:

[7]Abbiamo anche portato a termine un'implementazione di KSAT in C++, costruita sull'implementazione di DPLL di Max Böhm [Buro e Buning, 1992]. Tale procedura, realizzata in collaborazione con Mauro Chimenti, non è stata ancora sufficientemente provata fino ad ora (febbraio 1997).

[8]Le varie versioni di KSAT, il codice del test e tutti i risultati presentati sono disponibili via anonymous FTP a `ftp.mrg.dist.unige.it`, in

(*a*) descriviamo i risultati di alcuni confronti empirici estensivi con procedure tableau-based, sia implementazioni dirette (TABLEAU) che sistemi allo stato dell'arte (KRIS). Da tali test emerge come le prestazioni di KSAT superino di vari ordini di grandezza quelle delle altre procedure;

(*b*) mostriamo che tali risultati non sono casuali, ma che le procedure DPLL-based da noi ideate sono intrinsecamente più efficienti di quelle tableau-based. In particolare, evidenziamo due intrinseche cause di inefficienza all'interno del meccanismo di ragionamento proposizionale delle procedure tableau-based, non presenti nelle procedure SAT-based;

(*c*) infine, analizziamo dal punto di vista qualitativo i risultati ottenuti, proponiamo alcune osservazioni sulle caratteristiche delle formule K(m) al variare di vari parametri. In particolare, gli esperimenti suggeriscono l'esistenza di un fenomeno di "transizione di fase" per K(m). Questo fenomeno, comune a molti problemi NP-hard, non era mai stato rivelato precedentemente nel campo della decisione in logiche modali.

1.3.2 Un caso di studio: K(m)

La scelta di K(m) come caso di studio è dovuta ai seguenti motivi.

In primo luogo, K(m) è la logica di base comune a tutte i tipi di logiche da noi analizzate. In particolare

- le **logiche modali normali** $\mathcal{N}$ sono costruite su K(m) utilizzando lo stesso linguaggio ed imponendo restrizioni sulla semantica (o, secondo un approccio assiomatico, aggiungendo assiomi);

- le **logiche terminologiche** sono costruite su K(m) [9] aggiungendo alcuni nuovi operatori e costrutti sintattici;

`pub/mrg-systems/ksat/`. TABLEAU è disponibile a `ftp.mrg.dist.unige.it` in `pub/mrg-systems/tableau/`. KRIS è disponibile a `ftp.dfki.uni-sb.de` in `pub/tacos/KRIS/`.

[9] Più precisamente da $\mathcal{ALC}$, una variante notazionale di K(m). Si veda Sezione 2.3.

- le **logiche dinamiche** sono costruite su K(m) aggiungendo alcuni nuovi operatori e costrutti sintattici;
- **CTL** (computational tree logic), la più importante logica temporale utilizzata nell'area dei metodi formali di verifica, è costruita su K(1) aggiungendo alcuni nuovi operatori.

È importante osservare che tutte le logiche considerate sono costruite su K(m) aggiungendo costrutti sintattici o vincoli semantici, e che questi *non coinvolgono alcuna componente proposizionale*. Come illustrato nel Capitolo 4, le nostre procedure differiscono dalle procedure tableau-based proprio per tale componente. Di conseguenza, l'analisi teorica ed empirica di K(m) risulta significativa per tutte le logiche sopra citate.

In secondo luogo, tra le logiche modali e terminologiche, la struttura di K(m) è quella che si presta meglio ad essere analizzata dal punto di vista computazionale, in quanto permette di individuare gli effetti di variazioni di singoli parametri in modo più evidente (si veda il Capitolo 9).

Infine, il problema della decisione in K(m) è PSPACE-complete [Ladner, 1977; Halpern e Moses, 1992], e quindi di elevatissima complessità computazionale. Di conseguenza, nonostante la relativa semplicità sintattica, K(m) costituisce un banco di prova molto significativo per una nuova classe di procedure di decisione.

1.4 Struttura della tesi

La tesi è organizzata come segue.

- **Capitolo 1: Introduzione.** Introduce finalità e motivazioni della tesi, e ne descrive sommariamente i contenuti.

Parte I: Preliminari. Presenta una serie di nozioni preliminari, ben note in letteratura, necessarie alla comprensione della tesi.

- **Capitolo 2: Logica.** Illustra una serie di nozioni sulla logica proposizionale, sulle logiche modali e sulle logiche terminologiche.

- **Capitolo 3: Algoritmi di decisione.** Descrive lo stato dell'arte delle procedure di decisione per logiche proposizionali, modali e terminologiche.

Parte II: Teoria. Presenta i fondamenti teorici su cui si basa la costruzione della nuova classe di procedure di decisione.

- **Capitolo 4: Un nuovo approccio.** Presenta il nuovo approccio SAT-based alla creazione di formalismi ed algoritmi di decisione. Introduce una serie di nuove definizioni; illustra un formalismo e conseguente algoritmo generalizzato; dimostra una serie di teoremi di correttezza e completezza; illustra un metodo generale per estendere l'approccio ad altre logiche.

Parte III: Implementazione. Descrive in dettaglio le procedure implementate.

- **Capitolo 5: Una nuova classe di procedure di decisione.** Descrive KSAT, una procedura DPLL-based per K(m), dalla versione di base alle sue successive evoluzioni.

- **Capitolo 6: Un nuova procedura di test.** Descrive la nuova procedura di test utilizzata, ne illustra le prerogative e mostra come possa essere estesa ad altre logiche.

Parte IV: Testing. Descrive e commenta i risultati di test empirici estensivi effettuati sulle procedure implementate.

- **Capitolo 7: Procedure DPLL-based contro tableau-based.** Descrive i risultati di alcuni confronti empirici estensivi con procedure tableau-based.

- **Capitolo 8: Spiegazione dei risultati.** Fornisce una spiegazione del divario di prestazioni evidenziato nel Capitolo 7, mostrando due motivazioni per cui gli algoritmi SAT-based (DPLL-based) sono intrinsecamente più efficienti di quelli tableau-based.

- **Capitolo 9: Un'analisi empirica di K(m).** Analizza i risultati empirici del Capitolo 7, proponendo alcune osservazioni qualitative. In particolare, suggerisce l'esistenza di un fenomeno di "transizione di fase" per K(m).

Infine

- **Capitolo 10: Conclusioni e lavoro futuro.** Presenta alcune considerazioni conclusive e descrive possibili direzioni per sviluppi futuri.

Parte I

Preliminari

Capitolo 2

Logica

"Leibnitz, Boole and Gödel worked with logic.
I work with logic.
Thus I'm Leibnitz, Boole and Gödel."
[Anonimo]

In questo capitolo riassumiamo alcuni concetti fondamentali di logica, e introduciamo la terminologia e la notazione usata in tutto il lavoro. In particolare, nella Sezione 2.1 analizziamo la logica proposizionale, nella Sezione 2.2 le logiche modali normali e nella Sezione 2.3 le principali logiche terminologiche.

2.1 La logica proposizionale

Presentiamo qui alcune nozioni di base sulla logica proposizionale Si veda qualsiasi testo di logica matematica, per esempio, [Kleene, 1952; Kleene, 1967; Borga, 1984] per dettagli ed approfondimenti.

2.1.1 Sintassi

Dato un insieme non vuoto di lettere proposizionali $\mathcal{A} = \{A_1, A_2, \ldots\}$ e l'insieme delle costanti booleane $\{T, F\}$ ("vero", "falso") [1], definiamo il linguaggio proposizionale Λ come il più piccolo insieme di

[1] Con un piccolo abuso di notazione, useremo $\{T, F\}$ per indicare sia le *costanti* del linguaggio, sia per indicare i *valori semantici* vero e falso. Invece, nei capitoli successivi useremo $True, False$ per indicare i valori booleani restituiti da una procedura.

formule contenente $\mathcal{A}$ e $\{T, F\}$ chiuso sotto l'insieme dei connettivi proposizionali $\{\neg, \wedge\}$. Notazionalmente, useremo le lettere greche $\alpha, \beta, \varphi, \psi$ per rappresentare formule in Λ. Useremo inoltre abbreviazioni standard quali "$\varphi_1 \vee \varphi_2$" per "$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \supset \varphi_2$" per "$\neg(\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \equiv \varphi_2$" per "$\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\varphi_2 \wedge \neg\varphi_1)$". Chiameremo *dimensione* di φ, scritto $|\varphi|$, il numero di sottoformule di φ. Chiameremo *atomo* ogni proposizione semplice $A_i \in \mathcal{A}$. $Atoms(\varphi)$ è l'insieme degli atomi di φ. Chiameremo *letterale* un atomo A_i o la sua negazione $\neg A_i$. Se $l = \neg A_i$, per "$\neg l$" intenderemo convenzionalmente A_i anziché $\neg\neg A_i$.

Una formula proposizionale è detta essere in *forma normale negativa* (*NNF*) se è scritta utilizzando solo $\wedge, \vee$ e letterali. È sempre possibile convertire ogni formula proposizionale φ in forma NNF della stessa lunghezza. La conversione (si veda, per esempio, [Kleene, 1952; Borga, 1984]) si basa sull'applicazione ricorsiva delle seguenti regole:

$$\neg(\varphi_1 \vee \varphi_2) \quad \Longrightarrow \quad (\neg\varphi_1 \wedge \neg\varphi_3),$$

$$\neg(\varphi_1 \wedge \varphi_2) \quad \Longrightarrow \quad (\neg\varphi_1 \vee \neg\varphi_3)$$

(e regole equivalenti per "$\neg \supset$" e "$\neg \equiv$'), che "spostano verso il basso" le negazioni.

Una formula proposizionale è detta essere in *forma normale congiuntiva* (*CNF*) se è scritta nella forma:

$$\bigwedge_i \bigvee_j l_{ij},$$

dove i vari l_{ij} sono letterali. Ogni congiunto $\bigvee_j l_{ij}$ è detto anche *clausola*. Una clausola è detta *unitaria* se è costituita da un unico letterale. Una formula è detta essere in kCNF, $k = 2$,3 ,4,..., se ogni clausola ha esattamente k letterali. Un'occorrenza di un letterale l in una clausola C è detta *positiva* se $C = \ldots \vee l \vee \ldots$, *negativa* se $C = \ldots \vee \neg l \vee \ldots$. È sempre possibile convertire ogni formula proposizionale φ in forma CNF. Vi sono due metodi di conversione alternativi. Il metodo classico (si veda, per esempio, [Kleene, 1952; Borga, 1984]) si basa sull'applicazione ricorsiva delle

delle regole di conversione NNF e delle regole di DeMorgan:

$$((\varphi_1 \vee \varphi_2) \wedge \varphi_3) \quad \Longrightarrow \quad ((\varphi_1 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_3))$$

$$((\varphi_1 \wedge \varphi_2) \vee \varphi_3) \quad \Longrightarrow \quad ((\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)).$$

Il secondo si basa sull'introduzione di nuovi atomi, ciascuno dei quali indica una particolare sottoformula di φ [Plaisted e Greenbaum, 1986; de la Tour, 1990]. Il primo metodo può causare la crescita esponenziale della dimensione della formula. Il secondo causa l'incremento del numero degli atomi proposizionali.

2.1.2 Teoria assiomatica

Presentiamo dapprima la logica proposizionale secondo una caratterizzazione assiomatica (si vedano, per esempio, [Kleene, 1952; Borga, 1984]). Per altri calcoli si vedano, ad esempio, [Gentzen, 1969; Prawitz, 1965; Smullyan, 1968].

Supponiamo per semplicità di riscrivere tutte le nostre formule utilizzando solamente $\{\neg, \supset\}$. Consideriamo gli schemi di assiomi:

$$\begin{array}{ll} I. & \alpha \supset (\beta \supset \alpha), \\ II. & (\alpha \supset (\beta \supset \gamma)) \supset ((\alpha \supset \beta) \supset (\alpha \supset \gamma)), \\ III. & (\neg\alpha \supset \beta) \supset ((\neg\alpha \supset \neg\beta) \supset \alpha), \end{array}$$

e la seguente regola di inferenza:

$$\frac{\alpha \quad \alpha \supset \beta}{\beta} \ (\textit{modus ponens})$$

dove α, β e γ sono formule qualsiasi. Una formula φ è *deducibile* da un insieme di formule Γ (scritto $\Gamma \vdash \varphi$) se può essere ricavata applicando ripetutamente il modus ponens alle formule $\varphi_i \in \Gamma$ e ad istanze degli assiomi. φ è un *teorema* (scritto $\vdash \varphi$) se è deducibile dall'insieme vuoto. Si verifica che $\varphi_1 \vdash \varphi_2$ se e solo se $\vdash \varphi_1 \supset \varphi_2$, e quindi che $\vdash \varphi_1 \equiv \varphi_2$ se e solo se $\varphi_1 \vdash \varphi_2$ e $\varphi_2 \vdash \varphi_1$. Una formula φ_1 è *inconsistente* se, per ogni formula φ_2, $\varphi_1 \vdash \varphi_2$ e $\varphi_1 \vdash \neg\varphi_2$, consistente altrimenti. Si verifica che $\vdash \varphi$ se e solo se $\neg\varphi$ è inconsistente.

2.1.3 Semantica

Un *assegnamento di verità* μ per una formula $\varphi \in \Lambda$ è un assegnamento di valori di verità agli atomi di φ:

$$\begin{aligned} \mu \;\; = \{ \;\; & A_1 := T, \ldots, A_R := T, \\ & A_{R+1} := F, \ldots, A_S := F\}, \end{aligned}$$

dove ":=" è l'operatore di assegnazione (se $l = \neg A_i$, diremmo "$l := T$" ["$l := F$"] intendendo "$A_i := F$" [$A_i := T$]). Chiameremo *parziale* un assegnamento di verità μ per φ che assegni valori di verità solo ad un sottoinsieme stretto di $Atoms(\varphi)$. In generale un assegnamento di verità per $\varphi \in \Lambda$ può assegnare valori anche ad atomi non appartenenti ad $Atoms(\varphi)$. Tali valori sono *indifferenti* per φ. Notazionalmente, chiamiamo $\parallel \mu \parallel$ il numero di elementi dell'insieme μ. Se $\mu_2 \subseteq \mu_1$, diciamo che μ_1 *estende* μ_2 e che μ_2 *sussume* μ_1. Qualora questo non causi ambiguità, useremo spesso "assegnamento" al posto di "assegnamento di verità". Useremo le lettere greche μ, η per rappresentare assegnamenti. Si noti anche che la costante T rappresenta un assegnamento vuoto.

Un assegnamento μ per $\varphi \in \Lambda$ *soddisfa* φ, scritto "$\mu \models \varphi$", se e solo se rende φ vera, cioè, se per tutte le sub-formule φ_1, φ_2 di φ, si ha:

$$\begin{array}{lcl} \mu \models \varphi_1, \; \varphi_1 \in Atoms(\varphi) & \Longleftrightarrow & \{\varphi_1 := T\} \subseteq \mu; \\ \mu \models \neg\varphi_1 & \Longleftrightarrow & \mu \not\models \varphi_1; \\ \mu \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & \mu \models \varphi_1 \; e \; \mu \models \varphi_2. \end{array}$$

Un assegnamento *parziale* μ soddisfa φ se e solo se tutti gli assegnamenti per φ che estendono μ soddisfano φ. D'ora in poi, se non esplicitamente specificato, con "assegnamento" intenderemo anche assegnamenti parziali. Una formula $\varphi \in \Lambda$ è *soddisfacibile* se e solo se esiste un assegnamento μ tale che $\mu \models \varphi$, *insoddisfacibile* altrimenti. Per ogni coppia di formule φ_1 e φ_2, diciamo che $\varphi_1 \models \varphi_2$ se e solo se $\mu \models \varphi_2$ per ogni μ tale che $\mu \models \varphi_1$. È facile verificare che $\varphi_1 \models \varphi_2$ se e solo se $\models \varphi_1 \supset \varphi_2$, e quindi che $\models \varphi_1 \equiv \varphi_2$ se e solo se $\varphi_1 \models \varphi_2$ e $\varphi_2 \models \varphi_1$. Diciamo inoltre che $\models \varphi$ (φ è *valida*) se e solo se $\mu \models \varphi$ per ogni possibile assegnamento μ per φ. Si verifica che $\models \varphi$ se e solo se $\neg\varphi$ è insoddisfacibile.

Vale il seguente teorema di correttezza e completezza:

$$\vdash \varphi \quad \Longleftrightarrow \quad \models \varphi.$$

Di conseguenza, φ è consistente se e solo se è soddisfacibile.

Il problema di determinare la soddisfacibilità di una formula proposizionale in forma CNF è normalmente chiamato SAT. Il problema di determinare la soddisfacibilità di una formula proposizionale, NCF o non-CNF, è notoriamente decidibile e NP-complete [Cook, 1971; Garey e Johnson, 1979]. Nella Sezione 3.1 presentiamo i principali algoritmi di decisione per logica proposizionale riportati in letteratura.

2.2 Le logiche modali normali

Presentiamo qui alcune nozioni di base sulle logiche modali normali Si vedano, per esempio, [Chellas, 1980; Hughes e Cresswell, 1972; Fagin *et al.*, 1995] per dettagli ed approfondimenti.

2.2.1 Sintassi

Dato un insieme non vuoto di lettere proposizionali $\mathcal{A} = \{A_1, A_2, \ldots\}$, l'insieme delle costanti booleane $\{T, F\}$ ("vero", "falso"), e un insieme di m operatori modali $\mathcal{B} = \{\Box_1, \ldots, \Box_m\}$, definiamo il linguaggio $\Lambda_{\Box m}$ come il più piccolo insieme di formule contenente $\mathcal{A}$ e $\{T, F\}$ chiuso sotto l'insieme dei connettivi proposizionali $\{\neg, \wedge\}$ e l'insieme degli operatori modali $\mathcal{B}$. Si noti che i simboli "$\diamond_r$", usati normalmente come abbreviazioni di "$\neg\Box_r\neg$", non fanno parte del linguaggio; la motivazione per questa scelta sarà discussa in seguito. Notazionalmente, useremo le lettere greche $\alpha, \beta, \varphi, \psi$ per rappresentare formule in $\Lambda_{\Box m}$. Useremo le abbreviazioni standard "$\varphi_1 \vee \varphi_2$" per "$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \supset \varphi_2$" per "$\neg(\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \equiv \varphi_2$" per "$\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\varphi_2 \wedge \neg\varphi_1)$". Chiameremo *dimensione* di φ, scritto $|\varphi|$, il numero di sottoformule di φ. Chiameremo *profondità* di φ, scritto *depth(φ)*, il massimo numero di operatori modali annidati in φ.

2.2.2 Teoria assiomatica

Presentiamo dapprima le varie logiche modali normali secondo una caratterizzazione assiomatica (si vedano, per esempio, [Chellas, 1980; Hughes e Cresswell, 1972; Halpern e Moses, 1992]). Per altri calcoli si vedano, ad esempio, [Bull e Segerberg, 1984; Fitting, 1983]. Come in precedenza, supponiamo di riscrivere tutte le nostre formule utilizzando solamente $\{\neg, \supset\}$.

Siano dati i seguenti schemi di assiomi:

$$\begin{array}{ll}
I. & \alpha \supset (\beta \supset \alpha), \\
II. & (\alpha \supset (\beta \supset \gamma)) \supset ((\alpha \supset \beta) \supset (\alpha \supset \gamma)), \\
III. & (\neg\alpha \supset \beta) \supset ((\neg\alpha \supset \neg\beta) \supset \alpha), \\
 & \\
K. & \Box_r\alpha \supset (\Box_r(\alpha \supset \beta) \supset \Box_r\beta), \\
B. & \alpha \supset \Box_r\neg\Box_r\neg\alpha, \\
D. & \Box_r\alpha \supset \neg\Box_r\neg\alpha, \\
T. & \Box_r\alpha \supset \alpha, \\
4. & \Box_r\alpha \supset \Box_r\Box_r\alpha, \\
5. & \neg\Box_r\alpha \supset \Box_r\neg\Box_r\alpha,
\end{array}$$

e le seguenti regole di inferenza:

$$\frac{\alpha \quad \alpha \supset \beta}{\beta} \text{ (modus ponens)},$$

$$\frac{\alpha}{\Box_r\alpha} \text{ (necessitazione)},$$

dove α, β e γ sono formule qualsiasi in $\Lambda_{\Box m}$ e $\Box_r$ è un generico operatore modale in $\mathcal{B}$. Un *sistema formale assiomatico modale normale* $\mathcal{L}$ (o "*logica modale normale*") su $\Lambda_{\Box m}$ è dato dalla terna composta

(i) dal linguaggio $\Lambda_{\Box m}$,
(ii) dall'insieme degli assiomi $\{I, II, III, K\} \cup \mathcal{S}$, dove $\mathcal{S}$ è un sottoinsieme stretto di $\{B, D, T, 4, 5\}$,
(iii) dalla coppia di regole di inferenza descritte sopra.

Chiameremo $\mathcal{N}$ l'insieme delle logiche modali normali. Data una logica normale $\mathcal{L} \in \mathcal{N}$, una formula φ è *deducibile in* $\mathcal{L}$ (*$\mathcal{L}$-deducibile*)

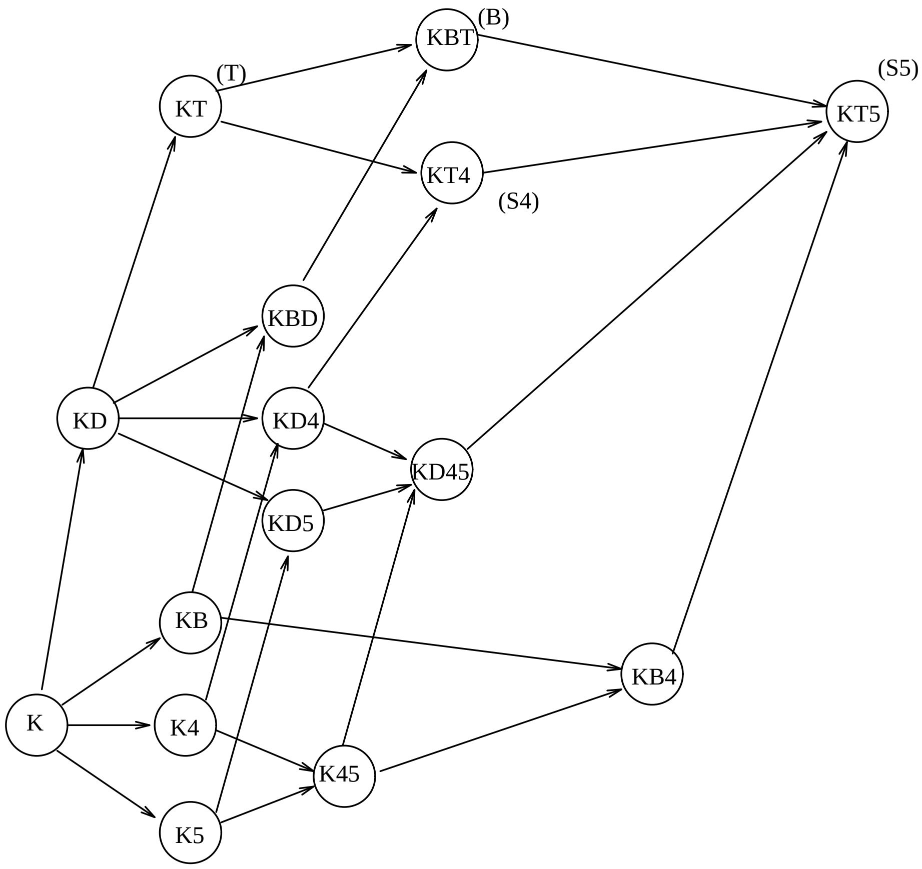


Figura 2.1: La gerarchia delle logiche modali normali $\mathcal{N}$, da K(m) a S5(m). "$\mathcal{L}_1 \longrightarrow \mathcal{L}_2$" significa: "$\vdash_{\mathcal{L}_1} \varphi \Longrightarrow \vdash_{\mathcal{L}_2} \varphi$".

da un insieme di formule Γ (scritto $\Gamma \vdash_{\mathcal{L}} \varphi$) se e solo se può essere ricavata applicando ripetutamente le due regole sopra alle formule $\varphi_i \in \Gamma$ e ad istanze degli assiomi di $\mathcal{L}$. φ è un $\mathcal{L}$-*teorema* in $\mathcal{L}$ (scritto $\vdash_{\mathcal{L}} \varphi$) se è deducibile dall'insieme vuoto. Si verifica che $\varphi_1 \vdash_{\mathcal{L}} \varphi_2$ se e solo se $\vdash_{\mathcal{L}} \varphi_1 \supset \varphi_2$, e quindi che $\vdash_{\mathcal{L}} \varphi_1 \equiv \varphi_2$ se e solo se $\varphi_1 \vdash_{\mathcal{L}} \varphi_2$ e $\varphi_2 \vdash_{\mathcal{L}} \varphi_1$. Una formula φ_1 è *inconsistente in* L ($\mathcal{L}$-*inconsistente*) se, per ogni formula φ_2, $\varphi_1 \vdash_{\mathcal{L}} \varphi_2$ e $\varphi_1 \vdash_{\mathcal{L}} \neg\varphi_2$, consistente in $\mathcal{L}$ ($\mathcal{L}$-*consistente*) altrimenti. Si verifica che $\vdash_{\mathcal{L}} \varphi$ se e solo se $\neg\varphi$ è $\mathcal{L}$-inconsistente.

$\mathcal{N}$ è organizzato nella struttura gerarchica descritta in Figura 2.1, secondo l'ordinamento parziale "$\longrightarrow$": $\mathcal{L}_1 \longrightarrow \mathcal{L}_2$ se e solo se $\vdash_{\mathcal{L}_1} \varphi \Longrightarrow \vdash_{\mathcal{L}_2} \varphi$, per ogni φ. Ogni logica in $\mathcal{N}$ prende il nome dall'insieme $\{K\} \cup \mathcal{S}$. Ad esempio, KD45(m) è dato da $\{I, II, III, K, D, 4, 5\}$. Nel caso $m = 1$, la scritta "(1)" e il pedice "$_1$" degli operatori modali vengono talvolta omessi. Per migliore leggibilità ometteremo spesso la scritta "(m)". Inoltre, secondo prassi comune, scriveremo "T" per "KT", "B" per "KBT", "S4" per "KT4", "S5" per "KT5". Si noti che, nonostante vi siano $2^5 = 32$ possibili combinazioni, le logiche in $\mathcal{N}$ sono in realtà solo 15, in quanto molte combinazioni sono equivalenti (si veda Tabella 2.1, parte sinistra, per un riassunto delle equivalenze).

2.2.3 Semantica

Data una logica $\mathcal{L} \in \mathcal{N}$, una *struttura di Kripke per* $\mathcal{L}$ ($\mathcal{L}$*-struttura di Kripke*) è definita come una tupla $M = <\mathcal{U}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_m>$, dove

(i) $\mathcal{U}$ è un insieme di stati,

(ii) π è una funzione $\pi : \mathcal{A} \times \mathcal{U} \longmapsto \{T, F\}$,

(iii) ogni $\mathcal{R}_r$ è una relazione binaria sugli stati di $\mathcal{U}$, che verifica le proprietà relative a $\mathcal{L}$ riportate nelle Tabelle 2.1 e 2.2.

La tupla $<\mathcal{U}, \mathcal{R}_1, \ldots, \mathcal{R}_m>$ viene detta $\mathcal{L}$-*frame* di M. Chiamiamo inoltre una $\mathcal{L}$-*situazione* ogni coppia M, u tale che $u \in \mathcal{U}$. La relazione binaria $\models_{\mathcal{L}}$ tra una $\mathcal{L}$-situazione M, u e una formula modale φ è definita come segue:

$$
\begin{array}{lcl}
M, u \models_{\mathcal{L}} A_i,\ A_i \in \mathcal{A} & \Longleftrightarrow & \pi(A_i, u) = T; \\
M, u \models_{\mathcal{L}} \neg\varphi_1 & \Longleftrightarrow & M, u \not\models_{\mathcal{L}} \varphi_1; \\
M, u \models_{\mathcal{L}} \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & M, u \models_{\mathcal{L}} \varphi_1 \text{ e } M, u \models_{\mathcal{L}} \varphi_2; \\
M, u \models_{\mathcal{L}} \Box_r\varphi_1,\ \Box_r \in \mathcal{B} & \Longleftrightarrow & M, v \models_{\mathcal{L}} \varphi_1 \text{ per ogni } v \in \mathcal{U} \text{ t.c.} \\
 & & \mathcal{R}_r(u, v) \text{ valga in } M.
\end{array}
$$

"$M, u \models_{\mathcal{L}} \varphi$"si legge "$M, u$ *soddisfa* φ in $\mathcal{L}$" (M, u $\mathcal{L}$-*soddisfa* φ). Si dice che una formula $\varphi \in \Lambda_{\Box m}$ è *soddisfacibile* in $\mathcal{L}$ ($\mathcal{L}$-*soddisfacibile*) se esiste una situazione M, u che la $\mathcal{L}$-soddisfi. Per ogni coppia di formule φ_1 e φ_2, diciamo che $\varphi_1 \models_{\mathcal{L}} \varphi_2$ se e solo se $M, u \models_{\mathcal{L}} \varphi_2$ per ogni M, u tale che $M, u \models_{\mathcal{L}} \varphi_1$. Si verifica che $\varphi_1 \models_{\mathcal{L}} \varphi_2$ se e solo se $\models_{\mathcal{L}} \varphi_1 \supset \varphi_2$, e quindi che $\models_{\mathcal{L}} \varphi_1 \equiv \varphi_2$ se e solo se $\varphi_1 \models_{\mathcal{L}} \varphi_2$

Logica normale $L \in \mathcal{N}$	Proprietà di $\mathcal{R}_r$
K	—
KB	simmetrica
KD	seriale
KT = KDT (T)	riflessiva
K4	transitiva
K5	euclidea
KBD	simmetrica e seriale
KBT = KBDT (B)	simmetrica e riflessiva
KB4 = KB5 = KB45	simmetrica e transitiva
KD4	seriale e transitiva
KD5	seriale e euclidea
KT4 = KDT4 (S4)	riflessiva e transitiva
KT5 = KBD4 = KBD5 = KBT4 = KBT5 = KDT5 = KT45 = KBD45 = KBT45 = KDT45 = KBDT4 = KBDT5 = KBDT45 (S5)	riflessiva, transitiva e simmetrica (equivalenza)
K45	transitiva e euclidea
KD45	seriale, transitiva e euclidea

Tabella 2.1: Proprietà di $\mathcal{R}_r$ per le varie logiche modali normali, ed equivalenza tra le varie logiche. Tra parentesi (), i nomi comuni delle relative logiche.

Proprietà di $\mathcal{R}_r$	Descrizione
simmetrica	$\forall\ u\ v\ (\mathcal{R}_r(u,v) \Longrightarrow \mathcal{R}_r(v,u))$
seriale	$\forall\ u\ \exists\ v\ \mathcal{R}_r(u,v)$
riflessiva	$\forall\ u\ \mathcal{R}_r(u,u)$
transitiva	$\forall\ u\ v\ w\ (\mathcal{R}_r(u,v)\ e\ \mathcal{R}_r(v,w) \Longrightarrow \mathcal{R}_r(u,w))$
euclidea	$\forall\ u\ v\ w\ (\mathcal{R}_r(u,v)\ e\ \mathcal{R}_r(u,w) \Longrightarrow \mathcal{R}_r(v,w))$

Tabella 2.2: Le varie possibili proprietà di $\mathcal{R}_r$.

e $\varphi_2 \models_{\mathcal{L}} \varphi_1$. Diciamo inoltre che $\models_{\mathcal{L}} \varphi$ (φ è $\mathcal{L}$-*valida*) se e solo se $M, u \models_{\mathcal{L}} \varphi$ per ogni $\mathcal{L}$-situazione M, u. Si verifica che $\models_{\mathcal{L}} \varphi$ se e solo se $\neg\varphi$ è $\mathcal{L}$-insoddisfacibile.

$\vdash_{\mathcal{L}}$ e $\models_{\mathcal{L}}$ sono legati dal seguente teorema di correttezza e completezza:

$$\vdash_{\mathcal{L}} \varphi \quad \Longleftrightarrow \quad \models_{\mathcal{L}} \varphi.$$

Di conseguenza, φ è $\mathcal{L}$-consistente se e solo se è $\mathcal{L}$-soddisfacibile.

Il problema di determinare la $\mathcal{L}$-soddisfacibilità di una formula è decidibile, e tipicamente PSPACE-complete (ad eccezione di KD45(1) e S5(1) per cui e NP-complete) [Ladner, 1977; Halpern e Moses, 1992; Halpern, 1995]. Nella Sezione 3.2 elenchiamo i principali algoritmi di decisione per logiche modali riportati in letteratura.

In futuro, ogniqualvolta ometteremo il prefisso/pedice "$\mathcal{L}$", intenderemo implicitamente "K(m)".

2.3 Le logiche terminologiche

Presentiamo qui alcune nozioni di base sulle logiche terminologiche (dette anche logiche *descrittive*). Si tratta di formalismi derivati dal formalismo KL-ONE [Brachman e Levesque, 1984] utilizzati per implementare e manipolare basi di conoscenza. Dato l'enorme numero di logiche terminologiche descritte in letteratura, ci limiteremo a descrivere solo i costrutti principali. Si vedano, per esempio, [Donini *et al.*, 1992; Schmidt-Schauß e Smolka, 1991; Bresciani *et al.*, 1995] per dettagli ed approfondimenti.

2.3.1 Sintassi

Gli elementi di base del linguaggio terminologico sono i concetti, i ruoli, gli attributi e gli individui. Un *individuo* rappresenta una singola entità dell'universo rappresentato (ad es., `Mary`). Un *concetto* è una descrizione delle proprietà comuni di un insieme di individui (ad es. `WOMAN` - donna); può essere visto come un predicato su un insieme di individui. Un *ruolo* rappresenta una relazione binaria tra individui (ad es., `CHILD` - figlio). Un *attributo* rappresenta una funzione

C ::=			
A_i	$\mid$	A_i	(concetto atomico)
$\top$	$\mid$	`top`	(top)
$\bot$	$\mid$	`bot`	(bottom)
$\neg C$	$\mid$	(`not` C)	(negazione)
$C_1 \wedge C_2$	$\mid$	(`and` C_1 C_2)	(congiunzione)
$\forall R.C$	$\mid$	(`all` R C)	(restrizione universale)
$\forall f.C$	$\mid$	(`all` f C)	(restrizione universale)
$\leq nR$	$\mid$	(`atleast` n R)	(restrizione numerale $\leq$)
$\geq nR$	$\mid$	(`atmost` n R)	(restrizione numerale $\geq$)
$\{a_1, \ldots, a_n\}$	$\mid$	(`one-of` $a_1 \ldots a_n$)	(uno-di)
	$\ldots$		
R ::=			
r	$\mid$	r	(ruolo atomico)
$R_1 \wedge R_2$	$\mid$	(`and` R_1 R_2)	(congiunzione di ruoli)
R^{-1}	$\mid$	(`inverse` R)	(inversione di ruoli)
$\ldots$			
f ::=			
f	$\mid$	f	(attributo atomico)
$f_1 \circ f_2$	$\mid$	(`compose` f_1 f_2)	(composizione di attributo)
	$\ldots$		

Tabella 2.3: Sintassi dei principali costrutti terminologici.

tra individui (ad es., `MOTHER` - madre). Notazionalmente, useremo le lettere C, C_i per rappresentare concetti generici, $A_1, A_2, \ldots$ per i concetti atomici, le costanti $\{\top, \bot\}$ ("top", "bottom") per il concetto universale e nullo, R, R_i per i ruoli, r per i ruoli atomici, f, f_i per gli attributi, a, a_i per gli individui (generici e atomici), n per numeri interi. Attribuiremo inoltre il significato standard agli operatori matematici $\leq, \geq, ^{-1}$ e agli operatori logici $\neg, \wedge$. Come in precedenza, useremo inoltre abbreviazioni standard "$C_1 \vee C_2$" per "$\neg(\neg C_1 \wedge \neg C_2)$", e non useremo i costrutti "$\exists RC$" e "$\exists fC$", sostituendoli con i concetti equivalenti "$\neg\forall R \neg C$" e "$\neg\forall f \neg C$".

I costrutti principali per rappresentare concetti terminologici sono riassunti nella Tabella 2.3. I concetti sono costruiti sulla base di concetti primitivi, ruoli attributi e individui. I ruoli [attributi] sono costruiti sulla base di ruoli [attributi] primitivi. Ad esempio, se `FEMALE`, `MALE`, `ORPHAN` e `PERSON` (femmina, maschio, orfano e persona) sono concetti primitivi, `CHILD` e `LIVES_WITH` (figlio/a, convivente) sono ruoli primitivi, `MOTHER` (madre) é un attributo primitivo e `Mary` e `Paul` sono individui, allora

$$\texttt{FEMALE} \wedge \leq 3\texttt{CHILD} \wedge \neg\forall\texttt{CHILD MALE} \wedge \neg\texttt{MOTHER} \ \{\texttt{Mary Paul}\}$$

(le donne che hanno al più tre figli di cui almeno uno non maschio, escluse le madri di Mary e Paul) è un concetto composto,

$$\texttt{CHILD} \wedge \texttt{LIVES\_WITH}$$

(figlio/a convivente) è un ruolo composto, e

$$\texttt{MOTHER} \circ \texttt{MOTHER}$$

(nonna materna) è un attributo composto.

Tra le varie logiche terminologiche, va ricordata la logica di base $\mathcal{ALC}$ [Schmidt-Schauß e Smolka, 1991], basata unicamente sulla negazione e congiunzione di concetti e sulla restrizione universale tra concetti e ruoli. Ci riferiremo spesso a tale logica in futuro.

Un *Assioma terminologico* è un assioma scritto in una delle forme "$A_i := C$" (A_i è equivalente a C) o "$A_i \subseteq C$" (A_i è un/una C), ad esempio

$$\begin{aligned} \texttt{PERSON} := &\ \texttt{FEMALE} \vee \texttt{MALE}, \\ \texttt{ORPHAN} \subseteq &\ \texttt{PERSON}, \end{aligned}$$

("una persona è una femmina oppure un maschio", e "un orfano è una persona"). Una *terminologia (Tbox)* $\mathcal{T}$ è un insieme finito di assiomi terminologici, con la restrizione che ogni concetto primitivo A_i possa occorrere come termine sinistro di al più un unico assioma terminologico, e che la terminologia non possa avere cicli. Un'*asserzione di appartenenza* è un'asserzione su singoli individui, ad esempio:

```
(assert-ind PERSON Mary),
(assert-ind CHILD Paul Mary),
(assert-ind MOTHER Mary Paul).
```

Un insieme di asserzioni di appartenenza è detta *Abox*. Un insieme Σ di assiomi terminologici e di asserzioni di appartenenza è detta una *base di conoscenza*. Si noti che ogni base di conoscenza Σ può essere ridotta ad una equivalente Σ' senza assiomi terminologici, espandendo ogni concetto definito con la sua definizione all'interno dell'Abox.

2.3.2 Semantica

Un'*interpretazione* $\mathcal{I} =< \Delta^{\mathcal{I}}, ^{\mathcal{I}} >$ consiste in un insieme $\Delta^{\mathcal{I}}$ di individui dell'universo e in una funzione $^{\mathcal{I}}$ che mappa

(i) ogni individuo a_i in un elemento $a_i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, in modo tale che $a_i^{\mathcal{I}} \neq a_j^{\mathcal{I}}$ se $a_i \neq a_j$,

(ii) ogni concetto C in un insieme $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,

(iii) ogni ruolo R in un insieme $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,

(iv) ogni attributo f in una funzione parziale $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \longmapsto \Delta^{\mathcal{I}}$,

il tutto come descritto nella Tabella 2.4.

$\mathcal{I}$ *soddisfa* un assioma $A_i := C$ $[A_i \subseteq C]$ se e solo se $A_i^{\mathcal{I}} = C^{\mathcal{I}}$ $[A_i^{\mathcal{I}} \subseteq C^{\mathcal{I}}]$. $\mathcal{I}$ *soddisfa* un asserzione appartenenza

(assert-ind C a_i), (assert-ind R a_i a_j), (assert-ind f a_i a_j)

$$
\begin{array}{rcl}
A_i^{\mathcal{I}} & = & A_i^{\mathcal{I}} \\
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\perp^{\mathcal{I}} & = & \emptyset \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(C_1 \wedge C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \{i \in \Delta^{\mathcal{I}} | \ \ \forall j.(i,j) \in R^{\mathcal{I}} \Rightarrow j \in C^{\mathcal{I}}\} \\
(\forall f.C)^{\mathcal{I}} & = & \{i \in \Delta^{\mathcal{I}} | \ \ \forall j.(i,j) \in f^{\mathcal{I}} \Rightarrow j \in C^{\mathcal{I}}\} \\
(\leq nR)^{\mathcal{I}} & = & \{i \in \Delta^{\mathcal{I}} | \ \ |\{j|(i,j) \in R^{\mathcal{I}}\}| \leq n\} \\
(\geq nR)^{\mathcal{I}} & = & \{i \in \Delta^{\mathcal{I}} | \ \ |\{j|(i,j) \in R^{\mathcal{I}}\}| \geq n\} \\
\{a_1, \ldots, a_n\}^{\mathcal{I}} & = & \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\} \\
\ldots & & \\
\hline
r^{\mathcal{I}} & = & r^{\mathcal{I}} \\
(R_1 \wedge R_2)^{\mathcal{I}} & = & R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} \\
(R^{-1})^{\mathcal{I}} & = & \{(i,j) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (j,i) \in R^{\mathcal{I}}\} \\
\ldots & & \\
\hline
f^{\mathcal{I}} & = & f^{\mathcal{I}} \\
(f_1 \circ f_2)^{\mathcal{I}} & = & f_1^{\mathcal{I}} \circ f_2^{\mathcal{I}} \\
\ldots & &
\end{array}
$$

Tabella 2.4: Semantica dei principali costrutti terminologici.

se e solo se, rispettivamente

$$a_i^{\mathcal{I}} \in C^{\mathcal{I}},\ (a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \in R^{\mathcal{I}},\ a_i^{\mathcal{I}} = f^{\mathcal{I}}(a_j^{\mathcal{I}}).$$

Un'interpretazione $\mathcal{I}$ è un *modello* di una base di conoscenza Σ se e solo se $\mathcal{I}$ soddisfa tutti gli assiomi e tutte le asserzioni di Σ. Σ è detta *soddisfacibile* se e solo se ha un modello.

Data una base di conoscenza Σ, alcuni concetti C, C_1, C_2 un individuo a e un'asserzione α, si ha che

- Σ *implica logicamente* α (scritto $\Sigma \models \alpha$) se e solo se α è soddisfatta da tutti i modelli di Σ. Verificare $\Sigma \models \alpha$ è equivalente a verificare che $\Sigma \cup \{\neg\alpha\}$ sia insoddisfacibile.

- C è *soddisfacibile in* Σ se esiste un modello $\mathcal{I}$ di Σ tale che $C^{\mathcal{I}} \neq \emptyset$, cioè $\Sigma \not\models (C \equiv \bot)$. Verificare la soddisfacibilità di C è equivalente a trovare a tale che $\Sigma \cup \{\texttt{(assert-ind}\ C\ a\texttt{)}\}$ è soddisfacibile.

- C_1 è *sussunto* da C_2 in Σ (scritto $\Sigma \models C_1 \subseteq C_2$) se $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ per ogni modello $\mathcal{I}$ di Σ. Verificare $\Sigma \models C_1 \subseteq C_2$ è equivalente a verificare che $\Sigma \cup \{C_1 \wedge \neg C_2\}$ sia insoddisfacibile.

- a è detto essere un'*istanza* di C all'interno di Σ se e solo se $\Sigma \models \{\texttt{(assert-ind}\ C\ a\texttt{)}\}$. Verificare ciò (*instance checking*) è equivalente a verificare che $\Sigma \cup \{\neg\texttt{(assert-ind}\ C\ a\texttt{)}\}$ sia insoddisfacibile.

- Il problema di trovare tutte le istanze a di un concetto C (*retrieval problem*) può essere ridotto al problema si instance checking per ogni individuo in Σ.

Di conseguenza, tutte le possibili operazioni su basi di conoscenza possono essere ricondotte alla verifica della soddisfacibilità.

2.3.3 Relazione con le logiche modali

$$
\begin{array}{|rcl|}
\hline
\Phi(\top) & = & T \\
\Phi(\bot) & = & F \\
\Phi(A_i) & = & A_i \\
\Phi(\neg C) & = & \neg\Phi(C) \\
\Phi(C_1 \wedge C_2) & = & \Phi(C_1) \wedge \Phi(C_2) \\
\Phi(\forall r C) & = & \Box_r \Phi(C) \\
\hline
\end{array}
$$

Tabella 2.5: Descrizione ricorsiva della funzione biettiva Φ, che traduce concetti $\mathcal{ALC}$ in formule K(m).

Lo studio delle logiche terminologiche e delle logiche modali è stato portato avanti per anni in comunità distinte, ottenendo risultati analoghi ma distinti (ad esempio, [Halpern e Moses, 1992] e [Schmidt-Schauß e Smolka, 1991]). Recenti risultati hanno evidenziato una forte corrispondenza tra logiche terminologiche e logiche modali (o logiche dinamiche).

In particolare ci soffermiamo sulla logica terminologica $\mathcal{ALC}$. [Schild, 1991] mostra come $\mathcal{ALC}$ sia solo una variante notazionale di K(m). Nella Tabella 2.5 è riportata la descrizione di una funzione biettiva Φ che converte concetti $\mathcal{ALC}$ in formule K(m). Si verifica una corrispondenza biettiva tra le semantica di $\mathcal{ALC}$ e K(m). Analogamente, i risultati di correttezza, completezza, complessità per le due logiche coincidono.

Tali risultati sono stati estesi per un certo numero di logiche terminologiche. Si vedano [Schild, 1991; DeGiacomo e Lenzerini, 1994] per una descrizione particolareggiata.

Capitolo 3

Algoritmi di decisione

"... Pensate che Arlo Crut, una sera, è tornato a casa, ha sbagliato casa ed è finito dritto dritto nel letto di Metel Crut, suo fratello. Metel neanche se ne accorse, dormiva come un sasso, ma la moglie sì che se ne accorse. Un uomo che si infilava nel suo letto. Da non crederci. Beh, sapete cosa gli disse, lei?
E qui nella testa di Padre Pluche si scatenò la consueta sfida. Due belle frasi partirono dai blocchi di partenza del cervello con davanti a sè il traguardo ben preciso di una voce con cui uscire all'aperto. La più sensata delle due, considerato che si trattava pur sempre della voce di un prete, era sicuramente
- Fallo, e *mi metto a gridare.*
Ma aveva il difetto di essere falsa. *Vinse l'altra, quella* vera.
- Fallo, o *mi metto a gridare. ..."*
[A. Baricco, "Oceano mare"]

In questo capitolo illustriamo i principali algoritmi di decisione per logiche proposizionale (Sezione 3.1), modali (Sezione 3.2) e terminologiche (Sezione 3.3) presenti in letteratura.

3.1 Algoritmi di decisione per la logica proposizionale

In questa sezione descriviamo brevemente i principali algoritmi di decisione per logica proposizionale, in particolare Tableau (Sezione 3.1.1), DPLL (Sezione 3.1.2), OBDD (Sezione 3.1.3), KE (Sezione 3.1.4) e GSAT (Sezione 3.1.5). Altri algoritmi minori (o varianti dei precedenti) sono descritti, ad esempio, in [Stallmann e Sussman, 1977; Stalmarck e Saflund, 1990; Ginsberg, 1993].

È importante osservare che molti algoritmi (ad esempio Tableau e KE) possono essere visti nell'ottica

$$Algoritmo = Logica + Controllo$$

[Kowalski, 1979], cioè sono costituiti da un algoritmo di controllo applicato alle regole di inferenza di un formalismo logico.

3.1.1 Tableau

Gli algoritmi tableau-based sono basati sul formalismo tableau descritto in [Smullyan, 1968]. I tableau sono alberi binari di (liste di) formule, la cui radice contiene la formula che si vuole provare (in)soddisfacibile. Data una formula φ, si espandono dapprima tutte le sottoformule "$\equiv$" di φ:

$$\varphi_1 \equiv \varphi_2 \qquad \Longrightarrow \qquad (\varphi_1 \supset \varphi_2) \wedge (\varphi_2 \supset \varphi_1).$$

Un tableau viene costruito come segue. Al primo passo si costruisce la radice contenente φ. Al passo generico, si espande un ramo applicando ad una formula a scelta del ramo la regola corrispondente:

$$\frac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}} \qquad \frac{\neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\neg\varphi_1\\ \neg\varphi_2\end{array}} \qquad \frac{\neg(\varphi_1 \supset \varphi_2)}{\begin{array}{c}\varphi_1\\ \neg\varphi_2\end{array}} \qquad (\wedge\text{-elimination})$$

$$\frac{\neg\neg\varphi}{\varphi} \qquad (\neg\neg\text{-elimination})$$

$$\frac{\varphi_1 \vee \varphi_2}{\varphi_1 \quad\quad \varphi_2} \qquad \frac{\neg(\varphi_1 \wedge \varphi_2)}{\neg\varphi_1 \quad\quad \neg\varphi_2} \qquad \frac{\varphi_1 \supset \varphi_2}{\neg\varphi_1 \quad\quad \varphi_2} \qquad (\vee\text{-elimination}).$$

All'interno dello stesso ramo ad una formula può essere applicata la stessa regola una volta sola. Si noti che le regole $\vee$-elimination creano una diramazione del ramo. Un ramo è *chiuso* se e solo se contiene un atomo A_i e la sua negazione $\neg A_i$, *aperto* altrimenti [1]. Un tableau è *chiuso* se e solo se contiene solo rami chiusi, *aperto* altrimenti. Un ramo è *completamente espanso* se non ci sono più regole applicabili alle sue formule. Un tableau è *completamente espanso* se e solo se tutti i suoi rami aperti sono completamente espansi.

Valgono i seguenti risultati. Data una formula proposizionale φ, φ è insoddisfacibile ($\neg\varphi$ valida) se e solo se esiste un tableau chiuso per φ. Viceversa, l'insieme dei letterali contenuto in ogni ramo aperto completamente espanso del tableau costituisce un assegnamento μ tale che $\mu \models \varphi$. Inoltre, se interpretiamo ogni insieme μ_i come congiunzione dei suoi elementi, la lista $\mu_1, \mu_2, \ldots$ corrispondenti a tutti i rami aperti di un tableau completamente espanso per φ è tale che

$$\varphi \equiv \bigvee_j \mu_j.$$

L'algoritmo tableau-based per logica proposizionale, nella sua versione più diffusa, è rappresentato in Figura 3.1. La procedura viene inizialmente invocata dalla chiamata "$Tableau(\{\varphi\})$". (Per brevità di rappresentazione, supponiamo φ in NNF. Nella pratica vanno gestite tutte le possibili combinazioni equivalenti $\neg\vee$, $\neg\supset$, $\neg\wedge$, $\supset$.) L'idea è di cercare di costruire un tableau per φ percorrendo un ramo alla volta. La procedura termina quando si ottiene un ramo aperto completamente espanso (φ soddisfacibile), o non è più possibile espandere alcun ramo (φ insoddisfacibile). La procedura agisce ricorsivamente, secondo i passi seguenti:

- (**Branch closed**) Se A_i e $\neg A_i$ occorrono entrambi in Γ per qualche indice i, allora il ramo corrente è chiuso e la procedura restituisce $False$.

[1] In alcuni testi si trova la definizione "... se e solo se contiene una formula φ e la sua negazione $\neg\varphi$". Le due definizioni sono equivalenti.

```
function Tableau(Γ)
    if A_i ∈ Γ and ¬A_i ∈ Γ                      /* branch closed */
        then return False;
    if (φ1 ∧ φ2) ∈ Γ                              /* ∧-elimination */
        then return Tableau(Γ ∪ {φ1, φ2}\{(φ1 ∧ φ2)});
    if (¬¬φ1) ∈ Γ                                 /* ¬¬-elimination */
        then return Tableau(Γ ∪ {φ1}\{(¬¬φ1)});
    if (φ1 ∨ φ2) ∈ Γ                              /* ∨-elimination */
        then return Tableau(Γ ∪ {φ1}\{(φ1 ∨ φ2)})   or
                    Tableau(Γ ∪ {φ2}\{(φ1 ∨ φ2)});
    return True;                                  /* branch expanded */
```

Figura 3.1: Schema dell'algoritmo tableau per logica proposizionale.

- (**$\wedge$-elimination**) Altrimenti, se $(\varphi_1 \wedge \varphi_2) \in \Gamma$, allora viene sostituita con φ_1, φ_2 in Γ e *Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\vee$ e $\neg\supset$.

- (**$\neg\neg$-elimination**) Altrimenti, se $\neg\neg\varphi_1 \in \Gamma$, allora viene sostituita con φ_1 all'interno di Γ e *Tableau* viene invocato ricorsivamente sull'insieme risultante.

- (**$\vee$-elimination**) Altrimenti, se $(\varphi_1 \vee \varphi_2) \in \Gamma$, allora viene sostituita con φ_1 all'interno di Γ e *Tableau* viene invocato ricorsivamente sull'insieme risultante. Se la chiamata restituisce $True$, allora tale valore viene restituito a sua volta. Altrimenti, $(\varphi_1 \wedge \varphi_2)$ viene sostituita con φ_2 all'interno di Γ e *Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\wedge$ e $\supset$.

- (**Branch expanded**) Se nessuno dei precedenti casi si verifica, allora il ramo è completamente espanso, ed è aperto. Di conseguenza, la procedura restituisce $True$.

È importante osservare che, se sostituiamo il passo "Branch expanded" con il passo:

Print(Γ); /* branch expanded */
return *False*;

allora *Tableau* termina dopo aver stampato una lista di assegnamenti $\{\mu_1, \mu_2, \ldots\}$ tali che $\varphi \equiv \bigvee_j \mu_j$, (cioè con una minima modifica *Tableau* è in grado di produrre tutti gli assegnamenti per φ).

L'algoritmo di Figura 3.1 essenzialmente applica le regole tableau seguendo l'ordine "$\wedge$, $\neg\neg$, $\vee$". Le regole sono applicate in modo distruttivo: ad ogni applicazione la formula "origine" viene rimossa dal ramo, in quanto non piú necessaria. La scelta dell'ordine non è casuale, in quanto rimandare il più possibile le applicazioni della regola "$\vee$-elimination" minimizza la dimensione del tableau.

Un gran numero di algoritmi e formalismi tableau-based che estendono i tableau proposizionali a logiche più complesse sono presenti in letteratura. Tra questi citiamo, a titolo di esempio, algoritmi/formalismi per logica del primo ordine [Smullyan, 1968; Fitting, 1988], logiche modali [Fitting, 1983; Catach, 1991; Massacci, 1994], logiche terminologiche [Hollunder *et al.*, 1990; Baader *et al.*, 1994; Bresciani *et al.*, 1995], logiche dinamiche [DeGiacomo e Massacci, 1996], logiche contestuali [Massacci, 1995]. Una raccolta di sistemi tableau-based implementati e disponibili è riportata in [Schumann, 1994].

3.1.2 DPLL

Allo stato dell'arte, la procedura più nota (e probabilmente la più efficiente) per la soddisfacibilità di formule proposizionali CNF è DPLL (Davis-Putnam-Longemann-Loveland) [Davis e Putnam, 1960; Davis *et al.*, 1962]. La versione base dell'algoritmo di DPLL è riportata in Figura 3.2. DPLL viene invocato dalla chiamata iniziale "$DPLL(\varphi, T)$", dove φ è è una formula proposizionale CNF e T

```
function DPLL(φ, μ)
    if φ = T                                       /* base      */
        then return True;
    if φ = F                                       /* backtrack */
        then return False;
    if {a unit clause (l) occurs in φ}             /* unit      */
        then return DPLL(assign(l, φ), μ ∧ l);
    l := choose-literal(φ);                        /* split     */
    return DPLL(assign(l, φ), μ ∧ l)  or
            DPLL(assign(¬l, φ), μ ∧ ¬l);
```

Figura 3.2: Schema dell'algoritmo DPLL.

l'assegnamento vuoto, e restituisce *True* se φ è soddisfacibile, *False* altrimenti. In generale, $DPLL()$ cerca di costruire un assegnamento μ che soddisfi la formula originale φ, partendo dall'assegnamento vuoto T e aggiungendo un letterale ad ogni chiamata ricorsiva. Questo avviene secondo i passi seguenti:

- (**base**) Se φ non contiene alcuna clausola (cioè $\varphi = T$), allora μ soddisfa la formula originale φ. Conseguentemente, la funzione restituisce *True*.

- (**backtrack**) Se φ contiene una clausola vuota (cioè $\varphi = F$), allora μ non soddisfa la formula originale φ, quindi DPLL restituisce $False$.

- (**unit**) Se un letterale l occorre in φ come clausola unitaria (o equivalente per formule non-CNF), cioè

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

  per qualche sottoformula φ_1 e φ_2, allora a l deve necessariamente essere assegnato T. A tal scopo, DPLL viene invocata ricorsivamente, passando come argomenti la formula restituita da *assign*(l, φ) e l'assegnamento $\mu \wedge l$ (cioè $\mu \cup \{\psi := T\}$

se $l = \psi$ e $\mu \cup \{\psi := F\}$ se $l = \neg\psi$. La funzione *assign*(l, φ) assegna $l := T$ 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, allora la funzione *choose-literal*(φ) restituire con qualche criterio euristico un letterale l non ancora assegnato. Quindi DPLL viene dapprima invocato con *assign*(l, φ) e $\mu \wedge l$. Se il risultato è negativo, DPLL viene invocato con *assign*$(\neg l, \varphi)$ e $\mu \wedge \neg l$.

È importante osservare che, se sostituiamo il passo base con il passo:

```
if φ = T                                  /* base      */
   then Print(μ);
        return False;
```

allora DPLL termina dopo aver stampato un insieme di assegnamenti $\{\mu_1, \mu_2, \ldots\}$ tali che $\varphi \equiv \bigvee_j \mu_j$, cioè con una minima modifica DPLL è in grado di produrre un insieme completo di assegnamenti per φ.

Efficienti implementazioni di DPLL sono descritte in [Buro e Buning, 1992; Crawford e Auton, 1993; Zhang e Stickel, 1994]. L'efficienza dell'implementazione dipende essenzialmente da due fattori: l'implementazione di *assign* e la scelta dell'euristica per *choose-literal*. Metodi per efficienti implementazioni della funzione *assign* sono descritti, ad esempio, in [Zhang e Stickel, 1994; Buro e Buning, 1992]. Esiste inoltre una nutrita bibliografia sulle euristiche per *choose-literal* (si veda, ad esempio, [Jeroslow e Wang, 1990; Buro e Buning, 1992]). Stiamo studiando la possibilità di usare a tal fine algoritmi di valutazione della *criticità* dei letterali, simili a quelli illustrati in [Bundy *et al.*, 1996a; Bundy *et al.*, 1996b].

Nonostante DPLL sia stato concepito per formule CNF, sono state proposte estensioni a tutte le formule proposizionali [Armando e Giunchiglia, 1993; van Gelder, 1988]. La differenza consiste nel concetto di clausola unitaria e nell'implementazione della funzione *assign*. Intuitivamente, l occorre come "clausola unitaria" in una formula non-CNF φ se occorre come clausola unitaria in $cnf(\varphi)$, dove $cnf(\varphi)$ è il risultato della conversione in CNF di φ applicando le regole di DeMorgan. In [Armando e Giunchiglia, 1993; van Gelder, 1988] vengono presentati metodi per rivelare la presenza di clausole unitarie senza calcolare esplicitamente $cnf(\varphi)$. La funzione *assign* per formule non-CNF può venire implementata tramite una "lazy evaluation", cioè tramite l'applicazione ricorsiva delle seguenti regole di semplificazione:

$$\begin{array}{lcl} \neg F & \Longrightarrow & T, \\ \neg T & \Longrightarrow & F, \\ \varphi \wedge T & \Longrightarrow & \varphi, \\ \varphi \wedge F & \Longrightarrow & F, \\ \varphi \vee T & \Longrightarrow & T, \\ \varphi \vee F & \Longrightarrow & \varphi, \end{array}$$

ed equivalenti per $\{\supset, \equiv\}$. Si noti che, se φ è CNF, questo corrisponde alla precedente descrizione di *assign*.

3.1.3 OBDD

Gli *Ordered Binary Decision Diagrams* (OBBD d'ora in poi) vennero inventati all'interno delle comunità dei progettisti hardware come uno strumento per rappresentare e manipolare efficientemente funzioni booleane [Bryant, 1992], e successivamente importati nelle comunità di deduzione automatica [Moore, 1994]. Un OBDD è un grafo orientato aciclico, dove ogni nodo è etichettato con un atomo proposizionale A_i e possiede due archi uscenti, detti *then* ed *else*, diretti ad altri nodi. Esiste un nodo *radice* senza archi entranti e due nodi terminali T e F, senza archi uscenti. Un arco uscente da un nodo A_i può entrare in un nodo A_j solo se $A_i \prec A_j$, dove $\prec$ è un ordine prefissato. La formula proposizionale φ rappresentata da un OBDD è definita ricorsivamente da una serie di costrutti

$$\text{``}if\ A_i\ then\ \varphi_1\ else\ \varphi_2'',$$

dove A_i è la variabile del nodo radice, φ_1 e φ_2 sono le formule proposizionali rappresentate dai suoi discendenti. In un OBDD rappresentante una formula φ per un ordinamento $\prec$ prefissato ($obdd(\varphi, \prec)$ d'ora in poi)) i nodi T e F incontrati al termine di ogni sequenza stabiliscono il valore booleano dell'assegnamento μ per φ corrispondente. In questo modo, l'insieme $\{\mu_1, \mu_2 \ldots\}$ dei percorsi che terminano in T è tale che $\varphi \equiv \bigvee_j \mu_j$. Un OBDD è quindi una rappresentazione compatta dell'insieme dei modelli di una formula proposizionale. Di conseguenza, la dimensione di $obdd(\varphi, \prec)$ è, al caso peggiore, $O(2^{Atoms(\varphi)})$. Gli OBDD godono di una serie di interessanti proprietà:

(i) $obdd(\varphi, \prec)$ è il grafo *minimo* per φ e $\prec$.

(ii) Se $\varphi_1 \equiv \varphi_2$, allora $obdd(\varphi_1, \prec) = obdd(\varphi_2, \prec)$, per ogni ordinamento $\prec$.

(iii) Per ogni operatore booleano $\odot$, $obdd(\varphi_1 \odot \varphi_2, \prec)$ è costruito *incrementalmente* fondendo in modo opportuno $obdd(\varphi_1, \prec)$ e $obdd(\varphi_2, \prec)$.

Per ogni $\prec$, $obdd(\varphi, \prec)$ è costituito dal solo nodo T se e solo se φ è valida, $obdd(\varphi, \prec)$ è costituito dal solo nodo F se e solo se φ è insoddisfacibile. Di conseguenza, ogni algoritmo OBDD-based per soddisfacibilità di formule proposizionali φ genera $obdd(\varphi, \prec)$ e poi analizza il grafo ottenuto. Un'efficiente implementazione di un algoritmo OBDD-based è descritta, ad esempio, in [Long, 1993]. Un confronto tra l'efficienza degli algoritmi OBDD-based con quella di DPLL è riportato in [Uribe e Stickel, 1994].

3.1.4 KE

KE è una variante del formalismo tableau introdotta in [D'Agostino, 1992; D'Agostino e Mondadori, 1994]. Come nei tableau, l'idea è di costruire alberi binari di (liste di) formule, la cui radice contiene la formula che si vuole provare (in)soddisfacibile. La differenza consiste nelle regole usate:

$$\frac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}} \quad \frac{\neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\neg\varphi_1\\ \neg\varphi_2\end{array}} \quad \frac{\neg(\varphi_1 \supset \varphi_2)}{\begin{array}{c}\varphi_1\\ \neg\varphi_2\end{array}} \qquad (\wedge\text{-elimination})$$

$$\frac{\neg\neg\varphi}{\varphi} \qquad (\neg\neg\text{-elimination})$$

$$\frac{\begin{array}{c}\neg\varphi_2\\ \varphi_1 \vee \varphi_2\end{array}}{\varphi_1} \quad \frac{\begin{array}{c}\varphi_2\\ \neg(\varphi_1 \wedge \varphi_2)\end{array}}{\neg\varphi_1} \quad \frac{\begin{array}{c}\varphi_1\\ \varphi_1 \supset \varphi_2\end{array}}{\varphi_2} \quad \frac{\begin{array}{c}\neg\varphi_2\\ \varphi_1 \supset \varphi_2\end{array}}{\neg\varphi_1} \qquad (\vee\text{-elimination}),$$

$$\frac{}{\varphi_1 \qquad \neg\varphi_1} \qquad (\text{bivalence}).$$

Le regole $\wedge$-elimination e $\neg\neg$-elimination sono identiche al tableau standard. Le regole $\vee$-elimination riducono una disgiunzione ad uno dei disgiunti qualora sia presente la negazione dell'altro (le prime due regole valgono anche simmetricamente). Le diramazioni nell'albero sono create unicamente dalla regola bivalence, che divide il ramo in funzione del valore di verità da attribuire ad una qualche sottoformula.

Valgono le stesse definizioni e risultati dei tableau. KE gode di un'importante proprietà nei confronti di tableau: KE simula linearmente tableau, mentre tableau non simula polinomialmente KE.

La procedura canonica KE è rappresentata in Figura 3.3. La procedura viene inizialmente invocata dalla chiamata "$KE(\{\varphi\})$". (Per brevità di rappresentazione, consideriamo solo i casi $\wedge$ e $\vee$. Nella pratica vanno gestite tutte le possibili combinazioni equivalenti $\neg\vee$, $\neg\supset$, $\neg\wedge$, $\supset$.) La procedura agisce ricorsivamente, secondo i passi seguenti:

- (**Branch closed**) Se A_i e $\neg A_i$ occorrono entrambi in Γ per qualche indice i, allora il ramo corrente è chiuso e la procedura restituisce $False$.

- (**$\wedge$-elimination**) Altrimenti, se $(\varphi_1 \wedge \varphi_2) \in \Gamma$, allora viene sostituita con φ_1, φ_2 all'interno di Γ e KE viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\vee$ e $\neg\supset$.

function $KE(\Gamma)$
    if $A_i \in \Gamma$ **and** $\neg A_i \in \Gamma$ /* branch closed */
        then return *False*;
    if $(\varphi_1 \wedge \varphi_2) \in \Gamma$ /* $\wedge$-elimination */
        then return $KE(\Gamma \cup \{\varphi_1, \varphi_2\} \backslash \{(\varphi_1 \wedge \varphi_2)\})$;
    if $(\neg\neg\varphi_1) \in \Gamma$ /* $\neg\neg$-elimination */
        then return $KE(\Gamma \cup \{\varphi_1\} \backslash \{(\neg\neg\varphi_1)\})$;
    if $((\varphi_1 \vee \varphi_2) \in \Gamma$ **and** $\neg\varphi_2 \in \Gamma)$ **or** /* $\vee$-elimination */
        $((\varphi_2 \vee \varphi_1) \in \Gamma$ **and** $\neg\varphi_2 \in \Gamma)$
        then return $KE(\Gamma \cup \{\varphi_1\} \backslash \{(\varphi_1 \vee \varphi_2)\})$
    if $(\varphi_1 \vee \varphi_2) \in \Gamma$ **and** $\varphi_1 \notin \Gamma$ **and** $\varphi_2 \notin \Gamma$ /* bivalence */
        then return $KE(\Gamma \cup \{\varphi_1\})$ **or**
                $KE(\Gamma \cup \{\neg\varphi_1\})$;
    return *True*; /* branch expanded */

Figura 3.3: Schema della procedura canonica per KE.

- (**$\neg\neg$-elimination**) Altrimenti, se $\neg\neg\varphi_1 \in \Gamma$, allora viene sostituita con φ_1 all'interno di Γ e *KE* viene invocato ricorsivamente sull'insieme risultante.

- (**$\vee$-elimination**) Altrimenti, se $(\varphi_1 \vee \varphi_2) \in \Gamma$ (o $(\varphi_2 \vee \varphi_1) \in \Gamma$) e $\neg\varphi_2 \in \Gamma$, allora viene sostituita con φ_1 all'interno di Γ e *KE* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\wedge$ e $\supset$.

- (**bivalence**) Altrimenti, se $(\varphi_1 \wedge \varphi_2) \in \Gamma$, $\varphi_1 \notin \Gamma$ e $\varphi_2 \notin \Gamma$, allora dapprima φ_1 viene aggiunto a Γ e *KE* viene invocato ricorsivamente sull'insieme risultante. Se la chiamata restituisce $True$, allora tale valore viene restituito a sua volta. Altrimenti, $\neg\varphi_1$ viene aggiunto a Γ e *KE* viene invocato ricorsivamente sull'insieme risultante.

- (**Branch expanded**) Se nessuno dei precedenti casi si verifica, allora il ramo è completamente espanso, ed è aperto. Di conseguenza, la procedura restituisce $True$.

Come in tableau, se sostituiamo il passo "Branch expanded" con il passo:

Print(Γ*)*; /* branch expanded */
return *False*;

allora *KE* termina dopo aver stampato una lista di assegnamenti $\{\mu_1, \mu_2, \ldots\}$ tali che $\varphi \equiv \bigvee_j \mu_j$,.

L'algoritmo di Figura 3.1 essenzialmente applica le regole KE seguendo l'ordine "$\wedge$, $\neg\neg$, $\vee$, bivalence". Anche in questo caso la scelta dell'ordine non è casuale, in quanto rimandare il più possibile le applicazioni della regola "bivalence" minimizza la dimensione del tableau.

3.1.5 GSAT

GSAT è un algoritmo per soddisfacibilità introdotto in [Selman *et al.*, 1992; Selman e Kautz, 1993]. Si tratta di un algoritmo *incompleto*, in quanto non è in grado di affermare l'insoddisfacibilità di una formula, ma solo la sua soddisfacibilità.

Si definisce *score* di un assegnamento μ per una formula CNF φ ($score(\mu, \varphi)$) il numero di clausole in φ falsificate da μ. L'idea è di cercare, con una tecnica di hill-climbing, un assegnamento che minimizzi $score(\mu, \varphi)$, finché $score(\mu, \varphi) = 0$ (φ soddisfacibile). Lo schema di Figura 3.4 descrive le varie varianti di GSAT (usiamo la notazione di [Gent e Walsh, 1993]). GSAT esegue al più *Max-tries* tentativi. Ad ogni tentativo l'algoritmo parte da un assegnamento casuale μ (prodotto da *rand-assign*), ed esegue al più *Max-flips* cicli. Ad ogni ciclo, se $score(\mu, \varphi) = 0$, allora restituisce $True$ e termina ($\mu \models \varphi$), altrimenti cerca un assegnamento dal punteggio migliore. Chiamando genericamente $\mu^{(i)}$ l'assegnamento ottenuto da μ invertendo il valore di verità assegnato ad A_i, la funzione *hill-climb(*μ, φ*)* seleziona l'insieme *best-flips* degli atomi A_i che che rendono minimo $score(\mu^{(i)}, \varphi)$. Una di queste variabili A_i viene pescata

```
function GSAT(φ)
    for i := 1 to Max-tries do
        μ := rand-assign(φ);
        for j := 1 to Max-flips do
            if score(φ, μ) = 0
                then
                    return True;
                else Best-flips := hill-climb(φ, μ);
                    A_i := rand-pick(Best-flips);
                    μ := flip(A_i, μ);
        end
    end
    return "no satisfying assignment found".
```

Figura 3.4: Uno schema generale di GSAT.

in modo casuale (funzione *rand-pick*) e μ diventa $\mu^{(i)}$ (funzione *flip*). Dopo *Max-tries*·*Max-flips* tentativi, GSAT termina riportando un messaggio di fallimento.

GSAT si è dimostrato estremamente efficiente per trovare assegnamenti di formule CNF soddisfacibili, superando nettamente le prestazioni di DPLL [Selman *et al.*, 1992; Selman e Kautz, 1993]. Sfortunatamente, il fatto che sia incompleto gli impedisce di essere utilizzato in deduzione automatica.

In [Sebastiani, 1994] è descritto NC-GSAT, un'estensione dell'algoritmo GSAT a formule non-CNF. L'idea si basa sull'osservazione che, nell'algoritmo di Figura 3.4, l'unico punto in cui è necessaria la forma CNF è il calcolo dei valori $score(\mu^{(i)}, \varphi)$ in *hill-climb*. Data una formula non-CNF φ, definiamo

$$score(\mu, \varphi) := score(\mu, cnf(\varphi)),$$

dove $cnf(\varphi)$ rappresenta la conversione CNF classica (basata sulle regole di DeMorgan) cioè il numero di clausole di $cnf(\varphi)$ falsifi-

φ	$s(\mu,\varphi)$	$s^-(\mu,\varphi)$
φ *atomo*	$\begin{cases} 0 & se\ \mu \models \varphi \\ 1 & altrimenti \end{cases}$	$\begin{cases} 1 & se\ \mu \models \varphi \\ 0 & altrimenti \end{cases}$
$\neg\varphi_1$	$s^-(\mu,\varphi_1)$	$s(\mu,\varphi_1)$
$\bigwedge_k \varphi_k$	$\sum_k s(\mu,\varphi_k)$	$\prod_k s^-(\mu,\varphi_k)$
$\bigvee_k \varphi_k$	$\prod_k s(\mu,\varphi_k)$	$\sum_k s^-(\mu,\varphi_k)$
$\varphi_1 \equiv \varphi_2$	$\begin{cases} s^-(\mu,\varphi_1)\cdot s(\mu,\varphi_2)+ \\ s(\mu,\varphi_1)\cdot s^-(\mu,\varphi_2) \end{cases}$	$\begin{cases} (s(\mu,\varphi_1)+s^-(\mu,\varphi_2))\cdot \\ (s^-(\mu,\varphi_1)+s(\mu,\varphi_2)) \end{cases}$
$\varphi_1 \supset \varphi_2$	$s^-(\mu,\varphi_1)\cdot s(\mu,\varphi_2)$	$s(\mu,\varphi_1)+s^-(\mu,\varphi_2)$

Tabella 3.1: Definizione ricorsiva di $score(\mu, cnf(\varphi)) := s(\mu,\varphi)$.

cate da μ. La funzione $s(\mu,\varphi)$ definita nella Tabella 3.1 calcola $score(\mu, cnf(\varphi))$ *senza generare esplicitamente* $cnf(\varphi)$, in tempo $O(|\varphi|)$. Poichè $|cnf(\varphi)| = O(2^{|\varphi|})$, calcolare $score(\mu, cnf(\varphi))$ esplicitamente richiederebbe un numero $O(2^{|\varphi|})$ di passi, mentre $s(\mu,\varphi)$ ne richiede $O(|\varphi|)$.

3.2 Algoritmi di decisione per le logiche modali $\mathcal{N}$

In questa sezione descriviamo brevemente i principali algoritmi di decisione per logiche modali normali, soffermandoci in particolare sugli algoritmi tableau-based.

3.2.1 Algoritmi tableau-based

Gli algoritmi tableau-based per logiche modali normali sono basati su varianti del formalismo tableau descritto in [Fitting, 1983], che estende il formalismo tableau proposizionale [Smullyan, 1968] (un formalismo con caratteristiche simili è descritto in [Massacci, 1994]). Tale estensione si basa sulla nozione di *formule etichettate*: una formula etichettata è data da una coppia $\sigma : \varphi$, dove φ è una formula e σ è un'*etichetta* (una sequenza di interi $n_1, n_2, \ldots$) che rappresenta uno stato in una struttura di Kripke. Intuitivamente, $\sigma : \varphi$ significa "la formula φ nello stato σ".

Per semplicità, nella trattazione seguente considereremo il caso di modalità singole $\Box$. Il caso di modalità multiple $\Box_1, \Box_2 \ldots$ è assolutamente analogo.

Per ogni logica modale normale $\mathcal{L} \in \mathcal{N}$, i tableau in $\mathcal{L}$ ($\mathcal{L}$-tableau) sono alberi binari di (liste di) formule etichettate, la cui radice contiene $\{\sigma : \varphi\}$, dove φ è la formula che si vuole provare (in)soddisfacibile, e σ è lo stato di partenza. Come nel caso proposizionale, si espandono dapprima tutte le sottoformule "$\equiv$". Un'etichetta si dice *usata* in un ramo di un $\mathcal{L}$-tableau se almeno una formula $\sigma : \varphi$ occorre nel ramo stesso; si dice *non ristretta* in un ramo se non è un segmento iniziale di un'etichetta usata nel ramo; si dice essere un'*estensione semplice* di un'altra etichetta σ' se $\sigma' = \sigma, n$ per qualche intero n. Si noti che queste definizioni sono indipendenti dalla logica considerata. Viene data inoltre una definizione di *accessibilità* tra etichette, funzione della logica considerata, che rappresenta l'equivalente nozione di accessibilità tra stati descritta nelle Tabelle 2.1 e 2.2 (si vedano [Fitting, 1983] per approfondimenti). Un $\mathcal{L}$-tableau viene costruito come segue. Al primo passo si costruisce la radice contenente $\{\sigma : \varphi\}$. Al passo generico, si espande un ramo applicando ad una formula del ramo la regola proposizionale corrispondente:

$$\frac{\sigma : (\varphi_1 \wedge \varphi_2)}{\begin{array}{c}\sigma : \varphi_1\\ \sigma : \varphi_2\end{array}} \qquad \frac{\sigma : \neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\sigma : \neg\varphi_1\\ \sigma : \neg\varphi_2\end{array}} \qquad \frac{\sigma : \neg(\varphi_1 \supset \varphi_2)}{\begin{array}{c}\sigma : \varphi_1\\ \sigma : \neg\varphi_2\end{array}} \qquad (\wedge\text{-elim.})$$

$$\frac{\sigma : \neg\neg\varphi}{\sigma : \varphi} \qquad (\neg\neg\text{-elim.})$$

$$\frac{\sigma : (\varphi_1 \vee \varphi_2)}{\sigma : \varphi_1 \quad \sigma : \varphi_2} \quad \frac{\sigma : \neg(\varphi_1 \wedge \varphi_2)}{\sigma : \neg\varphi_1 \quad \sigma : \neg\varphi_2} \quad \frac{\sigma : (\varphi_1 \supset \varphi_2)}{\sigma : \neg\varphi_1 \quad \sigma : \varphi_2} \qquad (\vee\text{-elim.}).$$

o le regole modali corrispondenti:

$$\frac{\sigma : \neg\Box\varphi}{\sigma' : \neg\varphi} \quad (\neg\Box\text{-elimination})$$

$$\frac{\sigma : \Box\varphi}{\sigma'' : \varphi} \qquad (\Box\text{-elimination})$$

Queste ultime sono sottoposte ad una serie di vincoli di applicabilità sulle etichette:

- **$\neg\Box$-elimination**: σ' dev'essere un'estensione semplice non ristretta di σ.

- **$\Box$-elimination**: σ'' dev'essere usata nel ramo. Nelle logiche KD, T, KBD, B, KD4, S4 ed S5 può alternativamente essere un'estensione semplice non ristretta di σ.

Un ramo è *chiuso* se e solo se contiene un atomo proposizionale $\sigma : A_i$ e la sua negazione $\sigma : \neg A_i$, *aperto* altrimenti. [2] Un $\mathcal{L}$-tableau è *chiuso* se e solo se contiene solo rami chiusi, *aperto* altrimenti. Un ramo è *completamente espanso* se non ci sono più regole applicabili alle sue formule. [3] Un $\mathcal{L}$-tableau è *completamente espanso* se e solo se tutti i suoi rami aperti sono completamente espansi. Data una formula modale φ, φ è $\mathcal{L}$-insoddisfacibile ($\neg\varphi$ $\mathcal{L}$-valida) se e solo se esiste un $\mathcal{L}$-tableau chiuso per $\{1 : \varphi\}$.

È importante osservare che i tableau etichettati catturano anche quelli non etichettati, che possono essere rappresentati usando un'unica etichetta. La quasi totalità degli algoritmi effettivamente implementati utilizzano tableau non etichettati.

Vari algoritmi tableau-based generali per varie logiche $\mathcal{N}$ sono stati proposti [Fitting, 1983; Halpern e Moses, 1992; Massacci, 1994]. In Figura 3.5 presentiamo in dettaglio una versione funzionale per la logica K(m), analoga a quella in [Halpern e Moses, 1992]. La procedura viene inizialmente invocata dalla chiamata "*K(m)-Tableau*($\{\varphi\}$)". (Per brevità di rappresentazione, supponiamo φ in NNF. Nella pratica vanno gestite tutte le possibili combinazioni equivalenti $\neg\vee$, $\neg\supset$, $\neg\wedge$, $\supset$.) La procedura agisce ricorsivamente, secondo i passi seguenti:

[2] La definizione di chiusura in cui si usano atomi generici o formule generiche è equivalente.

[3] In alcune logiche quali K4 e S4 è possibile avere rami aperti infiniti e ciclici. Per brevità in questa breve presentazione ignoriamo questo aspetto. Si veda [Fitting, 1983] per approfondimenti.

function *K(m)-Tableau*(Γ)
  if $A_i \in \Gamma$ **and** $\neg A_i \in \Gamma$ /* branch closed */
    then return *False*;
  if $(\varphi_1 \wedge \varphi_2) \in \Gamma$ /* $\wedge$-elimination */
    then return *K(m)-Tableau*($\Gamma \cup \{\varphi_1, \varphi_2\} \backslash \{(\varphi_1 \wedge \varphi_2)\}$);
  if $(\neg\neg\varphi_1) \in \Gamma$ /* $\neg\neg$-elimination */
    then return *K(m)-Tableau*($\Gamma \cup \{\varphi_1\} \backslash \{(\neg\neg\varphi_1)\}$);
  if $(\varphi_1 \vee \varphi_2) \in \Gamma$ /* $\vee$-elimination */
    then return *K(m)-Tableau*($\Gamma \cup \{\varphi_1\} \backslash \{(\varphi_1 \vee \varphi_2)\}$) **or**
      K(m)-Tableau($\Gamma \cup \{\varphi_2\} \backslash \{(\varphi_1 \vee \varphi_2)\}$);
  for any $\neg\Box_r \beta_{rj} \in \Gamma$ **do** /* branch expanded */
    if not *K(m)-Tableau*($\{\neg\beta_{rj}\} \cup \bigcup_{\Box_r \alpha_{ri} \in \Gamma} \{\alpha_{ri}\}$)
      then return *False*;
  return *True*;

Figura 3.5: Schema dell'algoritmo Tableau per logica modale K(m).

- (**Branch closed**) Se A_i e $\neg A_i$ occorrono entrambi in Γ per qualche indice i, allora il ramo corrente è chiuso e la procedura restituisce $False$.

- (**$\wedge$-elimination**) Altrimenti, se $(\varphi_1 \wedge \varphi_2) \in \Gamma$, allora viene sostituita con φ_1, φ_2 all'interno di Γ e *K(m)-Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\vee$ e $\neg \supset$.

- (**$\neg\neg$-elimination**) Altrimenti, se $\neg\neg\varphi_1 \in \Gamma$, allora viene sostituita con φ_1 all'interno di Γ e *K(m)-Tableau* viene invocato ricorsivamente sull'insieme risultante.

- (**$\vee$-elimination**) Altrimenti, se $(\varphi_1 \vee \varphi_2) \in \Gamma$, allora dapprima viene sostituita con φ_1 all'interno di Γ e *K(m)-Tableau* viene invocato ricorsivamente sull'insieme risultante. Se la chiamata restituisce $True$, allora tale valore viene restituito a sua volta. Altrimenti, $(\varphi_1 \wedge \varphi_2)$ viene sostituita con φ_2 all'interno di Γ e *K(m)-Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\wedge$ e $\supset$.

- (**Branch expanded**) Se nessuno dei precedenti casi si verifica, allora il ramo è aperto ed è costituito da un insieme Γ di formule di tipo A_i, $\neg A_i$, $\Box_r \alpha_{ri}$ o $\neg\Box_r \beta_{rj}$. Allora, per ogni formula del tipo $\Box_r \neg\beta_{rj}$ in Γ, *K(m)-Tableau* viene invocato sull'insieme $\{\neg\beta_{rj}\}$ unito all'insieme delle formule α_{ri} tali che $\Box_r \alpha_{ri} \in \Gamma$. Qualora una delle chiamate ricorsive restituisca *False*, allora anche la chiamata principale restituisce *False*. Altrimenti restituisce *True*.

L'algoritmo di Figura 3.1 essenzialmente applica le regole tableau seguendo l'ordine "$\wedge$, $\neg\neg$, $\vee$, $\neg\Box$, $\Box$". Le regole sono applicate in modo distruttivo: ad ogni applicazione la formula "origine" viene rimossa dal ramo, in quanto non piú necessaria. Si noti che l'uso delle etichette, in questo caso, è *implicito* nella procedura. Infatti, nel passo "branch expanded", la procedura "salta" da uno stato ad uno stato direttamente accessibile. Infatti, se chiamiamo j il contatore delle formule "$\neg\Box_r$" in Γ, ogni chiamata ricorsiva di *K(m)-Tableau* corrisponde ad applicare la regola $\neg\Box_r$-elimination a $\neg\Box_r \beta_{rj}$ con $\sigma' = \sigma, j$ e poi applicare la regola $\Box_r$-elimination a tutte le formule $\Box_r \alpha_{ri}$, con $\sigma'' = \sigma, j$. La rappresentazione esplicita delle etichette non è necessaria, in quanto in K(m) — contrariamente, ad esempio, a KB(m) — non si "ritorna" mai ad uno stato precedentemente abbandonato.

Nonostante un gran numero di formalismi ed algoritmi tableau-based siano stati presentati, sono relativamente poche le procedure effettivamente implementate. Tra queste ricordiamo quelle descritte in [Catach, 1991; Demri, 1995; Koshimura e Hasegawa, 1994; Heuerding *et al.*, 1996]

$$
\begin{array}{lcl}
\Phi(A_i, w) & \equiv & A_i(w) \\
\Phi(\neg\varphi, w) & \equiv & \neg\Phi(\varphi, w) \\
\Phi(\varphi_1 \wedge \varphi_2, w) & \equiv & \Phi(\varphi_1, w) \wedge \Phi(\varphi_2, w) \\
\Phi(\Box\varphi, w) & \equiv & \forall v \ (R(u, v) \supset \Phi(\varphi, u)).
\end{array}
$$

Tabella 3.2: La funzione di traduzione $\Phi : \mathcal{L} \longmapsto FOL$.

$\psi \in \mathcal{L}$	Proprietà	$Ax(\psi)$
B	simmetrica	$\forall\ u\ v\ (R(u,v) \supset R(v,u))$
D	seriale	$\forall\ u\ \exists\ v\ R(u,v)$
T	riflessiva	$\forall\ u\ R(u,u)$
4	transitiva	$\forall\ u\ v\ w\ (R(u,v) \wedge\ R(v,w) \supset R(u,w))$
5	euclidea	$\forall\ u\ v\ w\ (R(u,v) \wedge\ R(u,w) \supset R(v,w))$

Tabella 3.3: Rappresentazione di $Ax(\mathcal{L})$.

3.2.2 Altri algoritmi

Nella ricerca in deduzione in logica modale esistono due approcci distinti: l'approccio diretto e l'approccio translazionale o indiretto.

L'approccio *diretto* consiste nel definire sistemi formali specifici per logiche modali, e costruire di conseguenza algoritmi di decisione come algoritmi di controllo applicati alle regole di inferenza di tali meccanismi. Oltre ai già citati metodi tableau-based, ricordiamo metodi basati su Resolution [Ohlbach, 1988a; Enjalbert e del Cerro, 1989] e su Matrix Calculus [Wallen, 1987]. Non analizziamo questi metodi in dettaglio, in quanto, per quanto a nostra conoscenza [4], non ne esistono versioni effettivamente implementate.

L'approccio *indiretto* consiste nel tradurre una formula modale in una formula nella logica del primo ordine (FOL — First Order Logic — d'ora in poi) e utilizzare poi un theorem prover, ad esempio Otter [McCune, 1990], per deciderne la validità. Esistono vari metodi per tradurre formule modali in FOL. Citiamo a titolo di esempio

[4] Con "per quanto a nostra conoscenza" intendiamo, qui ed in tutta la trattazione, che abbiamo esplicitamente chiesto agli autori se avessero implementato il proprio sistema o se fossero a conoscenza che qualcuno l'avesse fatto, ottenendo risposta negativa.

un metodo standard, detto “semantico” o “relazionale”. Siano dati un predicato binario $R()$, che descrive la relazione di accessibilità tra mondi, e un insieme di predicati unari $A_i()$ corrispondenti agli atomi proposizionali A_i, tali che $A_i(w)$ se e solo se A_i vale nello stato w. Se consideriamo w lo stato corrente, allora è possibile tradurre la ogni coppia φ, w in una formula in FOL, secondo la funzione di traduzione ricorsiva Φ riportata in Tabella 3.2. Sia data una logica modale $\mathcal{L} \in \mathcal{N}$, siano ψ_i gli assiomi corrispondenti a $\mathcal{L}$ e sia $Ax(\mathcal{L}) = \bigwedge_i Ax(\psi_i)$, dove Ax è rappresentata nella Tabella 3.3. Si dimostra [Morgan, 1976] che φ è $\mathcal{L}$-valida se e solo se la formula $Ax(\mathcal{L}) \supset \forall w\ \Phi(\varphi, w)$ è valida in FOL. Altri metodi di traduzione di formule modali in FOL sono riassunti in [Auffray e Enjalbert, 1992; Ohlbach, 1988b]. Il problema più evidente di questo secondo approccio consiste nella *semidecidibilità* di FOL, che comporta di conseguenza la possibilità che l'algoritmo non termini.

Nonostante vi siano alcuni risultati preliminari [Schmidt, 1997], allo stato attuale non esiste in letteratura un confronto empirico sufficientemente esaustivo tra l'approccio diretto e quello indiretto.

3.3 Algoritmi di decisione per le logiche terminologiche

In questa sezione descriviamo brevemente i principali algoritmi di decisione per logiche terminologiche, soffermandoci in particolare sugli algoritmi tableau-based.

3.3.1 Algoritmi tableau-based

Gli algoritmi tableau-based per logiche terminologiche sono basati su varianti del formalismo tableau descritto in [Schmidt-Schauß e Smolka, 1991; Donini *et al.*, 1992]. Nonostante le evidenti analogie con i tableau etichettati di [Fitting, 1983], questi formalismi ed algoritmi sono stati sviluppati indipendentemente, fino alla scoperta delle relazioni tra logiche terminologiche e modali [Schild, 1991].

I formalismi tableau-based si basano sulla nozione di *vincolo*: un vincolo è un oggetto sintattico scritto in una delle seguenti forme:

$$x : C, \qquad xry, \qquad x \neq y,$$

dove C è un concetto, r è un ruolo primitivo e x, y sono variabili su individui. Intuitivamente, $x : C$ significa che x è un elemento di C, xry che x e y sono in relazione r tra loro, $x \neq y$ che x e y devono essere interpretati come individui distinti. Un *sistema di vincoli* S è un insieme di vincoli. Estendiamo la semantica ai sistemi di vincoli nel seguente modo: diciamo che $^{\mathcal{I}}$ soddisfa

$$\begin{array}{rll} x : C & \text{se} & x^{\mathcal{I}} \in C^{\mathcal{I}}, \\ xry & \text{se} & (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}, \\ x \neq y & \text{se} & (x^{\mathcal{I}} \neq y^{\mathcal{I}}). \end{array}$$

$^{\mathcal{I}}$ soddisfa un sistema di vincoli S se soddisfa tutti i vincoli in S. Se $R = \bigwedge_{i=1}^{m} r_i$, y è detto un *R-successore* di x in S se $xr_1y, \ldots, xr_my$ appartendono ad S. y è un *successore* di x in S se è un R-successore di x per qualche ruolo R. x e y sono *separati in* S se $x \neq y \in S$. Con $S[y/z]$ denotiamo il sistema di vincoli ottenuto da S rimpiazzando ogni occorrenza di y con z.

Un tableau per logiche terminologiche viene costruito tramite le regole seguenti (se l'antecedente della regola è in S e i vincoli di applicabilità sono rispettati, allora tutti i conseguenti vengono aggiunti a S):

$$\frac{x : (C_1 \wedge C_2)}{\begin{array}{c} x : C_1 \\ x : C_2 \end{array}} \quad \frac{x : \neg(C_1 \vee C_2)}{\begin{array}{c} x : \neg C_1 \\ x : \neg C_2 \end{array}} \quad \frac{x : \neg(C_1 \supset C_2)}{\begin{array}{c} x : C_1 \\ x : \neg C_2 \end{array}} \qquad (\wedge\text{-elim.})$$

$$\frac{x : \neg\neg C}{x : C} \qquad (\neg\neg\text{-elim.})$$

$$\frac{x : (C_1 \vee C_2)}{x : C_1} \quad \frac{x : \neg(C_1 \wedge C_2)}{x : \neg C_1} \quad \frac{x : (C_1 \supset C_2)}{x : \neg C_1} \qquad (\vee\text{-elim. left}).$$

$$\frac{x : (C_1 \vee C_2)}{x : C_2} \quad \frac{x : \neg(C_1 \wedge C_2)}{x : \neg C_2} \quad \frac{x : (C_1 \supset C_2)}{x : C_2} \qquad (\vee\text{-elim. right}).$$

o le regole puramente terminologiche:

$$\frac{x : \neg\forall RC}{\begin{array}{c} y : \neg C \\ xr_1y \\ \vdots \\ xr_my \end{array}} \qquad (\neg\forall R\text{-elimination})$$

$$\frac{x : \forall RC}{y : C} \qquad (\forall R\text{-elimination})$$

$$\frac{x : (\geq nR)}{\begin{array}{c} \{xr_jy_i | i \leq n, j \leq m\} \cup \\ \{y_i \neq y_j | i, j \leq n, i \neq j\} \end{array}} \qquad (\text{Atleast-restriction})$$

$$\frac{x : (\leq nR)}{S[y/z]} \qquad (\text{Atmost-restriction})$$

Queste ultime sono sottoposte ad una serie di vincoli di applicabilità:

- $\neg\forall R$**-elimination**: $R = \wedge_{i=1}^m r_i$ y dev'essere una variabile nuova e non deve esistere z tale che z sia un R-successore di x e $z : C \in S$.

- $\forall R$**-elimination**: y dev'essere un R-successore di x e $y : C \notin S$.

- **Atleast-restriction**: $R = \wedge_{i=1}^m r_i$, il numero di R-successori di x in S è minore di n e $y_1, \ldots, y_n$ sono variabili nuove.

- **Atmost-restriction**: x ha più di n R-successori in S, e y e z sono due R-successori non separati di x (questa regola *sostituisce* S anzichè aggiungere nuovi elementi ad S.)

Un insieme di regole più vasto per un linguaggio molto più esteso è descritto in [Bresciani *et al.*, 1995].

Un sistema di vincoli S è *completo* se nessuna regola è applicabile. Un sistema completo derivato da S è un *completamento* di S. Un *clash* è un sistema in una delle seguenti forme:

- $\{x : \bot\}$;

function $\mathcal{ALC}$-*Tableau*(S)
    if $A_i \in S$ **and** $\neg A_i \in S$ /* clash */
        then return *False*;
    if $(C_1 \wedge C_2) \in S$ /* $\wedge$-elimination */
        then return $\mathcal{ALC}$-*Tableau*($S \cup \{C_1, C_2\} \backslash \{(C_1 \wedge C_2)\}$);
    if $(\neg\neg C_1) \in S$ /* $\neg\neg$-elimination */
        then return $\mathcal{ALC}$-*Tableau*($S \cup \{C_1\} \backslash \{(\neg\neg C_1)\}$);
    if $(C_1 \vee C_2) \in S$ /* $\vee$-elimination */
        then return $\mathcal{ALC}$-*Tableau*($S \cup \{C_1\} \backslash \{(C_1 \vee C_2)\}$) **or**
                        $\mathcal{ALC}$-*Tableau*($S \cup \{C_2\} \backslash \{(C_1 \vee C_2)\}$);
    for any $\neg\forall r C_{rj} \in S$ **do** /* branch expanded */
        if not $\mathcal{ALC}$-*Tableau*($\{\neg C_{rj}\} \cup \bigcup_{\forall r C_{ri} \in S} \{C_{ri}\}$)
            then return *False*;
    return *True*;

Figura 3.6: Schema dell'algoritmo Tableau-based per $\mathcal{ALC}$.

- $\{x : A_i, x : \neg A_i\}$;
- $\{x : (\leq nR)\} \cup$
  $\{x r_j y_i | i \leq n+1, j \leq m\} \cup$
  $\{y_i \neq y_j | i, j \leq n+1, i \neq j\}$,
  dove $R = \wedge_{i=1}^{m} r_i$;

Si dimostra che un sistema di vincoli completo S è soddisfacibile se e solo se non contiene clash. La presenza di un clash individua quindi l'insoddisfacibilità di S. Intuitivamente, un clash può essere paragonato ad un ramo chiuso nei tableau modali.

Vari algoritmi tableau-based per logiche terminologiche sono descritti in [Hollunder *et al.*, 1990; Baader e Hollunder, 1991; Donini *et al.*, 1992; Baader *et al.*, 1994; Bresciani *et al.*, 1995]. Presentiamo qui in dettaglio (Figura 3.6) una versione funzionale per la $\mathcal{ALC}$ [Baader e Hollunder, 1991]. La procedura viene inizialmente invocata dalla chiamata "$\mathcal{ALC}$-*Tableau*($\{C\}$)". (Per brevità di rappresentazione, supponiamo C in NNF. Nella pratica vanno gestite tutte le possibili combinazioni equivalenti $\neg\vee$, $\neg\supset$, $\neg\wedge$, $\supset$.) La procedura agisce ricorsivamente, secondo i passi seguenti:

- (**Clash**) Se A_i e $\neg A_i$ occorrono entrambi in S per qualche indice i, allora il S contiene un clash e la procedura restituisce $False$.

- ($\wedge$-**elimination**) Altrimenti, se $(C_1 \wedge C_2) \in S$, allora viene sostituita con C_1, C_2 all'interno di S e $\mathcal{ALC}$-*Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\vee$ e $\neg\supset$.

- ($\neg\neg$-**elimination**) Altrimenti, se $\neg\neg C_1 \in S$, allora viene sostituita con C_1 all'interno di S e $\mathcal{ALC}$-*Tableau* viene invocato ricorsivamente sull'insieme risultante.

- ($\vee$-**elimination**) Altrimenti, se $(C_1 \vee C_2) \in S$, allora dapprima viene sostituita con C_1 all'interno di S e $\mathcal{ALC}$-*Tableau* viene invocato ricorsivamente sull'insieme risultante. Se la chiamata restituisce $True$, allora tale valore viene restituito a sua volta. Altrimenti, $(C_1 \wedge C_2)$ viene sostituita con C_2 all'interno di S e $\mathcal{ALC}$-*Tableau* viene invocato ricorsivamente sull'insieme risultante. Similmente avviene per $\neg\wedge$ e $\supset$.

- ($\neg\forall R$-**elimination**) Se nessuno dei precedenti casi si verifica, allora S è un insieme di formule di tipo A_i, $\neg A_i$, $\forall r C_{ri}$ o $\neg\forall r C_{rj}$. Allora, per ogni formula del tipo $\forall r \neg C_{rj}$ in S, $\mathcal{ALC}$-*Tableau* viene invocato sull'insieme $\{\neg C_{rj}\}$ unito all'insieme delle formule C_{ri} tali che $\forall r C_{ri} \in S$. Qualora una chiamata ricorsiva restituisca *False*, allora anche la chiamata principale restituisce *False*. Altrimenti restituisce *True*.

L'algoritmo di Figura 3.6 essenzialmente applica le regole tableau seguendo l'ordine "$\wedge$, $\neg\neg$, $\vee$, $\neg\forall$, $\forall$". Si noti l'analogia, nel senso indicato dalla Tabella 2.5, con la procedura *K(m)-tableau* di Figura 3.5. Anche in questo caso le regole sono applicate in modo distruttivo: ad ogni applicazione la formula "origine" viene rimossa dal ramo, in quanto non piú necessaria. L'uso delle variabili individuali x,y e z, in questo caso, è *implicito* nella procedura. Infatti, nel passo "$\neg\forall R$-elimination", la procedura "salta" da un individuo corrente ad un suo r-successore.

Nonostante un gran numero di formalismi ed algoritmi tableau-based siano stati presentati, sono relativamente poche le procedure effettivamente implementate. Siamo a conoscenza dell'esistenza di tre procedure:

1. TABLEAU [5], implementato da B. Nebel e in seguito migliorato da E. Franconi, è un'implementazione diretta in LISP dell'algoritmo di Figura 3.6.

2. KRIS, [Baader *et al.*, 1994; Baader e Hollunder, 1991] è un sistema di ragionamento tableau-based in LISP, ottimizzato ed ingegnerizzato, che implementa un'estensione funzionale dell'algoritmo di Figura 3.6.

3. CRACK [Bresciani *et al.*, 1995] è un sistema di ragionamento tableau-based in LISP, ottimizzato ed ingegnerizzato, basato su una struttura modulare anzichè funzionale. Rispetto a KRIS ha un linguaggio molto più espressivo, ma non è polinomiale in spazio (cioè può richiedere al caso peggiore una quantità esponenziale di memoria).

3.3.2 Altri algoritmi

I sistemi di ragionamento per logiche terminologiche attualmente reperibili in letteratura sono basati su due tipologie distinte di algoritmi, quelli *strutturali* e quelli basati su *regole*, o tableau-based. Avendo già analizzato questi ultimi nella sezione precedente, aggiungiamo una descrizione sommaria dei primi.

Gli algoritmi strutturali derivano dalla tradizione dei linguaggi a frames e reti semantiche [Brachman e Levesque, 1984]. Il calcolo si basa sulla struttura sintattica dei termini del linguaggio. Le due

[5] Il vero nome della procedura è $\mathcal{ALC}$. È qui convenzionalmente chiamata TABLEAU per evitare confusioni con la logica $\mathcal{ALC}$ stessa.

procedure fondamentali, di verifica della sussunzione tra termini e di riconoscimento di istanze di concetti, si basano su varianti di tecniche di confronto tra alberi sintattici. Viene definita allo scopo una forma normale in cui i termini devono essere rappresentati per poter applicare le procedure. La struttura di un termine è quella di un albero i cui nodi sono gli operatori e le foglie concetti e ruoli atomici. Un algoritmo strutturale esegue quindi un confronto tra alberi.

Aumentando l'espressività dei linguaggi terminologici, aumenta drasticamente la complessità computazionale dell'inferenza. É quindi talvolta necessario scegliere tra algoritmi *completi*, ma poco efficienti, e algoritmi più efficienti, ma *incompleti*. In quest'ottica citiamo due sistemi con approcci opposti:

1. CLASSIC [Patel-Schneider *et al.*, 1991], che implementa un calcolo completo, ma il cui linguaggio terminologico non comprende disgiunzione e negazione (e quindi le inferenze possibili non presentano componente proposizionale);

2. LOOM [MacGregor, 1991] che utilizza un linguaggio terminologico molto espressivo, ma che adotta algoritmi incompleti (e quindi non garantisce di arrivare a produrre un risultato).

Parte II

Teoria

Capitolo 4

Un nuovo approccio

```
PROOF TECHNIQUE #1 - 'Proof By Induction'
 1.Obtain a large power transformer.
 2.Find someone who does not believe your theorem.
 3.Get this person to hold the terminals on the
   HV side of the transformer.
 4.Apply 25000 volts AC to the LV side of the
   transformer.
 5.Repeat step (4) until they agree with the
   theorem.
```

[Anonimo]

In questo capitolo presentiamo le intuizioni e i fondamenti teorici di un nuovo approccio per la realizzazione di procedure di decisione per logiche modali e terminologiche. Il capitolo è organizzato come segue. Nella Sezione 4.1 presentiamo alcune nuove definizioni. Nella Sezione 4.2 dimostriamo un paio di fondamentali teoremi, comuni pressochè a tutte le logiche di nostro interesse. Nella Sezione 4.3 dimostriamo una serie di teoremi specifici per la logica K(m). Nella Sezione 4.4 illustriamo un formalismo, corretto e completo, comune a tutte le logiche modali normali $\mathcal{N}$, sulla base del quale costruire procedure di decisione. Nella Sezione 4.5 suggeriamo come generalizzare il formalismo della Sezione precedente ad altre logiche.

4.1 Definizioni

Introduciamo qui alcuni nuovi concetti. Tutte le definizioni ed i risultati presentati nelle Sezioni 4.1 e 4.2 valgono per ogni logica modale normale $\mathcal{L} \in \mathcal{N}$.

Definizione 4.1.1 (atomo,letterale) *Un* **atomo** *è ogni formula in $\Lambda_{\Box m}$ che non può essere decomposta in modo proposizionale, cioè ogni formula il cui connettivo principale non è proposizionale. Un* **letterale** *è un atomo o la negazione di un atomo.*

Esempi di atomi sono A_1, $\Box_1(A_1 \vee \neg A_2)$ e $\Box_2(\Box_1 A_1 \vee \neg A_2)$. Esempi di letterali sono A_1, $\neg A_2$, $\neg\Box_1(A_1 \vee \neg A_2)$ e $\neg\Box_2(\Box_1 A_1 \vee \neg A_2)$. Data una formula φ, un atomo [letterale] è un atomo *top-level* se occorre in φ sotto l'influenza di nessun operatore modale. $Atoms^0(\varphi)$ è l'insieme degli atomi top-level di φ.

Definizione 4.1.2 (assegnamento di verità) *Un* **assegnamento di verità** μ *per una formula* $\varphi \in \Lambda_{\Box m}$ *è un assegnamento di valori di verità agli atomi top-level di* φ*:*

$$
\begin{array}{lll}
\mu & = \{ & \Box_1\alpha_{11} := T, \ldots, \Box_1\alpha_{1N_1} := T, \\
 & & \Box_1\beta_{11} := F, \ldots, \Box_1\beta_{1M_1} := F, \\
 & & \Box_2\alpha_{21} := T, \ldots, \Box_2\alpha_{2N_2} := T, \\
 & & \Box_2\beta_{21} := F, \ldots, \Box_2\beta_{2M_2} := F, \\
 & & \ldots \\
 & & \Box_m\alpha_{m1} := T, \ldots, \Box_m\alpha_{mN_m} := T, \\
 & & \Box_m\beta_{m1} := F, \ldots \\
 & & A_1 := T, \ldots, A_R := T, \\
 & & A_{R+1} := F, \ldots, A_S := F\},
\end{array}
$$

dove ":=" è l'operatore di assegnazione. Chiameremo **parziale** *un assegnamento di verità* μ *per* φ *che assegni valori di verità ad un sottoinsieme stretto di* $Atoms^0(\varphi)$*.*

Ad atomi top-level identici viene sempre assegnato lo stesso valore di verità. Ad atomi diversi — anche logicamente equivalenti, come, ad Esempio $\Box_2(\varphi_1 \vee \varphi_2)$ e $\Box_2(\varphi_2 \vee \varphi_1)$ oppure $\Box_2\varphi_1$ e $\Box_2(\varphi_1 \wedge \varphi_1)$

— possono venir assegnati valori di verità diversi. Si noti che i simboli "$\diamond_r$", usati normalmente come abbreviazioni di "$\neg\Box_r\neg$", non fanno parte del linguaggio. Questo permette di evitare assegnamenti quali $\mu = \{\Box_r\alpha := T, \diamond_r\neg\alpha := T, \ldots\}$ che sono intrinsecamente insoddisfacibili.

Intuitivamente, se consideriamo φ come una formula proposizionale negli atomi in $Atoms^0(\varphi)$, la Definizione 4.1.2 coincide con la definizione di assegnamento di verità nella logica proposizionale. In generale un assegnamento di verità per $\varphi \in \Lambda_{\Box m}$ può assegnare valori anche anche ad atomi non appartenenti a $Atoms^0(\varphi)$. Tali valori sono *indifferenti* per φ. Per esempio, $\mu = \{\Box_1 A_1 := T, \Box_1 A_2 := F\}$ è un assegnamento di verità per $\varphi = \Box_1 A_1$.

Notazionalmente, chiamiamo $\parallel \mu \parallel$ il numero di elementi del'insieme μ. Se $\mu_2 \subseteq \mu_1$, diciamo che μ_1 *estende* μ_2 e che μ_2 *sussume* μ_1.

Definizione 4.1.3 (assegnamento ristretto a r) *La restrizione di un assegnamento μ all'indice r (o* **assegnamento ristretto a r***) è l'assegnamento:*

$$\begin{array}{lll} \mu^r & = \{ & \Box_r\alpha_{r1} := T, \ldots, \Box_r\alpha_{rN_r} := T, \\ & & \Box_r\beta_{r1} := F, \ldots, \Box_r\beta_{rM_r} := F\} \end{array}$$

dato dal sottoinsieme di μ riguardanti gli atomi top-level nella forma $\Box_r\psi$, $1 \leq r \leq m$.

Banalmente, μ^r sussume μ.

Qualora questo non causi ambiguità, useremo spesso "assegnamento" al posto di "assegnamento di verità". Useremo le lettere greche μ, η per rappresentare assegnamenti. Inoltre, scriveremo spesso un assegnamento μ come una formula: [1]

$$\mu = \bigwedge_i \Box_1\alpha_{1i} \wedge \bigwedge_j \neg\Box_1\beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m\alpha_{mi} \wedge \bigwedge_j \neg\Box_m\beta_{mj} \wedge \gamma, \quad (4.1)$$

[1]Si noti, in questo senso, la differenza tra $\parallel \mu \parallel$ e $|\mu|$. La prima rappresenta il numero di atomi top-level assegnati da μ. La seconda il numero delle sottoformule di μ, inteso come formula.

dove le $\Box_r \alpha_{ri}$ sono gli atomi modali posti a T, le $\Box_r \beta_{ri}$ sono gli atomi modali posti a F, e $\gamma = \bigwedge_{k=1}^{R} A_k \ \wedge \bigwedge_{h=R+1}^{S} \neg A_h$ è la congiunzione di letterali proposizionali. Si noti che γ è sempre soddisfacibile. In modo analogo, l'assegnamento ristretto μ^r è rappresentato dalla formula:

$$\mu^r = \bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg \Box_r \beta_{rj}. \tag{4.2}$$

Si noti anche che T rappresenta un assegnamento vuoto. Diremo che μ e μ^r sono $\mathcal{L}$-soddifacibili intendendo che le corrispondenti formule (4.1) e (4.2) sono $\mathcal{L}$-soddifacibili.

Esempio 4.1.1 Si consideri la seguente formula Λ_2 di profondità 1:

$$\begin{array}{rl}
\varphi = & \{\underline{\neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)} \ \vee A_1 \vee A_5\} \ \wedge \\
& \{\underline{\neg A_2} \vee \neg A_5 \vee \Box_2(\neg A_2 \vee \neg A_4 \vee \neg A_3)\} \ \wedge \\
& \{A_1 \vee \underline{\Box_2(\neg A_4 \vee A_5 \vee A_2)} \ \vee A_2\} \quad \wedge \\
& \{\neg\Box_2(A_4 \vee \neg A_3 \vee A_1) \ \vee \underline{\neg\Box_1(A_4 \vee \neg A_2 \vee A_3)} \ \vee \neg A_5\} \ \wedge \\
& \{\neg A_3 \vee A_1 \vee \underline{\Box_2(\neg A_4 \vee A_5 \vee A_2)}\} \quad \wedge \\
& \{\underline{\Box_1(\neg A_5 \vee A_4 \vee A_3)} \ \vee \Box_1(\neg A_1 \vee A_4 \vee A_3) \vee \neg A_1\} \ \wedge \\
& \{A_1 \vee \underline{\Box_1(\neg A_2 \vee A_1 \vee A_4)} \ \vee A_2\}
\end{array}$$

Si consideri il seguente assegnamento (parziale) μ, che assegna a T i letterali sottolineati nella formula precedente:

$$\begin{array}{rcll}
\mu & = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \ \wedge & \\
& & \Box_1(\neg A_2 \vee A_1 \vee A_4) \ \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\
& & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \ \wedge & \\
& & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \ \wedge & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\
& & \Box_2(\neg A_4 \vee A_5 \vee A_2) \ \wedge & [\bigwedge_i \Box_2 \alpha_{2i}] \\
& & \neg A_2. & [\gamma]
\end{array}$$

Si noti che le due occorrenze di $\Box_2(\neg A_4 \vee A_5 \vee A_2)$ alle righe 3 e 5 di φ sono entrambe assegnate a T. μ dà origine ai due assegnamenti ristretti:

$$\begin{array}{rcll}
\mu^1 & = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \ \wedge & \\
& & \Box_1(\neg A_2 \vee A_1 \vee A_4) \ \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\
& & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \ \wedge & \\
& & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\
\mu^2 & = & \Box_2(\neg A_4 \vee A_5 \vee A_2) & [\bigwedge_i \Box_2 \alpha_{2i}].
\end{array}$$

$\Box$

Diamo ora una definizione fondamentale.

Definizione 4.1.4 (Soddisfacibilità proposizionale, $\models_p$) *Un assegnamento μ per φ* **soddisfa proposizionalmente** *φ, scritto "$\mu \models_p \varphi$", se e solo se rende φ vera, cioè, se per tutte le sub-formule top-level φ_1, φ_2 di φ, si ha:*

$$\begin{array}{lcl} \mu \models_p \varphi_1, \ \varphi_1 \in Atoms^0(\varphi) & \Longleftrightarrow & \{\varphi_1 := T\} \subseteq \mu; \\ \mu \models_p \neg\varphi_1 & \Longleftrightarrow & \mu \not\models_p \varphi_1; \\ \mu \models_p \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & \mu \models_p \varphi_1 \text{ e } \mu \models_p \varphi_2. \end{array}$$

Si noti che, $\mu_1 \models_p \mu_2$ se e solo se $\mu_1 \supseteq \mu_2$, per ogni coppia di assegnamenti μ_1 e μ_2.

Intuitivamente, se consideriamo φ come una formula proposizionale nei suoi atomi top-level, allora "$\models_p$" corrisponde alla definizione standard di soddisfacibilità in logica proposizionale. Di conseguenza, è possibile estendere a $\models_p$ tutte le definizioni e proprietà relative alla soddisfacibilità in tale logica. In particolare, per ogni coppia di formule φ_1 e φ_2, diciamo che $\varphi_1 \models_p \varphi_2$ se e solo se $\mu \models_p \varphi_2$ per ogni μ tale che $\mu \models_p \varphi_1$ (questo ci permette di considerare μ indifferentemente come formula o come insieme). Diciamo inoltre che $\models_p \varphi$ (φ proposizionalmente valida) se e solo se $\mu \models_p \varphi$ per ogni possibile assegnamento μ per φ. [2] È facile verificare che $\varphi_1 \models_p \varphi_2$ se e solo se $\models_p \varphi_1 \supset \varphi_2$, e quindi che $\models_p \varphi_1 \equiv \varphi_2$ se e solo se $\varphi_1 \models_p \varphi_2$ e $\varphi_2 \models_p \varphi_1$.

Si noti che la nozione $\models_p$ è più forte di $\models_\mathcal{L}$, cioè che se $\varphi_1 \models_p \varphi_2$, allora $\varphi_1 \models_\mathcal{L} \varphi_2$, ma se $\varphi_1 \models_\mathcal{L} \varphi_2$, non necessariamente $\varphi_1 \models_p \varphi_2$. Per esempio, $\Box_1(A_1 \wedge A_2) \models_\mathcal{L} \Box_1 A_1 \wedge \Box_1 A_2$, ma $\Box_1(A_1 \wedge A_2) \not\models_p \Box_1 A_1 \wedge \Box_1 A_2$.

Il seguente lemma segue dalle definizioni di $\models_\mathcal{L}$ e $\models_p$:

Lemma 4.1.1 *Sia μ un assegnamento su $Atoms^0(\varphi)$, e sia M, u una $\mathcal{L}$-situazione tale che $M, u \models_\mathcal{L} \mu$. Allora si ha:*

$$M, u \models_\mathcal{L} \varphi \Longleftrightarrow \mu \models_p \varphi.$$

[2] Per questo motivo, quando si dice $\mu \models_p \varphi$, è indifferente intendere μ come assegnamento o come formula.

Prova Per induzione su φ.

- $\underline{\varphi \text{ atomo}}$:

$$M, u \models_{\mathcal{L}} \varphi \quad \Longleftrightarrow \quad \mu = \ldots \wedge \varphi \wedge \ldots \quad \Longleftrightarrow \quad \mu \models_p \varphi.$$

- $\underline{\varphi = \neg\varphi_1}$:

$$\begin{array}{ll} M, u \models_{\mathcal{L}} \neg\varphi_1 & \Longleftrightarrow [\text{Definizione di } \models_{\mathcal{L}}] \\ M, u \not\models_{\mathcal{L}} \varphi_1 & \Longleftrightarrow [\text{Ipotesi Induttiva}] \\ \mu \not\models_p \varphi_1 & \Longleftrightarrow [\text{Definizione di } \models_p] \\ \mu \models_p \neg\varphi_1. & \end{array}$$

- $\underline{\varphi = \varphi_1 \wedge \varphi_2}$:

$$\begin{array}{llll} M, u \models_{\mathcal{L}} \varphi_1 \wedge \varphi_2 & & & \Longleftrightarrow [\text{Definizione di } \models_{\mathcal{L}}] \\ M, u \models_{\mathcal{L}} \varphi_1 & e & M, u \models_{\mathcal{L}} \varphi_2 & \Longleftrightarrow [\text{Ipotesi Induttiva}] \\ \mu \models_p \varphi_1 & e & \mu \models_p \varphi_2 & \Longleftrightarrow [\text{Definizione di } \models_p] \\ \mu \models_p \varphi_1 \wedge \varphi_2. & & & \end{array}$$

Q.E.D.

Un assegnamento *parziale* μ soddisfa proposizionalmente φ se tutti gli assegnamenti per φ che estendono μ soddisfano proposizionalmente φ. Per esempio, se $\varphi = \Box_1\varphi_1 \vee \neg\Box_2\varphi_2$, allora $\mu = \{\Box_1\varphi_1 := T\}$ soddisfa proposizionalmente φ, in quanto sia $\{\Box_1\varphi_1 := T, \Box_2\varphi_2 := T\}$ sia $\{\Box_1\varphi_1 := T, \Box_2\varphi_2 := F\}$ soddisfano proposizionalmente φ.

Definizione 4.1.5 (assegnamento indotto) *Data una $\mathcal{L}$-situazione M, u e una formula φ, l'assegnamento per φ* **indotto** *da M, u è definito come segue:*

$$\begin{array}{rl} [M, u]^{\varphi} = & \{\varphi_i := T \mid \varphi_i \in Atoms^0(\varphi) \text{ } and \text{ } M, u \models_{\mathcal{L}} \varphi_i\} \cup \\ & \{\varphi_i := F \mid \varphi_i \in Atoms^0(\varphi) \text{ } and \text{ } M, u \not\models_{\mathcal{L}} \varphi_i\}. \end{array}$$

Intuitivamente, $[M, u]^{\varphi}$ è l'assegnamento dato dai valori di verità assegnati da M, u agli atomi top-level di φ. È immediato verificare le seguenti proprietà.

Proprietà 4.1.1 $M, u \models_{\mathcal{L}} [M, u]^{\varphi}$, per ogni formula φ.

Proprietà 4.1.2 $[M, u]^{\mu} = \mu$, per ogni assegnamento μ.

Proprietà 4.1.3 $[M, u]^{\varphi} \models_p \varphi \iff [M, u]^{\varphi \wedge \psi} \models_p \varphi$, per ogni formula φ e ψ.

Proprietà 4.1.4 $[M, u]^{\varphi} = [M, u]^{\neg\varphi}$, per ogni formula φ.

Gli assegnamenti indotti giocano un ruolo determinante nelle prove dei teoremi di correttezza e completezza dei nostri algoritmi, come risulterà evidente nella Sezione 4.2.

4.2 Risultati: $\models_{\mathcal{L}}$ via $\models_p$

In questa sezione presentiamo tre fondamentali teoremi, che garantiscono la correttezza e completezza dei nostri algoritmi di decisione.

Per primo dimostriamo il seguente:

Lemma 4.2.1 *Data una formula φ e una $\mathcal{L}$-situazione M, u, si ha:*

$$M, u \models \varphi \iff [M, u]^{\varphi} \models_p \varphi.$$

Prova Per induzione su φ.

- $\underline{\varphi \text{ atomo:}}$ $[M, u]^{\varphi}$ è $\{\varphi := T\}$. Quindi la tesi segue dalla definizione di $\models_p$.

- $\underline{\varphi = \neg\varphi_1:}$

$$\begin{array}{ll}
M, u \models_{\mathcal{L}} \neg\varphi_1 & \iff [\text{Definizione di } \models_{\mathcal{L}}] \\
M, u \not\models_{\mathcal{L}} \varphi_1 & \iff [\text{Ipotesi Induttiva}] \\
[M, u]^{\varphi_1} \not\models_p \varphi_1 & \iff [\text{Proprietà 4.1.4}] \\
[M, u]^{\neg\varphi_1} \not\models_p \varphi_1 & \iff [\text{Definizione di } \models_p] \\
[M, u]^{\neg\varphi_1} \models_p \neg\varphi_1. &
\end{array}$$

- $\underline{\varphi = \varphi_1 \wedge \varphi_2}$:

$$
\begin{array}{ll}
M, u \models_{\mathcal{L}} \varphi_1 \wedge \varphi_2 & \Longleftrightarrow [\text{Definizione di } \models_{\mathcal{L}}] \\
M, u \models_{\mathcal{L}} \varphi_1 \;\; e & \Longleftrightarrow [\text{Ipotesi Induttiva}] \\
M, u \models_{\mathcal{L}} \varphi_2 & \\
[M, u]^{\varphi_1} \models_p \varphi_1 \;\; e & \Longleftrightarrow [\text{Proprietà 4.1.3}] \\
[M, u]^{\varphi_2} \models_p \varphi_2 & \\
[M, u]^{\varphi_1 \wedge \varphi_2} \models_p \varphi_1 \;\; e & \Longleftrightarrow [\text{Definizione di } \models_p] \\
[M, u]^{\varphi_1 \wedge \varphi_2} \models_p \varphi_2 & \\
[M, u]^{\varphi_1 \wedge \varphi_2} \models_p \varphi_1 \wedge \varphi_2. &
\end{array}
$$

Q.E.D.

Come conseguenza del Lemma 4.2.1, otteniamo il seguente:

Teorema 4.2.1 *Una formula modale φ è $\mathcal{L}$-soddisfacibile se e solo se esiste un assegnamento $\mathcal{L}$-soddisfacibile μ, anche parziale, tale che $\mu \models_p \varphi$.*

Prova

- Se: $\mu \models_p \varphi$ e $M, u \models_{\mathcal{L}} \mu$, per qualche $\mathcal{L}$-situazione M, u. Quindi, per ogni $\varphi_1, \varphi_2 \in Atoms^0(\varphi)$, si ha che, se $\varphi_1 := T \in \mu$, allora $M, u \models_{\mathcal{L}} \varphi_1$, e se $\varphi_2 := F \in \mu$, allora $M, u \not\models_{\mathcal{L}} \varphi_2$. Quindi, dalla definizione di $\models_{\mathcal{L}}$, segue immediatamente che $M, u \models_{\mathcal{L}} \varphi$.

  Se μ è parziale, allora possiamo estendere μ agli atomi mancanti prendendo i corrispondenti valori da $[M, u]^{\varphi}$. Sia μ' l'assegnamento risultante. Banalmente $M, u \models_{\mathcal{L}} \mu'$ e $\mu' \models_p \varphi$. Quindi $M, u \models_{\mathcal{L}} \varphi$.

- Solo se: $M, u \models_{\mathcal{L}} \varphi$, per qualche M, u. Quindi, chiamando $\mu = [M, u]^{\varphi}$, si ha che $M, u \models_{\mathcal{L}} \mu$ (Proprietà 4.1.1) e $\mu \models_p \varphi$ (Lemma 4.2.1).

Q.E.D.

Il Teorema 4.2.1 riconduce la $\mathcal{L}$-soddisfacibilità di una formula φ a quella di un assegnamento, anche parziale, che la soddisfi proposizionalmente. D'ora in poi, se non specificato, con "assegnamento" intenderemo indifferentemente anche assegnamenti parziali.

Il riferimento ad assegnamenti parziali è molto importante, in quanto permette di restringere drasticamente il numero di assegnamenti μ considerati. Infatti, un assegnamento parziale μ rappresenta tutti i $2^{Atoms^0(\varphi)-\|\mu\|}$ assegnamenti per φ che lo estendono, dove $Atoms^0(\varphi)- \parallel \mu \parallel$ è il numero degli atomi non assegnati da μ. Ad esempio, considerando la formula φ e l'assegnamento parziale μ dell'Esempio 4.3.1, μ rappresenta $2^7 = 128$ assegnamenti.

Il Teorema 4.2.1 può quindi essere riscritto in una nuova forma, molto più interessante dal punto di vista computazionale.

Teorema 4.2.2 *Si considerino una formula modale φ e un qualsiasi insieme di assegnamenti $\{\mu_1, \mu_2, \ldots\}$, anche parziali, tale che*

$$\models_p \varphi \equiv \bigvee_k \mu_k. \tag{4.3}$$

Allora φ è $\mathcal{L}$-soddisfacibile se e solo se almeno uno degli assegnamenti $\mu_j \in \{\mu_1, \mu_2, \ldots\}$ è $\mathcal{L}$-soddisfacibile.

Prova La parte "Se" è ovvia dal Teorema 4.2.1. Supponiamo per ipotesi φ $\mathcal{L}$-soddisfacibile. Dal Teorema 4.2.1, esistono un assegnamento μ e una $\mathcal{L}$-situazione M, u tali che $M, u \models_{\mathcal{L}} \mu$ e $\mu \models_p \varphi$. Dalla (4.3) si ha

$$\mu \models_p \bigvee_k \mu_k.$$

Dalla definizione di $\models_p$, esiste $\mu_r \in \{\mu_1, \mu_2, \ldots\}$ tale che

$$\mu \models_p \mu_j.$$

Dal Teorema 4.2.1 μ_j è quindi soddisfacibile. Q.E.D.

Nella ricerca di un assegnamento, anzichè considerare tutti i $2^{Atoms^0(\varphi)}$ assegnamenti su $Atoms^0(\varphi)$, è sufficiente quindi restringersi ad un qualsiasi insieme di assegnamenti parziali che verifichi la (4.3). Ad esempio, l'insieme degli assegnamenti parziali generati da una procedura di decisione proposizionale, come DPLL, OBDD, KE, ecc.

4.3 Risultati: soddisfacibilità in K(m)

Nel valutare il problema della $\mathcal{L}$-soddisfacibilità di un assegnamento μ, ci soffermiamo qui in modo particolarmente approfondito sul caso K(m). I risultati qui riportati verranno ripresi nel Capitolo 5.

Teorema 4.3.1 *L'assegnamento*

$$\mu = \bigwedge_i \Box_1 \alpha_{1i} \wedge \bigwedge_j \neg\Box_1 \beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m \alpha_{mi} \wedge \bigwedge_j \neg\Box_m \beta_{mj} \wedge \gamma,$$

è K(m)-soddisfacibile se e solo se gli assegnamenti ristretti

$$\mu^r = \bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$$

sono soddisfacibili, per tutti i $\Box_r$.

Prova La parte "Solo se" è ovvia. Per la parte "Se", per ipotesi assumiamo che, per ogni indice r, esista una K(m)-situazione M_r, u_r tale che $M_r, u_r \models_{K(m)} \mu^r$. Possiamo costruire una K(m)-situazione M, u come segue. Si prenda uno stato u in cui γ sia vera (ciò è sempre possibile in quanto γ è soddisfacibile). Per ogni indice r e per ogni stato u_{rj} tale che $\mathcal{R}_r(u_r, u_{rj})$ valga in M_r, si "connetta" u con u_{rj} tramite $\mathcal{R}_r$ (cioè $\mathcal{R}_r(u, u_{rj})$ valga in M). Segue immediatamente che $M, u \models_{K(m)} \mu$. Q.E.D.

Il Teorema 4.3.1 riconduce la soddisfacibilità di un assegnamento μ alla soddisfacibilità delle restrizioni μ^r. L'idea è quella di di suddividere lo spazio di ricerca in tanti tanti sottospazi di ricerca indipendenti, uno per ciascun operatore $\Box_r$.

Teorema 4.3.2 *L'assegnamento ristretto*

$$\mu^r = \bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$$

è K(m)-soddisfacibile se e solo se le formule

$$\varphi^{rj} = \bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj} \tag{4.4}$$

sono K(m)-soddisfacibili, per ogni $\neg\Box_r\beta_{rj}$ *in* μ^r.

Prova

- Se: Assumiamo, per ogni j, esista M_{rj} e $u_{rj} \in M_{rj}$ tale che $M_{rj}, u_{rj} \models_{K(m)} \varphi^{rj}$. Si ha quindi che $M_{rj}, u_{rj} \models_{K(m)} \neg\beta_{rj}$ e $M_{rj}, u_{rj} \models_{K(m)} \alpha_{ri}$, per ogni i. Possiamo quindi costruire una nuova K(m)-situazione M_r, u_r come segue. Si prenda un nuovo stato u_r e, per ogni j, lo si connetta a u_{rj} tramite la relazione $\mathcal{R}_r$ (cioè $\mathcal{R}_r(u_r, u_{rj})$ vale in M_r). Ne segue direttamente che $M_r, u_r \models_{K(m)} \mu^r$.

- Solo se: Supponiamo esista una K(m)-situazione M_r, u_r tale che $M_r, u_r \models_{K(m)} \mu^r$. Quindi, per ogni j, esiste un mondo u_{rj} tale che valga $\mathcal{R}_r(u_r, u_{rj})$ in M_r e $M, u_{rj} \models_{K(m)} \neg\beta_{rj}$. Inoltre, per ogni i, $M, u_{rj} \models_{K(m)} \alpha_{ri}$. Quindi $M, u_{rj} \models_{K(m)} \varphi^{rj}$.

Q.E.D.

Il Teorema 4.3.2 riconduce la soddisfacibilità degli assegnamenti ristretti μ^r a quella di un insieme di formule φ^{rj}. I Teoremi 4.3.1 e 4.3.2 possono essere riuniti in un unico teorema che affermi che μ è K(m)-soddisfacibile se e solo se lo sono tutte le formule φ^{rj}, per ogni r e j. È importante osservare che, per ogni r e j, si ha

$$depth(\varphi^{rj}) \leq depth(\mu^r) - 1 \leq depth(\mu) - 1 \leq depth(\varphi) - 1.$$

Per dimostrare la K(m)-soddisfacibilità di una formula φ, la sequenza dei Teoremi 4.2.1, 4.3.1 e 4.3.2 può quindi essere applicata ricorsivamente la profondità di φ, fino ad ottenere formule puramente proposizionali, i cui assegnamenti sono per definizione soddisfacibili.

Esempio 4.3.1 Nell'esempio 4.3.1 si consideri la formula φ e gli assegnamenti μ, μ^1 e μ^2. μ soddisfa proposizionalmente φ, in quanto rende vero un letterale in ogni clausola. Quindi, per il Teorema 4.2.1, μ è K(m)-soddisfacibile se e solo se φ è K(m)-soddisfacibile.

Per il Teorema 4.3.1 μ è K(m)-soddisfacibile se e solo se sia μ^1 sia μ^2 lo sono. Per il Teorema 4.3.2, μ^2 è banalmente K(m)-soddisfacibile, poiché non contiene "$\Box$" negati, e μ^1 è K(m)-soddisfacibile se e solo se entrambe le formule

$$\begin{array}{rcl}
\varphi^{11} & = & \bigwedge_i \alpha_{1i} \wedge \neg\beta_{11} \\
 & = & (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \ \wedge A_3 \wedge A_1 \wedge \neg A_2, \\
\varphi^{12} & = & \bigwedge_i \alpha_{1i} \wedge \neg\beta_{12} \\
 & = & (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \ \wedge \neg A_4 \wedge A_2 \wedge \neg A_3
\end{array}$$

sono K(m)-soddisfacibili. Poiché lo sono, φ è K(m)-soddisfacibile. $\Box$

4.4 Risultati: soddisfacibilità in $\mathcal{N}$

Analizziamo ora il problema della $\mathcal{L}$-soddisfacibilità per tutte le logiche modali normali $\mathcal{L} \in \mathcal{N}$.

Diamo prima una definizione generale.

Definizione 4.4.1 (decisore proposizionale) *Chiamiamo* **decisore proposizionale** *una funzione computabile totale f che mappa ogni formula $\varphi \in \Lambda$ in una collezione completa di assegnamenti che soddisfino φ, cioè $f(\varphi) = \mathcal{M} = \{\mu_1, \dots, \mu_n\}$, t.c.*

$$\models_p \varphi \equiv \bigvee_j \mu_j.$$

Si noti che, se consideriamo ogni formula $\varphi \in \Lambda$ come una formula proposizionale nei suoi atomi top-level, allora la definizione di decisore proposizionale combacia con quella standard delle procedure di decisione proposizionali presentate nel Capitolo 3.

Definizione 4.4.2 ($\mathcal{L}$-tableau$_f$) *Data una logica $\mathcal{L} \in \mathcal{N}$ e un decisore proposizionale f, un $\mathcal{L}$-tableau$_f$ è il formalismo ottenuto da $\mathcal{L}$-tableau sostituendo le regole proposizionali con la singola regola:*

$$\frac{\sigma : \varphi}{\sigma : \mu_1 \quad \sigma : \mu_2 \quad \dots \quad \sigma : \mu_n} \ (\textit{f-application}),$$

dove $\{\mu_1, \dots, \mu_n\} = f(\varphi)$.

Tutte le altre definizioni relative a $\mathcal{L}$-tableau (chiusura, ecc.) valgono pari pari per $\mathcal{L}$-tableau$_f$. Intuitivamente, usiamo un decisore proposizionale per decomporre "in un colpo solo" una formula φ in una collezione completa di assegnamenti che la soddisfino proposizionalmente. Chiamiamo $\mathcal{L}$-tableau$_{DPLL}$, $\mathcal{L}$-tableau$_{OBDD}$ e $\mathcal{L}$-tableau$_{KE}$ gli $\mathcal{L}$-tableau ottenuti rispettivamente con DPLL, OBDD e KE. Come per i tableau, è poi possibile generare una procedura di decisione tramite un algoritmo di controllo applicato alle regole $\mathcal{L}$-tableau$_f$.

Per provare la correttezza e completezza di $\mathcal{L}$-tableau$_f$, seguiamo passo passo la corrispondente prova per $\mathcal{L}$-tableau presentata nel Capitolo 8 di [Fitting, 1983], introducendo solo le minime modifiche necessarie per gestire le regole "f-application". Poiché le dimostrazioni sono per la maggior parte identiche, riassumiamo brevemente le parti identiche e spieghiamo esplicitamente le parti che è stato necessario modificare.

Per provare la correttezza, introduciamo una variante del Lemma 3.1 in [Fitting, 1983].

Lemma 4.4.1 *Si consideri una logica $\mathcal{L} \in \mathcal{N}$ e un decisore proposizionale f. Sia T un $\mathcal{L}$-tableau$_f$ $\mathcal{L}$-soddisfacibile, e sia T' un $\mathcal{L}$-tableau$_f$ ottenuto da T dall'applicazione di una singola regola $\mathcal{L}$-tableau$_f$ R. Allora T' è a sua volta $\mathcal{L}$-soddisfacibile.*

Prova Se R è un'f-application, allora dalla dimostrazione del Teorema 4.2.2 T' è $\mathcal{L}$-soddisfacibile. Se R è una regola modale, la dimostrazione è identica a quella del Lemma 3.1 in [Fitting, 1983]. Q.E.D.

La correttezza degli $\mathcal{L}$-tableau$_f$ segue direttamente dal Lemma 4.4.

Per provare la completezza, in Figura 4.1 presentiamo una procedura sistematica $\mathcal{L}$-tableau$_f$. è identica alla procedura in [Fitting, 1983], ad eccezione del caso "non-literal" che sostituisce i casi $\wedge/\vee$ ed equivalenti. Introduciamo inoltre una definizione modificata di insiemi "$\mathcal{L}$-downward saturated". [3]

[3] Intraducibile in italiano.

Place $1 : \varphi$ in the origin;
Repeat
Choose a not-finished wff occurrence $\sigma : \varphi$ as high up in the tree as possible;
If (φ is not a propositional literal)
then
For each open branch θ through the occurrence of $\sigma : \varphi$ **do**
Case φ **of**
non-literal:
Compute $f(\varphi) = \{\mu_1 \dots \mu_n\}$;
Split the end of θ into n sub-branches $\theta_1 \dots \theta_n$;
Add each element of μ_i to θ_i, for every i;
$\Box\varphi_1$**:**
For each prefix σ' used in θ and σ' $\mathcal{L}$-accessible from σ **do:**
add $\sigma' : \varphi_1$ to θ;
![$\mathcal{L}$ deontic:] **if** (there is no prefix σ' used in θ and σ' $\mathcal{L}$-accessible from σ)
! **then**
! **let** (k be the smallest integer s.t. $\sigma, k : \varphi_1$ unrestricted in θ)
! add $\sigma, k : \varphi_1$ to θ;
Add a fresh occurrence of $\sigma : \varphi$ at the end of θ;
$\neg\Box\varphi_1$**:**
let (k be the smallest integer s.t. $\sigma, k : \varphi_1$ unrestricted in θ)
add $\sigma, k : \neg\varphi_1$ to θ;
Declare $\sigma : \varphi$ finished;
Until (all branches closed) **or** (all labelled wff occurrences are finished);
if (all branches closed)
then
return φ unsatisfiable;
else
return φ satisfiable;

Figura 4.1: Schema di una procedura sistematica $\mathcal{L}$-tableau$_f$.

Definizione 4.4.3 (Insieme $\mathcal{L}$-downward saturated$_f$) *Data una logica $\mathcal{L} \in \mathcal{N}$ e un decisore proposizionale f, un insieme S di formule etichettate è* **$\mathcal{L}$-downward saturated$_f$** *se e solo se:*

- *Nessun atomo proposizionale A_i è tale che $\sigma : A_i \in S$ per qualche $\sigma : \neg A_i \in S$;*
- *Se un non-letterale $\sigma : \varphi$ è in S, allora esiste un assegnamento μ_i in $f(\varphi)$ t.c. $\sigma : \mu_i \subseteq S$;*
- *Se $\sigma : \Box\varphi_1 \in S$, allora $\sigma' : \varphi_1 \in S$, per ogni σ' $\mathcal{L}$-accessibile da σ; Inoltre, se $\mathcal{L}$ è deontica, allora $\sigma' : \varphi_1 \in S$, per qualche σ' $\mathcal{L}$-accessibile da σ;*
- *Se $\sigma : \neg\Box\varphi_1 \in S$, allora $\sigma' : \neg\varphi_1 \in S$, per qualche σ' $\mathcal{L}$-accessibile da σ;*

Come in precedenza, questa definizione differisce dall'analoga per $\mathcal{L}$-tableau solo per il caso "non-literal". Possiamo ora introdurre una versione modificata del Lemma 6.1 in [Fitting, 1983].

Lemma 4.4.2 *Si consideri una logica $\mathcal{L} \in \mathcal{N}$ e un decisore proposizionale f. Se S è un insieme $\mathcal{L}$-downward saturated$_f$, allora S è $\mathcal{L}$-satisfiable in un modello di Kripke i cui stati sono quelli etichettati dalle formule in S.*

Prova Supponiamo S sia $\mathcal{L}$-downward saturated$_f$. Costruiamo un $\mathcal{L}$-modello $M =< \mathcal{U}, \pi, \mathcal{R} >$ come segue. Sia $\mathcal{U}$ l'insieme degli stati etichettati in S. Per ogni coppia di etichette $\sigma, \sigma' \in \mathcal{U}$, sia $\mathcal{R}$ t.c. $\mathcal{R}(\sigma, \sigma')$ se e solo se σ' è $\mathcal{L}$-accessibile da σ. Tutto questo crea il "frame" $< \mathcal{U}, \mathcal{R} >$. Poiché il frame coinvolge solo stati e accessibilità tra di essi, la prova che $< \mathcal{U}, \mathcal{R} >$ è un $\mathcal{L}$-frame è identica a quella del Lemma 6.1 in [Fitting, 1983].

Definiamo ora π in modo tale che $\pi(A_i, \sigma) = True$ sse $\sigma : A_i \in S$. Per induzione su φ, se $\sigma : \varphi \in S$, allora $M, \sigma \models_{\mathcal{L}} \varphi$:

- $\underline{\varphi \text{ letterale:}}$ identica a quella del Lemma 6.1 in [Fitting, 1983].

- φ non-letterale: dalla Definizione 4.4.3, esiste $\mu_i \in f(\varphi)$ t.c. $\sigma : \mu_i \subseteq S$. Di conseguenza, dall'ipotesi induttiva, $M, \sigma \models_{\mathcal{L}} \mu_i$. Per il Teorema 4.2.2, $M, \sigma \models_{\mathcal{L}} \varphi$.

Q.E.D.

Di conseguenza, abbiamo il seguente teorema di completezza.

Teorema 4.4.1 *Dati una logica $\mathcal{L}$, un decisore proposizionale f e un insieme di formule Γ, se Γ è $\mathcal{L}$-insoddisfacibile, allora esiste un $\mathcal{L}$-tableau$_f$ chiuso per Γ.*

Prova Identica al Teorema 6.2 in [Fitting, 1983], sostituendo "$\mathcal{L}$-downward saturated" con "$\mathcal{L}$-downward saturated$_f$" e "Lemma 6.1" con "Lemma 4.4.2" Q.E.D.

Fondendo il Lemma 4.4 e il Teorema 4.4.1, otteniamo il teorema di correttezza e completezza di $\mathcal{L}$-tableau$_f$:

Corollario 4.4.1 *Dati una logica $\mathcal{L}$, un decisore proposizionale f e un insieme di formula Γ, Γ è $\mathcal{L}$-insoddisfacibile se e solo se esiste un $\mathcal{L}$-tableau$_f$ chiuso per Γ.*

Per quanto riguarda la decidibilità, [Fitting, 1983] (Capitolo 8, Sezione 7) propone una leggera modifica alla procedura sistematica $\mathcal{L}$-tableau che ne garantisce la terminazione, per ogni logica $\mathcal{L}$. Gli stessi argomenti valgono identicamente anche per la procedura in Figura 4.1.

4.5 Oltre $\mathcal{N}$

Il metodo utilizzato nella Sezione 4.4 per passare da $\mathcal{L}$-tableau a $\mathcal{L}$-tableau$_f$ suggerisce un approccio generalizzato.

Si consideri una logica generica $\mathcal{L}$, con l'unica assunzione che dia l'interpretazione standard ai connettivi proposizionali. Supponiamo esista un formalismo tableau corretto e completo (lo chiamiamo anche in questo caso $\mathcal{L}$-tableau), dato dalle regole seguenti

$$\left\{ \frac{\sigma : \varphi_1 \wedge \sigma : \varphi_2}{\sigma : \varphi_1, \sigma : \varphi_2} (\wedge), \quad \frac{\sigma : \varphi_1 \vee \sigma : \varphi_2}{\sigma : \varphi_1 \qquad \sigma : \varphi_2} (\vee), \ \ldots \right\} \bigcup R'. \quad (4.5)$$

dove ogni regola $r \in R'$ è scritto nella forma generale

$$\frac{\sigma : \psi}{\sigma_{11} : \varphi_{11}, \sigma_{12} : \varphi_{12}, \ldots \qquad \ldots \qquad \sigma_{K1} : \varphi_{K1}, \sigma_{K2} : \varphi_{K2}, \ldots} (r) \ ,$$

dove ψ è un *letterale*. [4]

Intuitivamente, le regole sono suddivise in *puramente proposizionali* e *puramente non-proposizionali* (le regole in R'). Come nel caso dei tableau di Fitting, la nostra proposta è quindi di definire $\mathcal{L}$-tableau$_f$ sostituendo le regole proposizionali con l'applicazione di un decisore proposizionale f:

$$\left\{ \frac{\sigma : \varphi}{\sigma : \mu_1 \quad \sigma : \mu_2 \quad \ldots \quad \sigma : \mu_n} (f) \right\} \bigcup R', \qquad (4.6)$$

dove $f(\varphi) = \{\mu_1, \mu_2 \ldots\}$.

A causa della grande varietà di formalismi tableau disponibile in letteratura, fino ad ora [5] non è stato possibile trovare una prova comune di correttezza e completezza per $\mathcal{L}$-tableau$_f$ generali. Si noti comunque che, nelle dimostrazioni di correttezza e completezza, la parte "difficile" della prova consiste tipicamente coinvolge le regole in R', mentre la parte "proposizionale" risulta semplice. Questo permette di formulare un paio di suggerimenti.

1. Se lo scopo è implementare un decisore per una logica $\mathcal{L}$ per la quale un formalismo $\mathcal{L}$-tableau corretto e completo esista già, suggeriamo di definire l'equivalente $\mathcal{L}$-tableau$_f$ e provarne la correttezza e completezza semplicemente adattando le corrispondenti prove per $\mathcal{L}$-tableau. Le considerazioni di cui sopra suggeriscono come questo possa venir fatto con sforzo relativamente lieve.

[4]Nella maggioranza dei formalismi tableau-based, ogni regola $r \in R'$ genera al più un'unica formula.

[5]Febbraio 1997.

2. Se lo scopo è di definire un *nuovo* formalismo per qualche logica $\mathcal{L}$, suggeriamo di definire direttamente un $\mathcal{L}$-tableau$_f$ in quanto, come illustrato nel Capitolo 8, si presta direttamente ad implementazioni molto più efficienti. Le considerazioni di qui sopra suggeriscono come dimostrarne la correttezza e completezza possa venir fatto con uno sforzo sostanzialmente equivalente rispetto al corrispondente $\mathcal{L}$-tableau.

Parte III

Implementazione

Capitolo 5

Una nuova classe di procedure di decisione

"Ove il romanzo giustifica il suo titolo."
[T. Gautier, "Il Capitan Fracassa", titolo del Cap. VII]

Il Teorema 4.2.2 riduce la $\mathcal{L}$-soddisfacibilità di una formula φ a quella dei suoi assegnamenti, per ogni $\mathcal{L} \in \mathcal{N}$. I Teoremi 4.3.1 e 4.3.2 mostrano come ricondurre la K(m)-soddisfacibilità di un assegnamento a quella di formule di profondità minore. Questo processo può essere applicato ricorsivamente sulla profondità della formula, riducendo la profondità delle formule ad ogni ricorsione.

Basandoci su queste osservazioni, è stato possibile concepire una nuova classe di algoritmi per la K(m)-soddisfacibilità, detti genericamente KSAT, che alternano ricorsivamente i due passi seguenti:

1. "ragionamento proposizionale": usando una procedura di decisione proposizionale, trovare un assegnamento μ tale che $\mu \models_p \varphi$;

2. "ragionamento modale": verificare la K(m)-soddisfacibilità di μ generando i corrispondenti assegnamenti ristretti μ_r e formule φ^{rj}.

I due passi precedenti si alternano ricorsivamente sulla profondità della formula, fino ad arrivare ad assegnamenti puramente proposizionali. A ogni livello, il processo viene ripetuto finché o viene trovato un assegnamento soddisfacibile (nel qual caso φ è K(m)-soddisfacibile) oppure nessun assegnamento è più disponibile (nel qual caso φ non è K(m)-soddisfacibile).

In questo capitolo presentiamo le varie versioni di KSAT. La descrizione viene fatta incrementalmente. Nella Sezione 5.1 viene presentata la versione di base. Nella Sezione 5.2 viene presentata la seconda versione, ottenuta dalla precedente introducendo un preordinamento delle formule. Nella Sezione 5.3 viene presentata la terza versione, ottenuta dalla precedente introducendo una particolare fattorizzazione dello spazio di ricerca. Nella Sezione 5.4 viene presentata la quarta versione, ottenuta dalla precedente introducendo un test di soddisfacibilità intermedio su assegnamenti incompleti.

5.1 KSAT versione 1: l'algoritmo di base

La versione base dell'algoritmo KSAT è riportata in Figura 5.1. KSAT prende in ingresso una formula modale φ e restituisce un valore booleano relativo alla la K(m)-soddisfacibilità di φ. KSAT invoca KSAT_W (dove "$_W$" stà per "wff", cioè formula), passando come argomenti φ e l'assegnamento T (cioè, dalla (4.1), l'assegnamento vuoto). KSAT_W cerca un assegnamento K(m)-soddisfacibile che soddisfi φ. Questo avviene ricorsivamente, secondo i passi seguenti:

- (**base**) Se $\varphi = T$, allora μ soddisfa proposizionalmente la versione iniziale di φ. Conseguentemente, se μ è K(m)-soddisfacibile, anche φ lo è. Quindi, KSAT_W invoca $\text{KSAT}_A(\mu)$ (dove "$_A$" significa *A*ssegnamento). KSAT_A restituisce un valore booleano *True* se μ è K(m)-soddisfacibile, *False* altrimenti.

- (**backtrack**) Se $\varphi = F$, allora μ non soddisfa proposizionalmente la versione iniziale di φ, quindi KSAT_W ritorna $False$.

- (**unit**) Se un letterale l occorre in φ come clausola unitaria (o equivalente per formule non-CNF, si veda Sezione 3.1.2) allora a l deve essere assegnato T. A tal scopo, KSAT_W viene

```
function KSAT(φ)
    return KSAT_W(φ, T);

function KSAT_W(φ, μ)
    if φ = T                                          /* base      */
        then return KSAT_A(μ);
    if φ = F                                          /* backtrack */
        then return False;
    if {a unit clause (l) occurs in φ}                /* unit      */
        then return KSAT_W(assign(l, φ), μ ∧ l);
    l := choose-literal(φ);                           /* split     */
    return KSAT_W(assign(l, φ), μ ∧ l)   or
           KSAT_W(assign(¬l, φ), μ ∧ ¬l);

function KSAT_A(⋀_i □_1 α_1i ∧ ⋀_j ¬□_1 β_1j ∧ ... ∧ ⋀_j ¬□_m β_mj ∧ γ)
    for any box index r do
        if not KSAT_AR(⋀_i □_r α_ri ∧ ⋀_j ¬□_r β_rj)
            then return False;
    return True;

function KSAT_AR(⋀_i □_r α_ri ∧ ⋀_j ¬□_r β_rj)
    for any conjunct "¬□_r β_rj" do
        if not KSAT(⋀_i α_ri ∧ ¬β_rj)
            then return False;
    return True;
```

Figura 5.1: KSAT: Versione di base.

invocata ricorsivamente, passando come argomenti la formula restituita da *assign*(l, φ) e l'assegnamento $\mu \wedge l$ (cioè $\mu \cup \{\psi := T\}$ se $l = \psi$ e $\mu \cup \{\psi := F\}$ se $l = \neg\psi$.

- (**split**) Se nessuno dei casi precedenti si verifica, allora la funzione *choose-literal*(φ) restituire con qualche criterio euristico un letterale l non ancora assegnato. Quindi KSAT$_W$ viene dapprima invocato con *assign*(l, φ) e $\mu \wedge l$. Se il risultato è negativo, $\textsc{Ksat}_W$ viene invocato con *assign*$(\neg l, \varphi)$ e $\mu \wedge \neg l$.

Confrontando Figura 5.1 con Figura 3.2, è immediato osservare come $\textsc{Ksat}_W$ sia una variante di una versione non-CNF di DPLL. La differenza fondamentale si trova nel passo "base". Normalmente DPLL viene usato per trovare *un unico* assegnamento che soddisfi la formula in ingresso. Per questo motivo, quando è stato trovato un assegnamento μ tale che $\mu \models_p \varphi$, DPLL restituisce semplicemente $True$. In $\textsc{Ksat}_W$, invece, DPLL viene usato per *generare* assegnamenti (tipicamente un gran numero), la cui soddisfacibilità viene ricorsivamente verificata tramite la funzione $\textsc{Ksat}_A$. $\textsc{Ksat}_A(\mu)$ invoca $\textsc{Ksat}_{AR}(\mu_r)$ (dove"$_{AR}$" significa *A*ssegnamento *R*istretto) per ogni indice r tale che $\Box_r$ occorra in μ. Questo si ripete finché o $\textsc{Ksat}_{AR}$ restituisce un valore negativo (nel cui caso $\textsc{Ksat}_A(\mu)$ restituisce *False*) oppure non vi sono più $\Box_r$ disponibili (nel cui caso $\textsc{Ksat}_A(\mu)$ restituisce *True*). $\textsc{Ksat}_{AR}(\mu_r)$ restituisce $True$ se μ_r è K(m)-soddisfacibile, $False$ altrimenti. $\textsc{Ksat}_{AR}(\mu^r)$ invoca $\textsc{Ksat}(\varphi^{rj})$ per ogni congiunto $\neg\Box_r\beta_{rj}$ in μ_r. Questo si ripete finché o $\textsc{Ksat}$ restituisce un valore negativo (nel cui caso $\textsc{Ksat}_{AR}(\mu^r)$ restituisce *False*) oppure non vi sono più $\neg\Box_r\beta_{rj}$ disponibili (nel cui caso $\textsc{Ksat}_{AR}(\mu^r)$ restituisce *True*).

$\textsc{Ksat}$ *termina* sempre in un numero finito di passi. Infatti, $\textsc{Ksat}$ lavora ricorsivamente su $depth(\varphi)$ (valore finito) e, ad ogni passo, esegue al più un numero finito di passi, in quanto

(i) $\textsc{Ksat}$ invoca $\textsc{Ksat}_A$ al più $2^{|Atoms^0(\varphi)|}$,

(ii) $\textsc{Ksat}_A$ invoca $\textsc{Ksat}_{AR}$ al più m volte,

(iii) $\textsc{Ksat}_{AR}$ invoca $\textsc{Ksat}$ al più $|Atoms^0(\mu^r)|$ volte.

$\textsc{Ksat}$ è inoltre corretto e completo, cioè $\textsc{Ksat}(\varphi)$ restituisce $True$ se φ è K(m)-soddisfacibile, *False* altrimenti. Infatti $\textsc{Ksat}_W$, $\textsc{Ksat}_A$ e $\textsc{Ksat}_{AR}$ implementano direttamente i Teoremi 4.2.2, 4.3.1 e 4.3.2.

È molto importante osservare che non siamo minimamente legati ad usare DPLL come algoritmo proposizionale. Infatti, per il Teorema 4.2.2, ogni procedura di decisione proposizionale in grado di generare un insieme di assegnamenti $\{\mu_1, \mu_2, \ldots\}$ che verifichi la (4.3) è adatta a sostituire DPLL in KSAT. Tra questi, OBDD e KE sono due ottimi potenziali candidati. È inoltre possibile pensare un algoritmo incompleto basato su GSAT.

5.2 KSAT versione 2: ordinamento

Una delle potenziali cause di inefficienza dell'algoritmo di Figura 5.1 è il numero di assegnamenti trovati dalla funzione KSAT_W, che può essere molto grande. Questo è diretta conseguenza del grande numero di atomi modali distinti che possono occorrere all'interno di una formula modale φ.

In generale, una soluzione per ridurre drasticamente il numero degli atomi (e conseguentemente degli assegnamenti) è quella di trattare atomi logicamente equivalenti come distinte occorrenze dello stesso atomo (e quindi attribuire loro sempre gli stessi valori di verità). Sfortunatamente, questo comporterebbe in generale un costo computazionalmente inaccettabile, in quanto verificare l'equivalenza di due atomi φ_1 e φ_2 è equivalente a verificare la soddisfacibilità della formula $\neg\varphi_1 \equiv \varphi_2$, e quindi estremamente oneroso.

Ciononostante, alcune operazioni di preprocessing per collassare assieme atomi banalmente equivalenti possono venir effettuate a basso costo. Per esempio, tutti gli atomi modali possono venire internamente *ordinati* mediante qualche criterio di ordinamento sulle sottoformule (la scelta di tale criterio è irrilevante). Questo fa sì che diverse permutazioni delle stesse sottoformule vengano trasformate in istanze della stessa sottoformula, e come tali vengano sempre assegnati loro valori di verità uguali.

Esempio 5.2.1 Si considerino gli atomi occorrenti nella formula φ nell'Esempio 4.3.1 (e.g. $\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)$ nella prima riga. Per ciascun atomo ci possono essere fino a $3! = 6$ permutazioni logicamente equivalenti, le quali, in caso di ordinamento, vengono convertite tutte nello stesso atomo (e.g., $\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)$ e $\Box_1(\neg A_1 \neg A_3 \vee \vee A_2)$ in $\Box_1(\neg A_1 \vee A_2 \vee \neg A_3)$).

5.3 KSAT versione 3: fattorizzazione

In Figura 5.1, KSAT$_{AR}$ invoca ripetutamente KSAT passando come argomenti formule nella forma $\wedge_i \alpha_{ri} \wedge \neg\beta_{r1}, \ \ldots, \ \wedge_i \alpha_{ri} \wedge \neg\beta_{rM_r}$. Tutte queste formule hanno la congiunzione $\wedge_i \alpha_{ri}$ in comune. (Si noti che il numero M_r e la dimensione medi delle β_{rj} sono gli stessi delle α_{ri}, e che M_r può essere molto grande.) Alla chiamata $j$esima, KSAT cerca un assegnamento η_j che soddisfi $\wedge_i \alpha_{ri} \wedge \neg\beta_{rj}$. Questo viene fatto senza tentare di riutilizzare nessuno degli assegnamenti $\eta_1 \ldots \eta_{j-1}$ trovati precedentemente.

L'idea alla base della Versione 3 è quella di "fattorizzare" la ricerca degli assegnamenti che soddisfino la componente comune $\wedge_i \alpha_{ri}$. Dati $\alpha = \wedge_i \alpha_{ri}$ e l'insieme non vuoto $B = \{\beta_{r1}, \ldots, \beta_{rM_r}\}$, un algoritmo proposizionale viene usato per trovare una sequenza di assegnamenti $\eta_1 \ \eta_2 \ldots$ che soddisfino α. Al k-simo assegnamento η_k, tutte le formule β_{rj} "compatibili" con η_k (i.e., tali che $\eta_k \wedge \neg\beta_{rj}$ sia K(m)-soddisfacibile) vengono rimosse da B. Questo viene ripetuto finché B è vuoto (cioè μ^r soddisfacibile) o nessun altro η_k è disponibile (cioè μ^r insoddisfacibile).

In Figura 5.2 presentiamo una versione rivisitata della funzione KSAT$_{AR}$. Come in precedenza, KSAT$_{AR}$ prende come argomenti un assegnamento ristretto $\mu^r = \bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$ e ritorna il valore booleano $True$ se μ^r è K(m)-soddisfacibile, $False$ altrimenti. Se nessun congiunto nella forma $\neg\Box_r \beta_{rj}$ occorre in μ^r, allora μ^r è soddisfacibile, e quindi KSAT$_{AR}$ restituisce $True$. In caso contrario, KSAT$_{AR}$ invoca la funzione *Incompatible-Subset*, passando come argomenti la formula $\alpha = \bigwedge_i \alpha_{ri}$, l'assegnamento vuoto T l'insieme (sempre non vuoto) $B = \{\beta_{r1}, \ldots, \beta_{rM_r}\}$. *Incompatible-Subset($\alpha, T, B$)* restituisce l'insieme delle formule β_{rj} in B che non sono compatibili con alcun assegnamento η tale che $\eta \models_p \alpha$, cioè tali che $\alpha \wedge \neg\beta_{rj}$ sia

function $\text{KSAT}_{AR}(\bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj})$
  if {there is no conjunct "$\neg\Box_r\beta_{rj}$"}
    then return *True*;
  return *Is-Empty(Incompatible-Subset(*$\bigwedge_i \alpha_{ri}, T, \{\beta_{r1}, \ldots, \beta_{rM_r}\}$*))*;

function *Incompatible-Subset(*α, η, B*)*
  if $\alpha = T$ /* base */
    then return $\{\beta_{rj} \in B \mid \text{KSAT}(\eta \wedge \neg\beta_{rj}) = False\}$;
  if $\alpha = F$ /* backtrack */
    then return B;
  if {a unit clause (l) is in α} /* unit */
    then return *Incompatible-Subset(assign*$(l, \alpha), \eta \wedge l, B$*)*;
  l := *choose-literal(*α*)*; /* split */
  B' := *Incompatible-Subset(assign*$(l, \alpha), \eta \wedge l, B$*)* ;
  if *Is-Empty(*B'*)*
    then return B';
    else return *Incompatible-Subset(assign*$(\neg l, \alpha), \eta \wedge \neg l, B'$*)*;

Figura 5.2: KSAT Versione 3: la nuova funzione KSAT_{AR}

insoddisfacibile. [1] KSAT_{AR} restituisce $True$ se e solo se tale insieme

[1] Più in generale, *Incompatible-Subset(*α, η, B*)* restituisce l'insieme:

$$\{\beta_{rj} \in B \mid \alpha \wedge \eta \wedge \neg\beta_{rj} \text{ not K(m)-satisfiable}\}. \tag{5.1}$$

Se $\eta = T$, allora (5.1) rappresenta l'insieme dei β_{rj} tali che $\alpha \wedge \neg\beta_{rj}$ sia insoddisfacibile, cioè l'insieme richiesto da KSAT_{AR}. Se $\alpha = T$ allora (5.1) si riduce all'insieme dei β_{rj} tali che $\eta \wedge \neg\beta_{rj}$ sia insoddisfacibile, cioè l'insieme restituito dal passo base. Per meglio capire il passo "split", si noti che, per ogni letterale l, (5.1) è uguale a

$$\begin{array}{ll}\{\beta_{rj} \in B \mid \alpha \wedge \eta \wedge l \wedge \neg\beta_{rj} \text{ not K(m)-satisfiable}\} & \cap \\ \{\beta_{rj} \in B \mid \alpha \wedge \eta \wedge \neg l \wedge \neg\beta_{rj} \text{ not K(m)-satisfiable}\}. & \end{array}$$

Sia B' il primo insieme dell'intersezione. Dalle proprietà degli insiemi si ha

$$B' \subseteq B \implies B' \cap \{\beta_{rj} \in B | \ldots\} = B' \cap \{\beta_{rj} \in B' | \ldots\}$$

Di conseguenza (5.1) risulta essere uguale a

$$B' \cap \{\beta_{rj} \in \underline{B'} \mid \alpha \wedge \eta \wedge \neg l \wedge \neg\beta_{rj} \text{ not K(m)-satisfiable}\}.$$

è vuoto. Analogamente a KSAT$_W$, *Incompatible-Subset* cerca di costruire assegnamenti η t.c. $\eta \models_p \alpha$. Ogniqualvolta ne trova uno, tutte le sottoformule β_{rj} compatibili con η vengono rimosse da B. Come in precedenza, questo viene fatto ricorsivamente seguendo i passi seguenti:

- (base) Se $\alpha = T$, allora KSAT viene invocato su tutte le formule $\eta \wedge \neg\beta_{rj}$, per ogni β_{rj} in B. Viene quindi restituito l'insieme delle formule β_{rj} in B non compatibili con η (cioè, tali che KSAT$\eta \wedge \neg\beta_{rj}$) dia *False*).

- (backtrack) Se $\alpha = F$, allora η non soddisfa α. Di conseguenza *Incompatible-Subset* restituisce l'intero insieme B.

- (unit) Se un letterale l occorre in α come clausola unitaria (o forma equivalente per formule non-CNF), l viene aggiunto a η e *Incompatible-Subset* viene invocato ricorsivamente con *assign(*l, α*)*, $\eta \wedge l$ e B.

- (split) Se nessuno dei casi precedenti si verifica, allora *choose-literal(*α*)* restituisce un letterale l non ancora assegnato. La funzione *Incompatible-Subset* viene dapprima invocata passando *assign(*l, α*)*, $\eta \wedge l$ e B, e l'insieme restituito viene chiamato B'. Se B' è vuoto, allora le formule incompatibili sono esaurite, e quindi viene restituito B'. Altrimenti, *Incompatible-Subset* viene invocato con *assign(*$\neg l, \alpha$*)*, $\eta \wedge \neg l$ e B'.

Come in precedenza, è possibile riconoscere in *Incompatible-Subset* una variante di DPLL. La differenza più importante consiste nel fatto che *Incompatible-Subset* restituisce *insiemi di formule* anziché valori di verità. Come in KSAT$_W$, il passo "base" viene modificato, rispetto a DPLL standard, al fine di generare tutti gli assegnamenti. Si noti inoltre l'assimmetria del passo "split": alla seconda chiamata ricorsiva, *Incompatible-Subset* viene invocata passando l'insieme B' anzichè B, $B' \subseteq B$.

Da questo deriva l'assimmetria del passo "split".

Esempio 5.3.1 Si considerino φ, μ^1, φ^{11} e φ^{12} come negli Esempi 4.3.1 e 5.2.1. Abbiamo che

$$\begin{aligned}
\bigwedge_i \alpha_{1i} &= (\underline{\neg A_5} \vee A_4 \vee A_3) \ \wedge (\underline{\neg A_2} \vee \underline{\underline{A_1}} \vee A_4) \\
\neg\beta_{11} &= A_3 \wedge A_1 \wedge \neg A_2 \\
\neg\beta_{12} &= \neg A_4 \wedge A_2 \wedge \neg A_3.
\end{aligned}$$

Supponiamo che *Incompatible-Subset* selezioni in sequenza i letterali $\neg A_5$ e $\neg A_2$, trovando quindi l'assegnamento $\eta_1 = \neg A_5 \wedge \neg A_2$, che soddisfa α. $\eta_1 \wedge \neg\beta_{11}$ è soddisfacibile ma $\eta_1 \wedge \neg\beta_{12}$ no. Al passo "base" della chiamata ricorsiva più bassa, *Incompatible-Subset* restituisce l'insieme $\{\beta_{12}\}$. Quindi alla chiamata intermedia, per il secondo ramo del passo "split", la funzione viene invocata con A_2, trovando l'assegnamento $\eta_2 = \neg A_5 \wedge A_2 \wedge A_1$. Poichè $\eta_2 \wedge \neg\beta_{12}$ è soddisfacibile, *Incompatible-Subset* restituisce l'insieme vuoto. Quindi μ^1, μ e φ risultano essere K(2)-soddisfacibili. $\square$

La fattorizzazione delle componente $\bigwedge_i \alpha_{ri}$ ha un'importante motivazione teorica. Il Teorema 4.3.2 riconduce l'esistenza di una situazione ad albero M, u per μ^r all'esistenza di una situazione M_j, u_j per $\varphi^{rj} = \bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}$, per ogni $\neg\Box\beta_{rj} \in \mu^r$. Nulla però dice che le M_j, u_j debbano essere tra loro *distinte*. È possibile avere situazioni che soddisfino più formule φ^{rj}. La fattorizzazione di $\bigwedge_i \alpha_{ri}$ corrisponde proprio a fattorizzare più possibile i rami modali dell'albero M. Conseguentemente, questo riduce drasticamente il branching modale di M, e conseguentemente la dimensione stessa di M.

5.4 KSAT versione 4: test intermedi

Nonostante i notevoli miglioramenti introdotti dall'ordinamento interno delle formule (si veda Sezione 5.2), il numero di assegnamenti trovati può essere ancora troppo grande. La versione 4 di KSAT nasce dalla considerazione empirica che molti assegnamenti sono "banalmente" insoddisfacibili, cioè resterebbero insoddisfacibili anche se venissero rimossi alcuni (molti) dei loro congiunti. (Con maggior precisione formale, possiamo dire che tali assegnamenti sono *sovra-vincolati* [Williams e Hogg, 1994].) Se un assegnamento

```
function KSAT_W(φ, μ)
    if φ = T                                    /* base      */
        then return KSAT_A(μ);
    if φ = F                                    /* backtrack */
        then return False;
    if {a unit clause (l) occurs in φ}          /* unit      */
        then return KSAT_W(assign(l, φ), μ ∧ l);
    if Likely-Unsatisfiable(μ)                  /* inc. ass. check */
        if not KSAT_A(μ)
            then return False;
    l := choose-literal(φ);                     /* split     */
    return KSAT_W(assign(l, φ), μ ∧ l)    or
           KSAT_W(assign(¬l, φ), μ ∧ ¬l);
```

Figura 5.3: KSAT Versione 4: la nuova funzione KSAT_W.

incompleto μ' è insoddisfacibile, sicuramente lo sono anche tutte le sue estensioni $\mu' \wedge \ldots$. Se l'insoddisfacibilità di μ' viene rivelata per tempo, si evita di verificare la soddisfacibilità di tutti (fino a $2^{|Atoms^0(\varphi)|-\|\mu'\|}$) gli assegnamenti che estendono μ'.

Queste considerazioni suggeriscono l'introduzione di un test di soddisfacibilità intermedio sugli assegnamenti "in costruzione", primi di ogni passo "split" (si noti che non c'è alcuna necessità di introdurre test simili prima dei passi "unit"). Questo può esser fatto introducendo all'interno della funzione KSAT_W, tra i passi "unit" e "split", il passo "incomplete assignments check" (Figura 5.3).

Esempio 5.4.1 Si consideri la formula φ dell'Esempio 4.3.1. Supponiamo che, dopo tre chiamate ricorsive, KSAT_W abbia costruito l'assegnamento incompleto:

$$\begin{aligned} \mu' = \; & \Box_1(\neg A_1 \vee A_4 \vee A_3) \wedge \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge \\ & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \end{aligned}$$

(righe 6, 7 e 4 di φ). Se invocato su μ', KSAT_A verifica la K(2)-soddisfacibilità della formula

$$(\neg A_1 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3,$$

che è insoddisfacibile. Fatto questo, non c'è più necessità di selezionare altri letterali, e quindi la procedura effettuerà un backtracking.

È possibile pensare che l'introduzione di una verifica di inconsistenza prima di *ogni* split possa influire negativamente sulle prestazioni globali al caso peggiore. Infatti, poiché in un albero binario il numero dei nodi interni con due figli è uguale a quello delle foglie meno uno, nel caso ipotetico in cui tutti i test intermedi si rivelassero inutili, il numero di chiamate a KSAT_A per chiamata a KSAT potrebbe raddoppiare. Per evitare questo, può valere la pena introdurre una funzione euristica *Likely-Unsatisfiable*, che dia una (rapida) stima della probabilità di (in)soddisfacibilità di μ in funzione di parametri quali il numero di congiunti, il numero di variabili proposizionali, la profondità, ecc. Per esempio, una semplice criterio può essere di effettuare il test solo quando l'ultimo letterale introdotto non è proposizionale. Euristiche più sofisticate potrebbero utilizzare diagrammi di transizione precedentemente tabulati, come quelli che vedremo nei prossimi capitoli. Si noti che, per rendere il test intermedio conveniente, è sufficiente che si eviti in media almeno uno split per ramo.

Capitolo 6

Un nuova procedura di test

Nell'area della deduzione automatica per logiche modali e terminologiche la bibliografia su insiemi di problemi e metodologie di test sembra essere molto limitata. La maggior parte degli autori non presenta risultato empirici [Fitting, 1983; Fitting, 1988; Massacci, 1994; Ohlbach, 1988a; Wallen, 1987; Auffray e Enjalbert, 1992; Enjalbert e del Cerro, 1989]. Alcuni autori provano i loro sistemi su gruppi di singole formule, nella maggior parte dei casi prese da libri di testo [Catach, 1991; Demri, 1995; Goré *et al.*, 1995; Koshimura e Hasegawa, 1994]. Pochi altri usano formule che sono derivate o simulano problemi realistici (si veda, ad esempio, [Baader *et al.*, 1994]).

Per quanto a nostra conoscenza, non esiste in letteratura alcuno studio su classi di formule di test e sulle loro proprietà intrinseche (per esempio, complessità, probabilità di essere soddisfacibili, ecc.). La logica modale S4 rappresenta un'interessante eccezione, in quanto è possibile utilizzare la traduzione di Gödel-Tarski dalla logica proposizionale intuizionista in S4. Un'altra notevole eccezione si può trovare in [Halpern e Moses, 1992; Fagin *et al.*, 1995], dove viene descritta una classe di di formule modali intrinsecamente difficili, in quanto soddisfacibili solo in strutture di Kripke esponenzialmente grandi. (Nel Capitolo 7 presenteremo un test empirico basato su tali formule.)

In Deduzione Automatica con logica proposizionale e del primo ordine, invece, esiste un'ampia bibliografia sia su insiemi di formule di test (si veda, ad esempio, [Pellettier, 1986; Suttner e Sutcliffe, 1995]) e di metodi di generazione di formule di test (si veda, ad esempio, [Buro e Buning, 1992; Mitchell *et al.*, 1992])

In questo capitolo presentiamo una nuova metodologia di test per procedure di decisione per logiche modali e terminologiche che generalizza una metodologia usata comunemente in SAT. Il nuovo metodo è facile da implementare e da usare, e permette un controllo statistico di alcune caratteristiche fondamentali delle formule generate, quali da difficoltà e la probabilità di essere soddisfacibili. Nella Sezione 6.1 descriviamo brevemente il metodo usato in SAT. Nella Sezione 6.2 descriviamo la nostra nuova metodologia per logiche $\mathcal{L} \in \mathcal{N}$. Nella Sezione 6.3 presentiamo un esempio di test, evidenziando le caratteristiche delle formule di test usate. Nella Sezione 6.4, descriviamo come generalizzare tale metodologia a molte altre logiche modali/terminologiche.

6.1 Il metodo "fixed clause-length" per SAT

Presentiamo dapprima il metodo "*fixed clause-length*" per SAT (si veda per esempio [Mitchell *et al.*, 1992; Buro e Buning, 1992]) con clausole di lunghezza 3 (d'ora in poi *3-clause-length*). Dato un numero N di variabili proposizionali, per valori crescenti del numero di clausole L, un numero sufficientemente grande (100, 500, 1000...) di formule 3CNF generate casualmente (random CNF d'ora in poi) vengono generate e date in ingresso alle procedure sotto test. Eseguita la computazione, viene effettuata un'analisi statistica dei valori ottenuti. I risultati statistici, quali le percentuali di formule soddisfacibili, i tempi medi/mediani di CPU e le dimensioni medie/mediane dello spazio di ricerca, vengono rappresentate su un grafo cartesiano in funzione del rapporto L/N. Questo processo può venir ripetuto per differenti numeri di variabili proposizionali. Le formule random 3CNF sono generate come segue:

"... *dati* L *e* N*, una formula random 3SAT viene prodotta gene-*

rando in modo random L clausole di lunghezza 3. Ciascuna clausola viene prodotta scegliendo in modo casuale un insieme di 3 variabili proposizionali, e negando ciascuna di esse con probabilità 0.5. ..." (Da [Mitchell *et al.*, 1992]).

Questo metodo è diventato uno standard nel test di procedure SAT [Buro e Buning, 1992; DIM, 1993]. Il suo successo è dovuto, a nostro avviso, a tre fondamentali prerogative.

1. Le formule 3CNF rappresentano tutte le formule proposizionali. Infatti, come è ben noto [Garey e Johnson, 1979], è possibile convertire tutte le formule proposizionali in forma CNF in modo da preservarne la soddisfacibilità.

2. Le formule 3CNF casuali possono essere generate in funzione di due soli parametri:

   (i) il numero di clausole L;

   (ii) il numero di variabili proposizionali N.

3. I parametri L e N permettono una regolazione approssimativa della probabilità di soddisfacibilità e della difficoltà delle formule casuali generate.

Quest'ultimo aspetto è molto importante e merita qualche ulteriore commento (si osservi, a titolo di esempio, la Figura 6.1). L $[N]$ incrementa [decrementa] *monotonicamente* il livello di *constraintness* [1] delle formule generate. Quindi, variando il rapporto L/N, si passa da una situazione dove le formule sono sotto-vincolate (e quindi prevalentemente soddisfacibili) ad una dove sono sovra-vincolate (e quindi prevelentemente insoddisfacibili). Di conseguenza, il grafico delle percentuali di formule soddisfacibili rivela una transizione da 100% soddifacibili a 100% insoddisfacibili, con il punto di 50% soddisfacibili sempre collocato intorno al valore fisso $L/N \approx 4.3$

[1] Intraducibile in italiano. Stabilisce la quantità di vincoli posta sui possibili modelli di una formula (per una trattazione formale si veda [Williams e Hogg, 1994]). Ad esempio, ogni clausola C stabilisce il vincolo che almeno ad uno dei letterali di C dev'essere assegnato T.

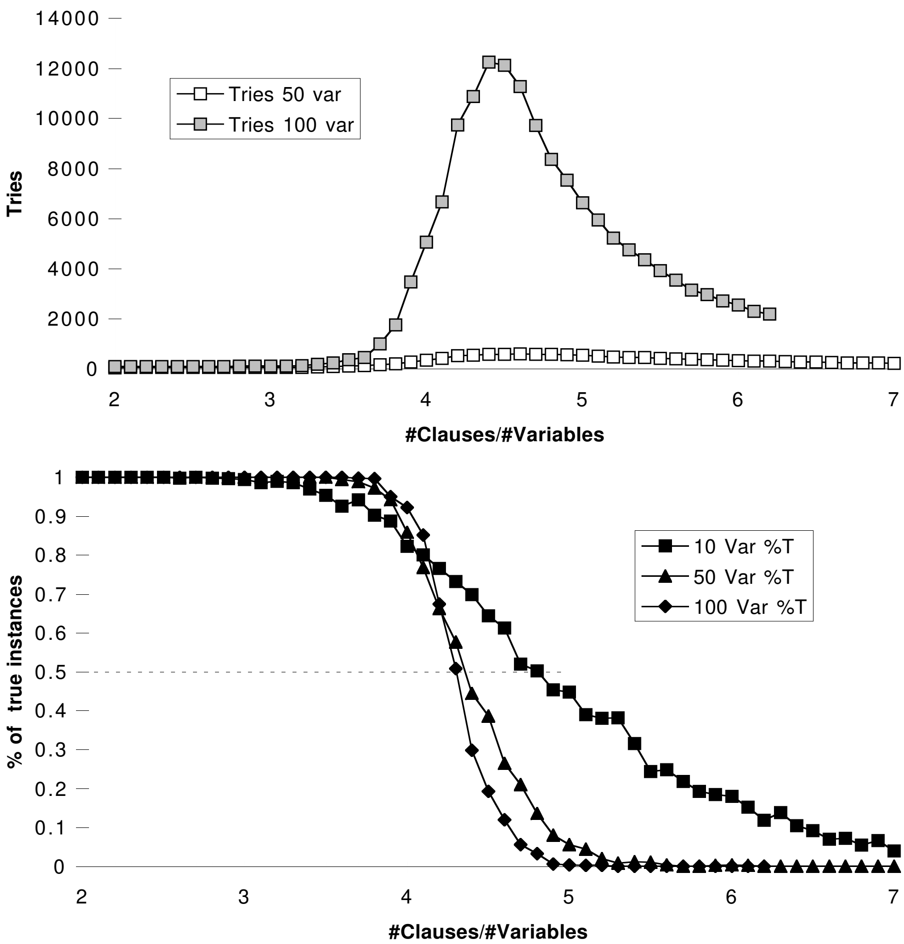


Figura 6.1: Risultati ottenuti con un algoritmo simile a DPLL su test random 3CNF, 500 formule/punto. Sopra: dimensioni medie dello spazio di ricerca. Sotto: percentuali di formule soddisfacibili. (*Per gentile concessione di M. Cadoli, A. Giovanardi e M. Schaerf*).

[Mitchell *et al.*, 1992; Crawford e Auton, 1996]. Inoltre, il valore medio e mediano della difficoltà delle formule (sia tempo di CPU utilizzato per risolverle che grandezza dello spazio di ricerca utilizzato) rivela un picco "easy-hard-easy" sempre centrato nella zona di transizione "100% soddisfacibile-100% insoddisfacibile. Aumentando N, le zone di transizione rimangono le stesse, solo le curve diventano più ripide (idealmente, per N tendente all'infinito, la curva di soddisfacibilità tende ad un gradino, mentre la curva di difficoltà tende ad un impulso, gradino ed impulso localizzati in $L/N \approx 4.3$). Questo fenomeno, rivelato per vari problemi NP-hard [Cheeseman *et al.*, 1991; Gent e Walsh, 1996] e noto come "transizione di fase" per certe analogie con fenomeni termodinamici, è stato lungamente studiato sia empiricamente [Mitchell *et al.*, 1992; Kirkpatrick e Selman, 1994] che teoricamente [Franco e Paull, 1983; Williams e Hogg, 1994; Yugami, 1995]. [2] Come conseguenza pratica, opportune scelte di L e N rendono possibile la creazione di formule difficili a piacere con circa 50% di probabilità di soddisfacibilità (cioè massima incertezza).

6.2 Il nuovo metodo "3CNF$_{\Box m}$"

Per estendere il metodo fixed-clause-length alle logiche modali normali $\mathcal{L} \in \mathcal{N}$ abbiamo bisogno di una nuova nozione di formule CNF per tali logiche (CNF$_{\Box m}$ d'ora in poi):

- una formula CNF$_{\Box m}$ è una congiunzione di clausole CNF$_{\Box m}$;
- una clausola CNF$_{\Box m}$ è una disgiunzione di letterali CNF$_{\Box m}$, cioè atomi CNF$_{\Box m}$ e loro negazioni;

[2] L'occorrenza di transizioni di soddisfacibilità e curve "easy-hard-easy" per la difficoltà delle formule nei problemi SAT venne osservata in [Mitchell *et al.*, 1992] su un test basato sul modello fixed-clause-length. Gli autori notarono che: (i) i picchi corrispondono con i punti "50%" nei grafici di soddisfacibilità; (ii) questi valori sono sempre localizzati intorno ad un valore fisso $L \approx 4.3N$, indipendentemente dal valore di N; (iii) entrambe le curve diventano più ripide aumentando N.

```
function rand_wff(d,m,L,N,p)
    for i := 1 to L do
        C_i := rand_clause(d,m,N,p);
    return ⋀_{i=1}^{L} C_i;

function rand_clause(d,m,N,p)
    for j := 1 to 3 do
        l_j := rand_lit(d,m,N,p);
    return ⋁_{j=1}^{3} l_j;

function rand_lit(d,m,N,p)
    φ := rand_atom(d,m,N,p);
    if flip_coin(0.5)
    then
        return φ;
    else
        return ¬φ;

function rand_atom(d,m,N,p)
    if (d=0 or flip_coin(p))
    then
        return rand_propositional_atom(N);
    else
        □_r := rand_box(m);
        C := rand_clause(d-1,m,N,p);
        return □_r C ;
```

Figura 6.2: L'algoritmo del generatore random di formule 3CNF$_{\Box m}$

- un atomo CNF$_{\Box m}$ è o un atomo proposizionale oppure è nella forma $\Box_r C$, dove $\Box_r \in \{\Box_1, \ldots, \Box_m\}$ e C è una clausola CNF$_{\Box m}$.

Si noti che gli operatori di congiunzione $\wedge$ compaiono solo al top-level di una formula CNF$_{\Box m}$. Senza perdere in generalità, possiamo fissare il numero di letterali per clausola. Una formula CNF$_{\Box m}$ è detta 3CNF$_{\Box m}$e tale numero è fissato a 3. È possibile generare formule 3CNF$_{\Box m}$ casuali in funzione di soli 5 parametri:

(i) la profondità d;

(ii) il numero m di "box" $\Box_r$ distinte;

(iii) il numero L di clausole al top-level;

(iv) il numero N di variabili proposizionali;

(v) la probabilità p con cui ogni atomo è proposizionale.

L'algoritmo del generatore random è descritto in Figura 6.2. Una formula 3CNF$_{\Box m}$ di profondità d viene prodotta generando in modo casuale L clausole 3CNF$_{\Box m}$ di profondità d. Una clausola 3CNF$_{\Box m}$ di profondità d viene prodotta generando in modo casuale 3 atomi 3CNF$_{\Box m}$ di profondità d, ciascuno negato con probabilità 0.5 (la funzione *flip_coin(p)* restituisce $True$ con probabilità p, $False$ altrimenti). Un atomo 3CNF$_{\Box m}$ di profondità d viene prodotto nel modo seguente. Se $d = 0$ oppure *flip_coin(p)* dà $True$, allora la funzione *rand_propositional_atom(N)* preleva casualmente un atomo $A_k \in \{A_1, \ldots, A_N\}$ (intuitivamente, il parametro p stabilisce la percentuale di atomi proposizionali ad ogni livello di profondità della formula). Altrimenti viene restituita una formula costituita da un operatore modale $\Box_r$, preso a caso tra $\{\Box_1, \ldots, \Box_m\}$ da *rand_box(m)*, seguito da una clausola di profondità $d-1$, generata ricorsivamente da $rand\_clause$.

Il metodo "3CNF$_{\Box m}$" viene quindi definito in analogia con la versione proposizionale. Per fissati valori di N, d, m e p, per valori crescenti di L, un certo numero (100, 500, 1000...) di formule casuali 3CNF$_{\Box m}$ vengono generate, ordinate internamente (si veda Sezione 5.2) e date in ingresso alle procedure sotto test. Le percentuali di soddisfacibilità, i tempi di CPU medi/mediani le dimensioni medie/mediane dello spazio di ricerca vengono rappresentate in funzione del rapporto L/N.

La metodologia proposta preserva le tre prerogative fondamentali della versione proposizionale.

1. Data $\mathcal{L} \in \mathcal{N}$, le formule CNF$_{\Box m}$ rappresentano tutte le formule di $\mathcal{L}$, poiché esiste la possibilità di convertire tutte le formule

function *$CNF_{\Box m}$-convert(φ)*
  foreach $\varphi_i \in Atoms^0_{\Box}(\varphi)$ **do**
    $\varphi_i :=$ $\mathrm{CNF}_{\Box m}$-convert(φ_i);
  foreach $\varphi_i \in Atoms^0_{\Box}(\varphi)$ **do**
    if $(\varphi_i = \Box_r \bigwedge_j \varphi_{ij})$
    then
      $\varphi_i := \bigwedge_j \Box_r \varphi_{ij}$;
  return *propositional_CNF-convert(φ)*;

Figura 6.3: Un semplice algoritmo di conversione in forma $\mathrm{CNF}_{\Box m}$.

in forma $\mathrm{CNF}_{\Box m}$ [$3\mathrm{CNF}_{\Box m}$] preservando la $\mathcal{L}$-soddisfacibilità. Un semplice algoritmo per effettuare tale conversione è illustrato in Figura 6.3. La conversione viene effettuata ricorsivamente sul livello modale delle sottoformule, dalle foglie alla radice della formula φ. Ad ogni livello vengono applicate sequenzialmente la trasformazione $\Box_r \bigwedge_j \bigvee_i \varphi_{ij} \Longrightarrow \bigwedge_j \Box_r \bigvee_i \varphi_{ij}$ [3] e la conversione CNF proposizionale standard (considerando gli atomi modali come variabili proposizionali).

2. L'uso della forma $3\mathrm{CNF}_{\Box m}$ minimizza il numero di parametri da gestire. Infatti, utilizziamo un unico parametro L di "and-branching" e nessuno di "or-branching", indipendentemente dalla profondità modale d. Questo è stato ottenuto "portando verso l'alto" tutte le congiunzioni e fissando la larghezza delle clausole.

3. Come nel caso proposizionale, L [N] incrementa [decrementa] *monotonicamente* il livello di *constraintness* delle formule generate, permettendo quindi una regolazione approssimativa della probabilità di soddisfacibilità e della difficoltà delle formule casuali generate. Anche in questo caso, il grafico delle

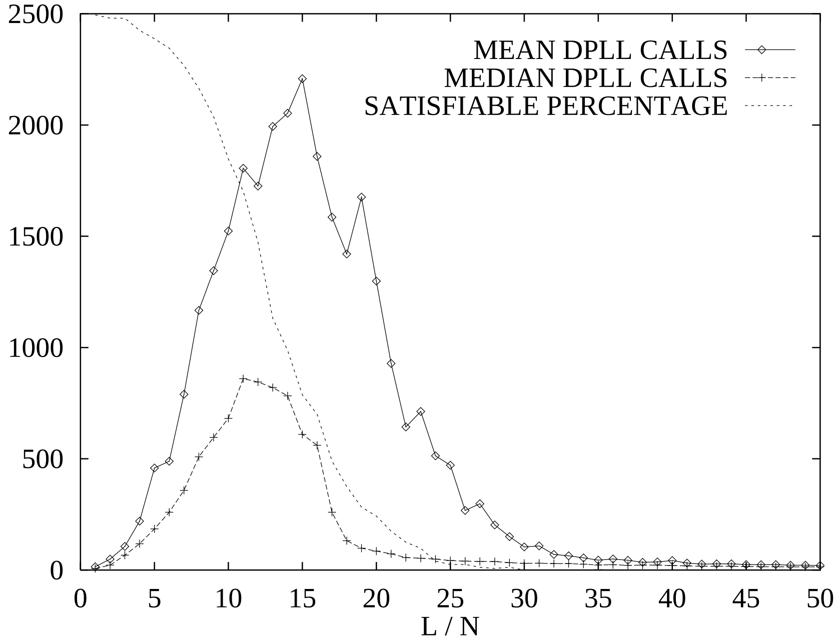
[3]Se $\mathcal{L} \in \mathcal{N}$, allora vale $\models_{\mathcal{L}} \Box_r \bigwedge_i \varphi_i \equiv \bigwedge_i \Box_r \varphi_i$, per ogni $\Box_r$ e φ_i.

percentuali di formule soddisfacibili rivela una transizione da 100% soddifacibili a 100% insoddisfacibili, e il valore medio e mediano della difficoltà delle formule rivela un picco "easy-hard-easy" sempre centrato nella zona di transizione "100% soddisfacibile-100% insoddisfacibile. Opportune scelte di L e N rendono possibile la creazione di formule difficili a piacere con circa 50% di probabilità di soddisfacibilità.

6.3 Un esempio

Per illustrare le caratteristiche delle formule generate da una sessione di test $3\text{CNF}_{\Box m}$ in Figura 6.4 presentiamo i grafici risultanti dall'esecuzione di KSAT su un campione di 40.000 formule random $3\text{CNF}_{\Box m}$ con $p = 0.5$, $d = 1$, $m = 1$, $N = 4$, $L/N = 1 \ldots 40$, 1000 formule/punto. Nel grafico superiore rappresentiamo i valori medi e mediani del numero di chiamate ricorsive a KSAT_W, cioè le dimensioni dello spazio di ricerca. Nel grafico inferiore rappresentiamo i valori medi e mediani dei tempi di CPU impiegati da KSAT a risolvere le formule. Sullo sfondo di entrambe le figure, rappresentiamo la percentuale di formule soddisfacibili (100% soddisfacibili a fondo scala).

Il primo grafico rivela l'esistenza di un picco "easy-hard-easy" nella dimensione dello spazio di ricerca, cioè della difficoltà delle formule. Confrontando con la curva di soddisfacibilità sullo sfondo, si nota che, all'aumentare di L/N la difficoltà delle formule inizialmente cresce nella zona prevalentemente soddisfacibile, raggiunge un massimo in prossimità del punto "50% soddisfacibili" (cioè massima indeterminazione delle formule) e poi *decresce* man mano che la percentuale di formule insoddisfacibili aumenta. Questo avviene sia per i valori medi sia per i valori mediani. Si noti che la difficoltà delle formule decresce con la dimensione delle formule. Tale comportamento è analogo a quello delle formule random 3SAT. Il secondo grafico mostra come le curve relative ai tempi di CPU risultino dalla combinazione delle curve "easy-hard-easy" e una componente lineare. Infatti il tempo di CPU è dato dal prodotto tra il numero di



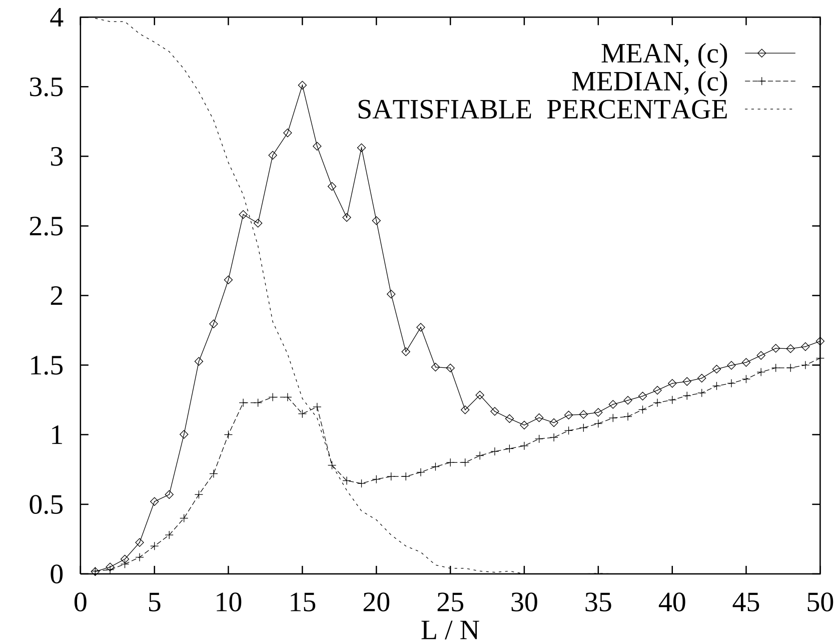


Figura 6.4: $p = 0.5$, $d = 1$, $m = 1$, $N = 4$, 1000 formule/punto. Sopra: dimensioni dello spazio di ricerca di KSAT, medie e mediane. Sotto: tempi di CPU di KSAT, medi e mediani. Sfondo sopra e sotto: percentuali di formule soddisfacibili (100% soddisfacibili a fondo scala).

chiamate alla funzione *assign* di Figura 5.1 (cioè la dimensione dello spazio di ricerca, curva easy-hard-easy) per il tempo di esecuzione della funzione *assign* stessa, che cresce linearmente con la dimensione della formule. Osserveremo lo stesso comportamento in tutti i test presentati nel Capitolo 7.

6.4 Estensione ad altre logiche

Fino a questo momento, abbiamo ristretto la nostra attenzione alle logiche $\mathcal{L} \in \mathcal{N}$ e, data la corrispondenza tra K(m) e $\mathcal{ALC}$, anche a $\mathcal{ALC}$. Nondimeno, il metodo 3CNF$_{\Box m}$ è estensibile ad un gran numero di logiche.

Mostriamo ora come estendere il nostro metodo a tutte le logiche che estendano la sintassi di K(m)/$\mathcal{ALC}$ con altri operatori prefissi $\mathcal{C}_i$ (ad esempio inversa, composizione ...) o con altri tipi di atomi $\mathcal{A}_i$ (ad esempio, restrizione numerica ...). Per ciascun nuovo operatore $\mathcal{C}_i$ [tipologia di atomo $\mathcal{A}_i$] deve esistere una funzione *rand_*$\mathcal{C}_i$*(*$v_1^{\mathcal{C}_i}, v_2^{\mathcal{C}_i}, \ldots$*)* [*rand_*$\mathcal{A}_i$*(*$v_1^{\mathcal{A}_i}, v_2^{\mathcal{A}_i}, \ldots$*)*], che usiamo per generare $\mathcal{C}_i$ [$\mathcal{A}_i$] in funzione di parametri specifici $v_1^{\mathcal{C}_i}, v_2^{\mathcal{C}_i}, \ldots$ [$v_1^{\mathcal{A}_i}, v_2^{\mathcal{A}_i}, \ldots$]. Per esempio, se $\mathcal{C}_i = \Box_r$, allora *rand_*$\mathcal{C}_i$*(*$v_1^{\mathcal{C}_i}, \ldots$*)* è *rand_box(m)*. Come mostrato in Figura 6.5, il generatore random di atomi viene esteso sostituendo la chiamata *rand_box(m)* con una sequenza di istruzioni più complessa che alternativamente

(i) seleziona, con probabilità $P_{\mathcal{C}_i}$, un operatore $\mathcal{C}_i$, e restituisce *rand_*$\mathcal{C}_i$*(*$v_1^{\mathcal{C}_i}, v_2^{\mathcal{C}_i}, \ldots$*)* concatenato a una clausola random di profondità $d-1$ generata da *rand_clause(*$d-1, \ldots$*)*, oppure

(ii) seleziona, con probabilità $P_{\mathcal{A}_i}$, un tipo di atomo $\mathcal{A}_i$, e restituisce *rand_*$\mathcal{A}_i$*(*$v_1^{\mathcal{A}_i}, v_2^{\mathcal{A}_i}, \ldots$*)*.

Ovviamente $\sum_i P_{\mathcal{C}_i} + \sum_i P_{\mathcal{A}_i} = 1$. I parametri $P_{\mathcal{C}_i}$, $v_1^{\mathcal{C}_i}, v_2^{\mathcal{C}_i}, \ldots P_{\mathcal{A}_i}$, $v_1^{\mathcal{A}_i}, v_2^{\mathcal{A}_i}, \ldots$ vengono aggiunti alla lista dei parametri del generatore e di tutte le funzioni intermedie. Per effettuare un test, i valori di tutti i parametri eccetto L e $v \in \{d, p, N, P_{\mathcal{C}_1},\ v_1^{\mathcal{C}_1}, v_2^{\mathcal{C}_1}, \ldots P_{\mathcal{A}_1}$, $v_1^{\mathcal{A}_1}, v_2^{\mathcal{A}_1}, \ldots\}$ vengono fissati. L'esperimento può venir eseguito in

function *rand_atom(d,m,N,p,* $P_{\mathcal{C}_1}, v_1^{\mathcal{C}_1}, v_2^{\mathcal{C}_1}, \ldots P_{\mathcal{A}_1}, v_1^{\mathcal{A}_1}, v_2^{\mathcal{A}_1}, \ldots$*)*
    if (*d=0* **or** *flip_coin(p)*)
    then
        return *rand_propositional_atom(N)*;
    else
        case *Rand-value([0,1])* **in**
            $[0, P_{\mathcal{C}_1})$:
                $\mathcal{C}$:= *rand_*$\mathcal{C}_1$*(*$v_1^{\mathcal{C}_1}, v_2^{\mathcal{C}_1}, \ldots$*)*;
                C := *rand_clause(d–1,m,N,p,*$P_{\mathcal{C}_1}, \ldots$*);*
                return $\mathcal{C}$ C;
            $[P_{\mathcal{C}_1}, P_{\mathcal{C}_1} + P_{\mathcal{C}_2})$:
                $\mathcal{C}$:= *rand_*$\mathcal{C}_2$*(*$v_1^{\mathcal{C}_2}, v_2^{\mathcal{C}_2}, \ldots$*)*;
                C := *rand_clause(d–1,m,N,p,*$P_{\mathcal{C}_1}, \ldots$*);*
                return $\mathcal{C}$ C;
            ...
            $[\sum_{k=1}^{i-1} P_{\mathcal{C}_k}, \sum_{k=1}^{i} P_{\mathcal{C}_k})$:
                $\mathcal{C}$:= *rand_*$\mathcal{C}_i$*(*$v_1^{\mathcal{C}_i}, v_2^{\mathcal{C}_i}, \ldots$*)*;
                C := *rand_clause(d–1,m,N,p,*$P_{\mathcal{C}_1}, \ldots$*);*
                return $\mathcal{C}$ C;
            ...
            $[\sum_i P_{\mathcal{C}_i}, \sum_i P_{\mathcal{C}_i} + P_{\mathcal{A}_1})$:
                $\mathcal{A}$:= *rand_*$\mathcal{A}_1$*(*$v_1^{\mathcal{A}_1}, v_2^{\mathcal{A}_1}, \ldots$*)*;
                return $\mathcal{A}$;
            $[\sum_i P_{\mathcal{C}_i} + P_{\mathcal{A}_1}, \sum_i P_{\mathcal{C}_i} + P_{\mathcal{A}_1} + P_{\mathcal{A}_2}))$:
                $\mathcal{A}$:= *rand_*$\mathcal{A}_2$*(*$v_2^{\mathcal{A}_2}, v_2^{\mathcal{A}_2}, \ldots$*)*;
                return $\mathcal{A}$;
            ...
        ;
;

Figura 6.5: Schema del generatore random “generalizzato” di atomi.

modo simile a quello descritti nelle sezioni precedenti (dove $\mathcal{C}_1 = \Box_r$, $v \in \{N, m, d\}$ e $P_{\mathcal{C}_1} = 1$).

Anche se non abbiamo ancora evidenze sperimentali a riguardo, abbiamo motivo di ritenere che il metodo descritto conservi le caratteristiche evidenziate nelle sezioni precedenti, almeno per tutte le logiche in cui vale la proprietà

$$\mathcal{C}_i(\varphi_1 \wedge \varphi_2) \equiv (\mathcal{C}_i\varphi_1 \wedge \mathcal{C}_i\varphi_2), \tag{6.1}$$

per ogni operatore prefisso $\mathcal{C}_i$. Questo riguarda l'intera gerarchia delle logiche modali normali [Chellas, 1980; Halpern e Moses, 1992], e molte altre logiche modali e terminologiche. Le seguenti osservazioni giustificano le nostre affermazioni di cui sopra.

1. Se vale la (6.1), le formule generate rappresentano tutte le formule della logica considerata, in quanto esiste un metodo di conversione simile a quello di Figura 6.3 che preservi la soddisfacibilità.

2. Il numero di parametri da gestire viene minimizzato in quanto, come per $3\mathrm{CNF}_{\Box m}$, utilizziamo un unico parametro L di "and-branching" e nessuno di "or-branching".

3. Come nel caso precedente L [N] incrementa [decrementa] monotonicamente il livello di constraintness delle formule generate. Questo dovrebbe quindi permettere quindi una regolazione approssimativa della probabilità di soddisfacibilità e della difficoltà delle formule casuali generate.

Come in precedenza, quest'ultimo punto merita un ulteriore approfondimento. In primo luogo, indipendentemente dalla logica considerata, il parametro L [N] incrementa [decrementa] monotonicamente la constraintness delle formule generate, causando quindi una transizione nei grafici delle percentuali di formule soddisfacibili. Logiche diverse possono solo influenzare la posizione e la ripidità delle curve stesse. Per esempio, se consideriamo la gerarchia delle logiche modali normali [Chellas, 1980; Halpern e Moses, 1992], a parità dei valori dei parametri considerati, più alta è la posizione gerarchica della logica considerata, più vincolate risulteranno le formule, e quindi più la transizione e i picchi si sposteranno verso sinistra In secondo luogo, le considerazioni presentate nel Capitolo 8 (ma si veda anche [Williams e Hogg, 1994]) suggeriscono come i picchi "easy-hard-easy" centrati nella zona di transizione della soddisfacibilità siano una conseguenza diretta dell'aumento della constraintness, indipendentemente dalla logica considerata. Ci aspettiamo quindi che tale fenomeno occorra nella maggior parte le logiche modali normali e terminologiche.

Parte IV

Testing

Capitolo 7

Procedure DPLL-based contro tableau-based

In questo capitolo presentiamo un confronto empirico tra l'approccio DPLL-based di KSAT e quello tableau-based di varie implementazioni allo stato dell'arte (quando non detto esplicitamente, per KSAT intendiamo Versione 4). La trattazione è organizzata come segue. Nella Sezione 7.1 confrontiamo le prestazioni delle varie versioni di KSAT (Capitolo 5) con TABLEAU, (Sezione 3.3) cioè due implementazioni "dirette" rispettivamente di KSAT e dell'algoritmo tableau-based descritto in [Hollunder *et al.*, 1990]. Questo ci permette di confrontare le prestazioni degli *algoritmi*, senza introdurre "distorsioni" nella valutazione dovute a sofisticate tecniche di implementazione. Nella Sezione 7.2 confrontiamo le procedure TABLEAU, KRIS, (Sezione 3.3) KSAT e KSAT_s, una versione lievemente migliorata di KSAT (Sezione 7.2). Questo ci permette di analizzare gli effetti della qualità dell'implementazione sull'efficienza delle procedure. Nella Sezione 7.3 confrontiamo KRIS e KSAT_s su un insieme molto grande di formule. Questo ci permette di portare a termine un confronto esaustivo tra le più efficienti procedure tableau-based e DPLL-based a nostra disposizione.

7.1 Confronto tra algoritmi

Abbiamo implementato KSAT in Common LISP sulla base di un'implementazione di DPLL per formule non-CNF sviluppata precedentemente [Armando e Giunchiglia, 1993]. Rispetto agli algoritmi descritti nel Capitolo 5, abbiamo fatto le seguente scelte di implementazione: la funzione *assign* (Figure 5.1 e 5.2) effettua un'(inefficiente) "lazy-evaluation" della formula φ, richiedendo tempo $O(|\varphi|)$ (per efficienti implementazioni di *assign* si vedano [Zhang e Stickel, 1994; Crawford e Auton, 1993; Buro e Buning, 1992]); la funzione *choose-literal(φ)* (Figure 5.1 e 5.2) utilizza una semplice euristica: "scegli la variabile che occorre più frequentemente in φ; nell'implementazione della versione 4, la funzione *Likely-Unsatisfiable* è implementata banalmente per restituire sempre "*True*". Inoltre, per garantire la massima correttezza del confronto di prestazioni, in tutti i test abbiamo disabilitato la funzione di ordinamento di KSAT e generato le formule già ordinate. Si è voluto così evitare che solo KSAT usufruisse del vantaggio dell'ordinamento tra atomi.

TABLEAU e KSAT sono stati compilati in `AKCL 1.600` ed eseguiti sotto `SunOS 4.1.3`, e confrontati su sei gruppi di 4000 formule random $3\text{CNF}_{\Box m}$ ciascuno, indicati con numeri romani da (i) a (vi), con $m = 1$, $p = 0.5$, $d = 1$, $N = 2, 3, 4$ (gruppi (i), (ii) e (iii)) e $m = 1$, $p = 0.5$, $d = 2$, $N = 1, 2, 3$ (gruppi (iv), (v) e (vi)). I risultati sono riportati rispettivamente in Figura 7.1 (percentuali di formule soddisfacibili) e nelle Figure 7.2 e 7.3 (tempi di CPU medi). Ogni gruppo è ripartito in 40 sottogruppi di 100 formule, ciascuno corrispondente ad un valore intero di L/N compreso tra 1 e 40 (un singolo punto nei grafici). Tale range è stato scelto per coprire l'intera transizione tra 100% soddisfacibile a 100% insoddisfacibile. La necessità di svolgere i test TABLEAU in tempi ragionevoli (per dare un'idea, la valutazione del singolo punto più alto della curva TABLEAU (v) di Figura 7.3 ha richiesto 137 ore di tempo di CPU su una workstation SUN SPARC10 32M!) ha imposto alcuni vincoli sulla metodologia di test. In primo luogo, abbiamo dovuto eseguire i test su sei differenti macchine:

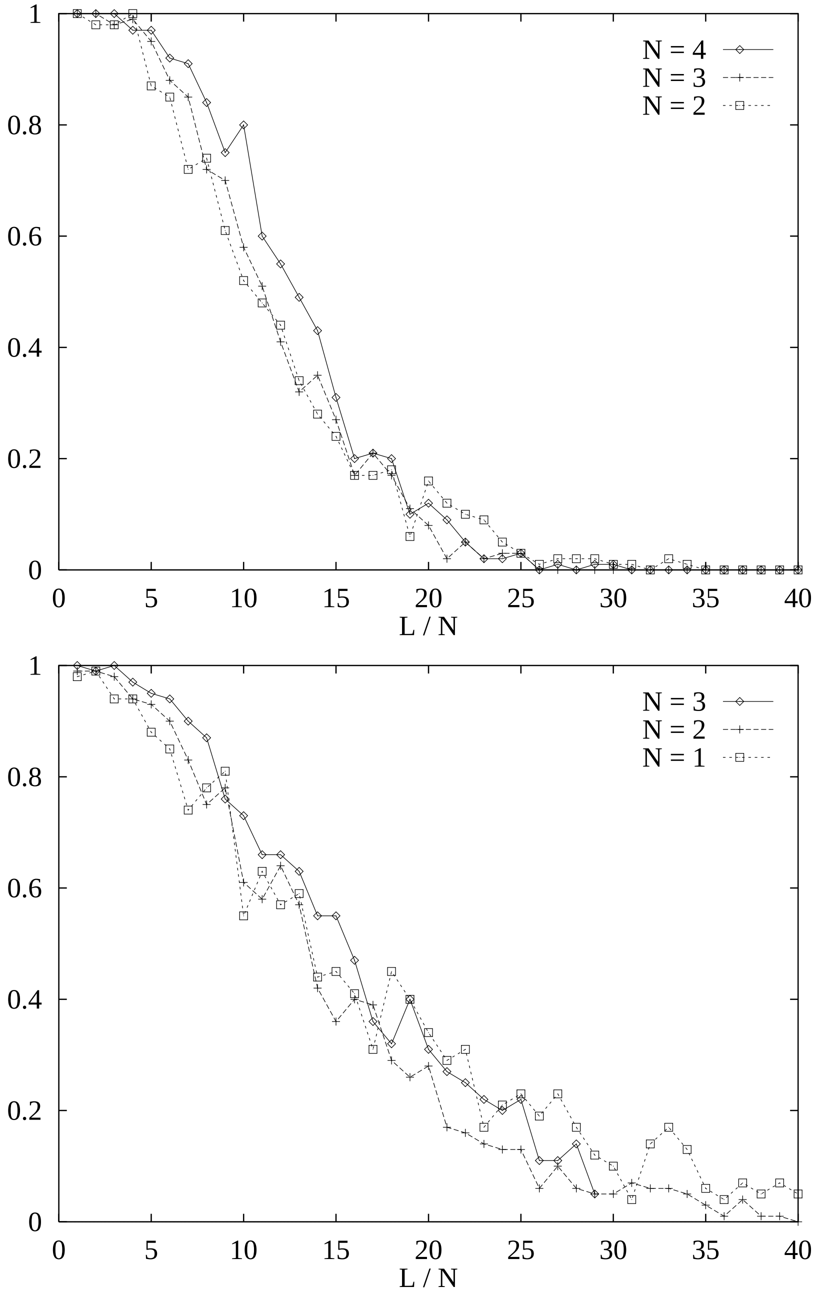


Figura 7.1: Percentuali di formule soddisfacibili. 100 formule/punto. Sopra: $p = 0.5$, $d = 1$, $N = 2, 3, 4$, $L/N = 1 \dots 40$. Sotto: $p = 0.5$, $d = 2$, $N = 1, 2, 3$, $L/N = 1 \dots 40$.

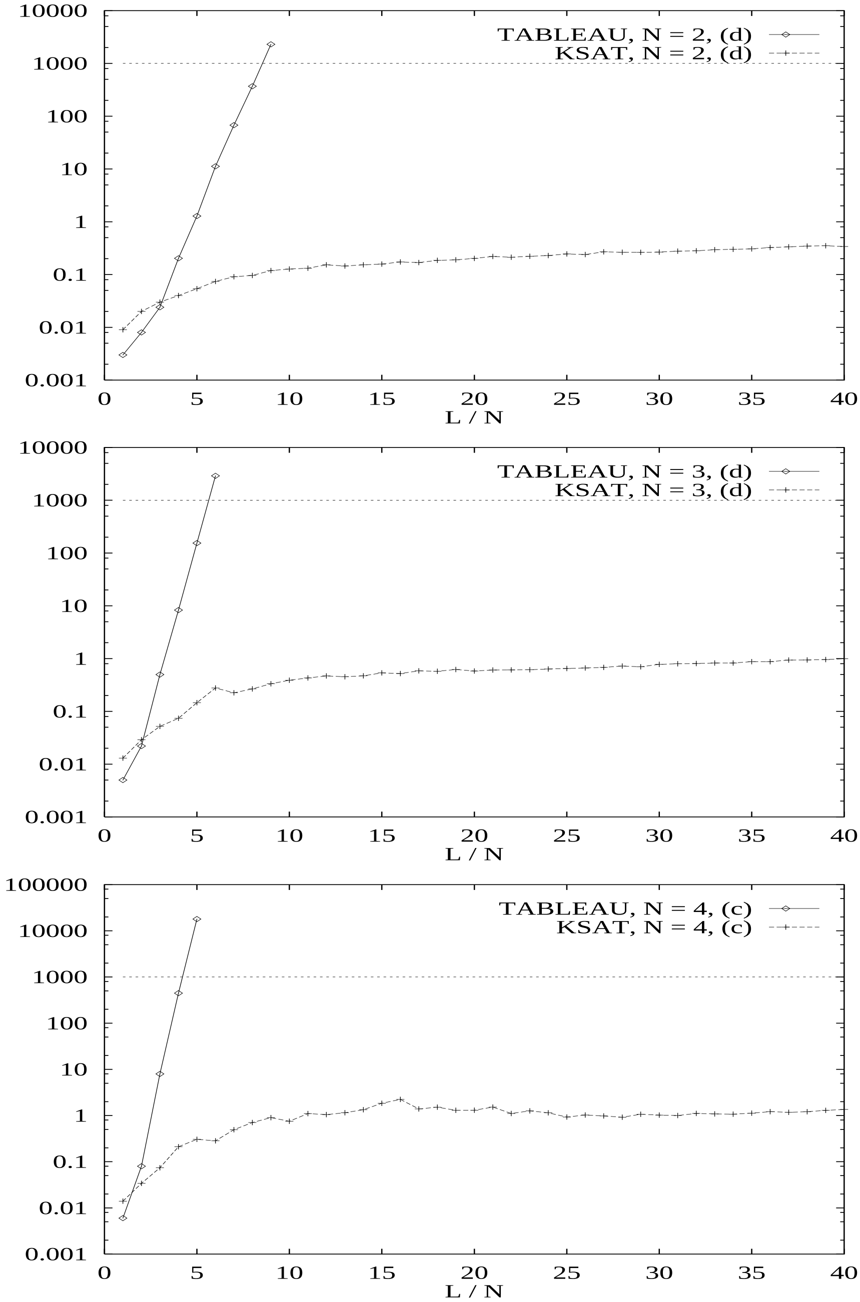


Figura 7.2: Tableau contro Ksat: tempi medi di CPU (secondi). 100 formule/punto. Gruppi (i), (ii) e (iii): $p = 0.5$, $d = 1$, $N = 2, 3, 4$, $L/N = 1 \ldots 40$. Le etichette (a)...(d) indicano il tipo di workstation usata per il test.

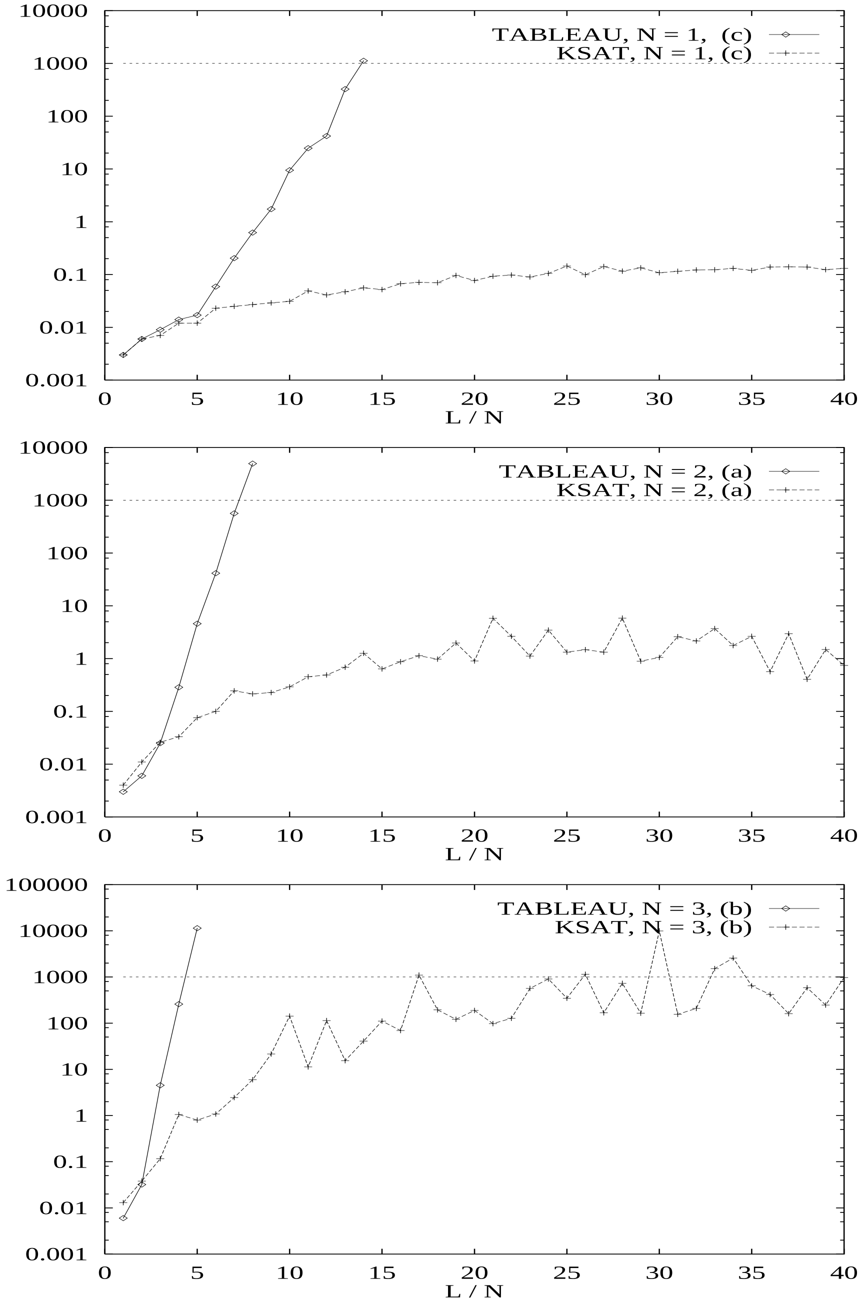


Figura 7.3: Tableau contro Ksat: tempi medi di CPU (secondi). 100 formule/punto. Gruppi (iv), (v) e (vi): $p = 0.5$, $d = 2$, $N = 1, 2, 3$, $L/N = 1 \ldots 40$. Le etichette (a)...(d) indicano il tipo di workstation usata per il test.

(a) una `SUN SPARC10 SUPERSPARC 32M`,

(b) due identiche `SUN SPARC2 SUN4/75 32M`,

(c) due identiche `SUN SPARC ELC SUN4/25 16M`,

(d) una `SUN SPARC SLC SUN4/20 16M`.

Per ogni gruppo, TABLEAU e KSAT sono sempre state eseguite o sulla stessa workstation o su workstation dalla stessa configurazione. In secondo luogo, ci siamo limitati a 100 formule/punto (per un totale di 4000 random $3\mathrm{CNF}_{\Box m}$ formule per gruppo). Infine, i tutti i test abbiamo fermato l'esecuzione di TABLEAU ogniqualvolta il valore medio ha superato i 1000 secondi.

Come si vede nelle Figure 7.2 e 7.3, KSAT supera drasticamente in prestazioni TABLEAU in tutti i problemi considerati (si noti la scala logaritmica sugli assi verticali!). I tempi di CPU medi di TABLEAU crescono esponenzialmente con il numero di clausole, mentre le corrispondenti curve di KSAT si "appiattiscono", presentando un'andatura apparentemente polinomiale (vi veda Capitolo 9). Le curve di TABLEAU raggiungono il limite superiore di 1000 secondi dopo pochi passi, circa 10^3-10^4 sopra i corrispondenti valori di KSAT. L'estrapolazione delle curve TABLEAU suggerisce che tale differenza in prestazioni possa raggiungere vari ordini di grandezza per valori di L/N prossimi a 40 (si veda anche Sezione 7.2).

Per avere una valutazione comparativa complessiva, abbiamo confrontato le versioni 2,3 e 4 di KSAT e TABLEAU sul gruppo di problemi (v) (non ha senso considerare la Versione 1 in quanto le formule sotto test sono già ordinate). Abbiamo anche analizzato il comportamento di MODIFIED TABLEAU, cioè TABLEAU modificato con la regola *lemma generation* [D'Agostino, 1992]. Per essere più precisi, MODIFIED TABLEAU è TABLEAU con la regola "$\vee$-elimination" $\frac{\varphi\vee\psi}{\varphi \quad \psi}$ sostituita con le nuove regole $\frac{\varphi\vee\psi}{\varphi \quad \psi,\neg\varphi}$, $\frac{\varphi\vee\psi}{\varphi,\neg\psi \quad \psi}$ (il motivo di questa scelta diverrà chiaro più avanti). I risultati sono riportati in Figura 7.4.[1]

[1]Il punto più in alto della curva relativa a MODIFIED TABLEAU rappresenta solo 21 formule, in quanto `AKCL` ha abortito alla 22sima formula.

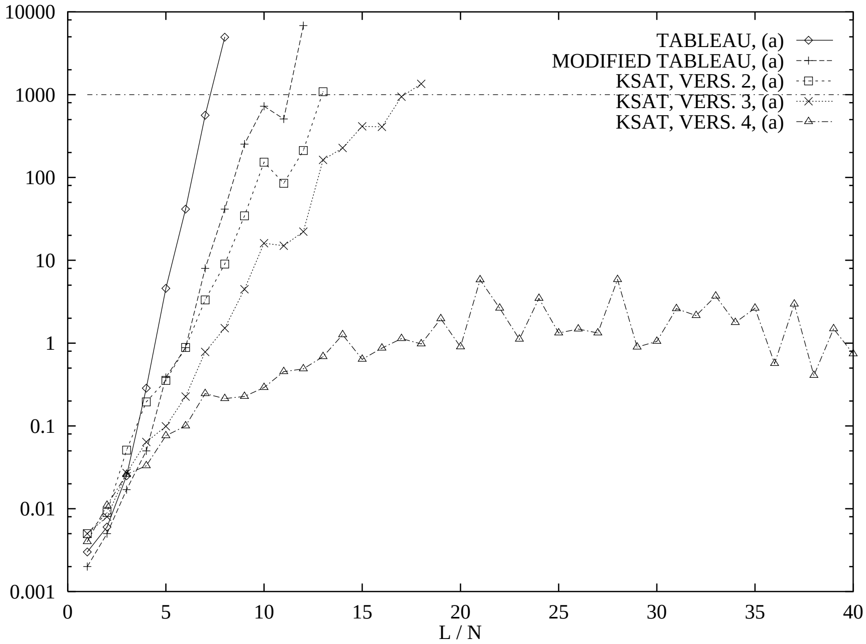


Figura 7.4: TABLEAU, MODIFIED TABLEAU e Versioni 2,3,4 di KSAT: tempi di CPU medi (secondi). $d = 2$. $p = 0.5$. $N = 2$, $L/N = 1 \ldots 40$. 100 formule/punto. L'etichetta (a) indica il tipo di workstation usata per il test.

Come si può facilmente notare, già la Versione 2 si comporta nettamente meglio sia di TABLEAU che di MODIFIED TABLEAU. TABLEAU supera il limite superiore dopo 8 passi, circa 10^2-10^3 sopra il corrispondente valore della Versione 2. MODIFIED TABLEAU migliora decisamente TABLEAU (un fattore 10^2 all'ottavo passo). Ciononostante, il miglioramento introdotto non compensa il divario di prestazioni con la Versione 2. L'aggiunta della fattorizzazione delle componenti $\bigwedge_i \alpha_i$ (Versione 3) introduce una ulteriore miglioramento (circa un ordine di grandezza in torno al 13^o passo). Infine, si ottiene un grosso miglioramento aggiungendo il test di soddisfacibilità sugli assegnamenti incompleti (Versione 4). Come in precedenza, l'estrapolazione delle curve suggerisce che il divario nelle prestazioni può raggiungere vari ordini di grandezza.

7.2 Confronto tra algoritmi e sistemi

Ci sono molti trucchi che possono rendere più efficiente l'implementazione di una procedura quali, ad esempio, l'uso di sofisticate strutture dati, varie forme di preprocessing, la gestione "distruttiva" dei dati rappresentanti le formule [Zhang e Stickel, 1994; Buro e Buning, 1992]. Una nuova implementazione, KSAT_s (dove "S" significa "System") è stata ottenuta da KSAT aggiungendo una fase iniziale di preprocessing delle formule ed altri miglioramenti relativamente minori. [2] Tutte le procedure sotto test, cioè TABLEAU, KRIS, KSAT e KSAT_s, sono state compilate ed eseguite in `Allegro CL 4.2` su una workstation `SUN SPARC10 32M`. Questo ci ha premesso si usare il meccanismo di timeout di `Allegro`.

Abbiamo messo a confronto TABLEAU, KRIS, KSAT e KSAT_s su un gruppo di formule simile al gruppo (vi) di Figura 7.3, cioè

[2] L'implementazione di KSAT_s è per molti aspetti ancora ancora "naive": è in LISP; non utilizza strutture dati sofisticate; è costruita su un'implementazione di DPLL "fatta in casa", ben lontana dalle prestazioni delle implementazioni allo stato dell'arte, come quelle descritte in [Buro e Buning, 1992; Zhang e Stickel, 1994; Crawford e Auton, 1993]. Comunque, i miglioramenti introdotti in KSAT_s ci permettono di avere idea degli effetti sull'efficienza della qualità dell'implementazione.

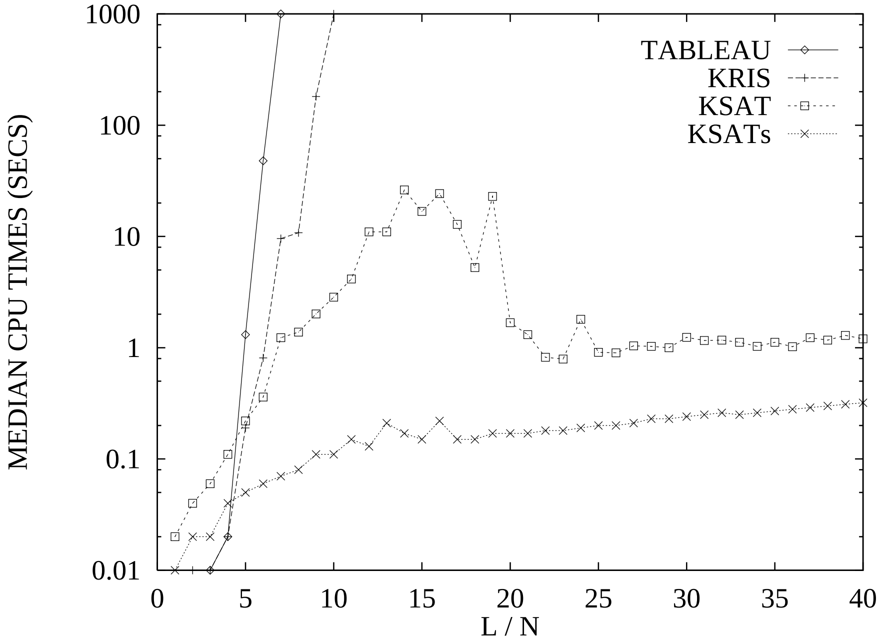


Figura 7.5: $d = 2$, $m = 1$, $N = 3$, $p = 0.5$, $L = N \ldots 40N$. TABLEAU, KRIS, KSAT e KSAT$_s$. Tempi mediani di CPU, 100 formule/punto.

4000 formule random $3\mathrm{CNF}_{\Box m}$ con $d = 2$, $m = 1$, $N = 3$, $p = 0.5$, $L/N \in \{1 \ldots 40\}$, 100 formule per punto. Per minimizzare i tempi e per rendere i risultati più significativi, abbiamo introdotto alcuni miglioramenti nella tecnica di test. I primo luogo, confrontiamo tempi *mediani* anziché tempi *medi*, in quanto i primi sono molto meno sensibili al rumore introdotto dagli "outliers" [3]. In secondo luogo, abbiamo introdotto un meccanismo di timeout di 1000 secondi su ogni formula: la procedura viene bloccata ogni volta che supera tale valore, e il valore viene posto convenzionalmente a 1000 secondi. [4]

I risultati sono presentati in Figura 7.5. Dall'analisi dei risultati emergono quattro osservazioni, che riportiamo in ordine crescente di importanza.

1. Migliorando la qualità dell'implementazione — cioè da Tableau a Kris o da Ksat a Ksat_s — si introducono rilevanti miglioramenti *quantitativi* delle prestazioni. Infatti, Kris raggiunge il limite superiore al 10^o passo, mentre Tableau lo raggiunge al 7^o, circa due ordini di grandezza peggio del corrispondente valore di Kris. Analogamente, Ksat ha un massimo al 14^o passo, più di 2 ordini di grandezza peggio del corrispondente valore di Ksat_s.

2. Migliorando la qualità dell'implementazione non si hanno effetti sul comportamento *qualitativo* delle procedure. Infatti, per quanto i grafici ci permettano di vedere, sia le curve di Tableau che quelle di Kris presentano una crescita esponenziale con il numero di clausole, mentre le curve di Ksat

[3] Come è ben noto nelle comunità SAT e CSP, i valori risultanti da test su problemi NP-hard non hanno andamento gaussiano, e in particolare presentano sempre singoli valori "outliers" estremamente alti, fino a vari ordini di grandezza superiori agli altri, che sfalsano completamente il calcolo dei valori medi. Per questo motivo si preferisce generalmente calcolare i valori mediani [Mitchell *et al.*, 1992; Williams e Hogg, 1994].

[4] Si noti che l'uso del timeout ha senso solo se vengono calcolati valori mediani. Un valore mediano di 1000 secondi significa "più del 50% delle formule non sono state risolte in 1000 secondi". Le percentuali di soddisfacibilità sono valutate solo sulle formule che vengono effettivamente risolte, quindi questi valori devono esser considerati puramente indicativi.

e KSAT$_s$ si appiattiscono quando il numero di clausole supera un certo valore.

3. Indipendentemente dalla qualità dell'implementazione, KSAT e KSAT$_s$ si comportano *quantitativamente* molto meglio di TABLEAU e KRIS. Per esempio, il divario di prestazioni tra KSAT$_s$ e KRIS al 10^o passo è di circa 4 ordini di grandezza. Inoltre, l'estrapolazione delle curve di KRIS suggeriscono come questo divario possa raggiungere vari ordini di grandezza per L/N prossimo a 40. Per verificare questa considerazione, abbiamo provato KRIS su cento formule con gli stessi valori e $L/N = 40$ (cioè i valori all'estrema destra di Figura 7.5). Nessuna formula è stata risolta all'interno del tempo di 1000 secondi concesso a ciascuna dal timeout. Disinserendo il meccanismo di timeout, KRIS non è stato in grado di risolvere la prima formula in un mese di esecuzione. Questo ci permette di dire che le curve relative alle procedure tableau-based *non decrescono* dopo un certo valore come le curve KSAT. Nella Sezione 8.2 presenteremo una spiegazione di questo fatto.

4. Indipendentemente dalla qualità dell'implementazione, KSAT e KSAT$_s$ si comportano *qualitativamente* molto meglio di TABLEAU e KRIS. Infatti, mentre le curve di TABLEAU e KRIS crescono esponenzialmente con il numero delle clausole, le curve di KSAT e KSAT$_s$ crescono polinomialmente. Più precisamente, tali curve sono una combinazione di un picco "easy-hard-easy" e una componente lineare (si veda Figura 6.4, ma anche le Figure 9.1, 9.2 e 9.2.

Per confermare ulteriormente il divario di prestazioni tra procedure DPLL-based e tableau-based, abbiamo effettuato un altro test, completamente differente, basato sulla classe di formule $\{\varphi_d^K\}_{d=1,2,\dots}$ presentata in [Halpern e Moses, 1992]. Si tratta di una classe di formule K(1)-soddisfacibili, parametriche nella profondità d, con $2d+1$ variabili proposizionali $A_1, \dots A_{d+1}, B_1, \dots B_d$. Tali formule sono

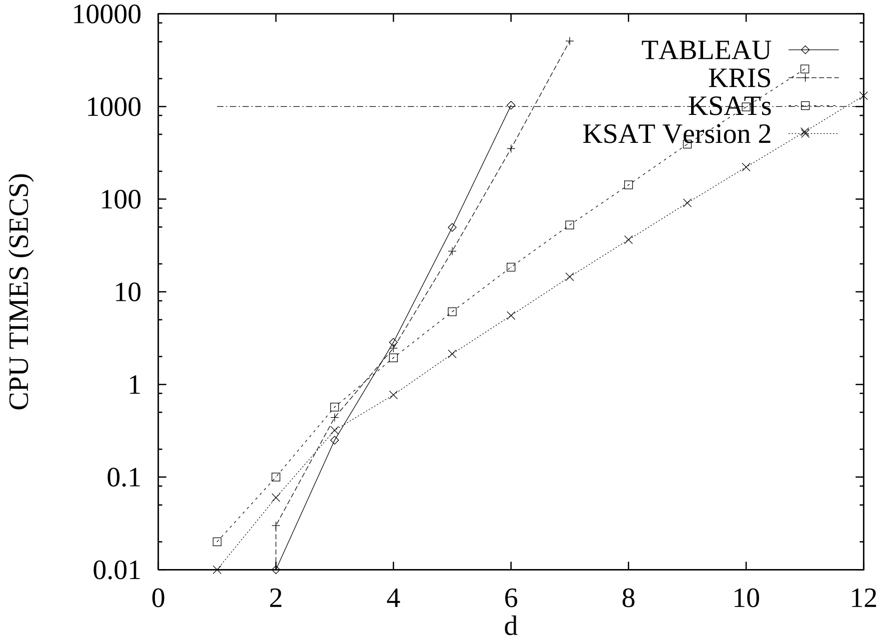


Figura 7.6: TABLEAU, KRIS, KSAT$_s$ e KSAT Versione 2: tempi di CPU per le formule φ_d^K.

descritte come segue: [5]

$$\varphi_d^K = A_0 \wedge \neg A_1 \wedge \bigwedge_{i=0}^{d} \Box^i(depth \wedge determined \wedge branching),$$

dove:

$$depth = \textstyle\bigwedge_{i=1}^{d+1}(A_i \supset A_{i-1}),$$

$$\begin{aligned} determined = \textstyle\bigwedge_{i=1}^{d}(A_i \supset (\,(B_i \supset \Box(A_i \supset B_i)) \ \wedge \\ (\neg B_i \supset \Box(A_i \supset \neg B_i)))), \end{aligned}$$

$$\begin{aligned} branching = \textstyle\bigwedge_{i=0}^{d-1}(A_i \wedge \neg A_{i+1}) \supset (\,\neg\Box\neg(A_{i+1} \wedge \neg A_{i+2} \wedge B_{i+1}) \wedge \\ \neg\Box\neg(A_{i+1} \wedge \neg A_{i+2} \wedge \neg B_{i+1}))). \end{aligned}$$

Queste formule sono paradigmatiche per la logica K(1), in quanto ogni modello M che soddisfi φ_d^K è tale che

$$|M| \geq 2^{d+1} - 1,$$

[5]Scriviamo $\Box$ per $\Box_1$ e $\Box^i$ intendendo la ripetizione $\Box\Box\ldots\Box$ i volte.

mentre $|\varphi_d^K| = O(d^2)$. Da questo possiamo assumere che le dimensioni delle spazio di ricerca crescano al minimo come 2^d per ogni algoritmo "standard" per K-soddisfacibilità come KSAT e tutte le procedure tableau-based (cioè tale che, cerchi di costruire depth-first un modello di Kripke per la formula in ingresso). Abbiamo provato TABLEAU, KRIS, KSAT$_s$ e KSAT Versione 2 (come sopra compilate ed eseguite con `Allegro CL 4.2` su una workstation `SUN SPARC10 32M`) sulle formule φ_d^K, per valori crescenti di d. [6] I risultati sono rappresentati in Figura 7.6. Le curve di TABLEAU, KRIS, KSAT$_s$ e KSAT Versione 2 crescono tutte con andamento esponenziale, circa come $(16.0)^d$, $(12.7)^d$, $(2.6)^d$ e $(2.4)^d$ rispettivamente, e superando il limite superiore di mille secondi $d = 6$, $d = 7$, $d = 11$ e $d = 12$. La differenza nelle prestazioni tra KSAT$_s$ e KSAT Versione 2 è dovuta all'overhead introdotto dalla fattorizzazione delle componente $\bigwedge_i \alpha_{ri}$ (Sezione 5.3), operazione inutile ed onerosa con queste formule. Tracciando il numero globale di assegnamenti trovati da KSAT$_s$ e KSAT Versione 2 (lo stesso per entrambe le versioni) si scopre che tale numero risulta essere $2^{d+1} - 1$ per ogni φ_d^K, cioè esattamente il numero minimo di stati per M. Se ne deduce che nè KSAT$_s$ nè KSAT Versione 2 hanno trovato assegnamenti ridondanti.

7.3 Confronto tra sistemi

Nei test descritti in questa sezione abbiamo confrontato KSAT$_s$ e KRIS, cioè le più efficienti procedure SAT-based e tableau-based a nostra disposizione. Abbiamo effettuato tre esperimenti su 48000 formule random 3CNF$_{\Box m}$, con gli stessi criteri presentati nella Sezione 7.2. I risultati sono presentati nelle Figure 7.7 e 7.8. Nella prima rappresentiamo le percentuali di soddisfacibilità delle formule (valutate con KSAT$_s$). Questo può essere preso come un'indicazione del livello di constraintness delle formule generate. Nella seconda rappresentiamo i tempi di CPU mediani ottenuti con KRIS e KSAT$_s$. Questa dà un'immagine globale del comportamento quali-

[6] L'idea di effettuare un test sulle formule φ_d^K è stata suggerita da Fabio Massacci.

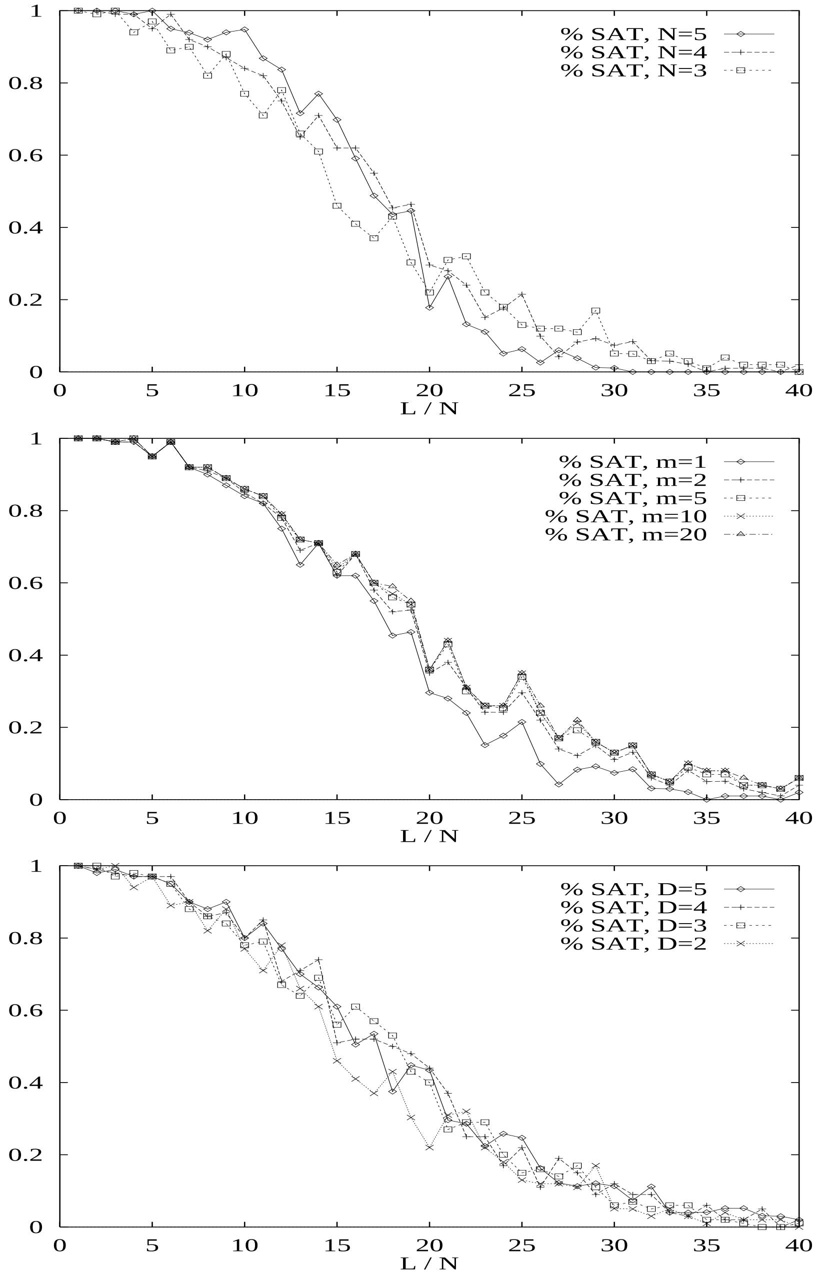


Figura 7.7: Percentuale di formule soddisfacibili. 1^a riga: Esp. 1, $d = 2$, $m = 1$, $p = 0.5$, $N = 3, 4, 5$. 2^a riga: Esp. 2, $d = 2$, $m = 1, 2, 5, 10, 20$, $p = 0.5$, $N = 4$. 3^a riga: Esp. 3, $d = 2, 3, 4, 5$, $m = 1$, $p = 0.5$, $N = 3$.

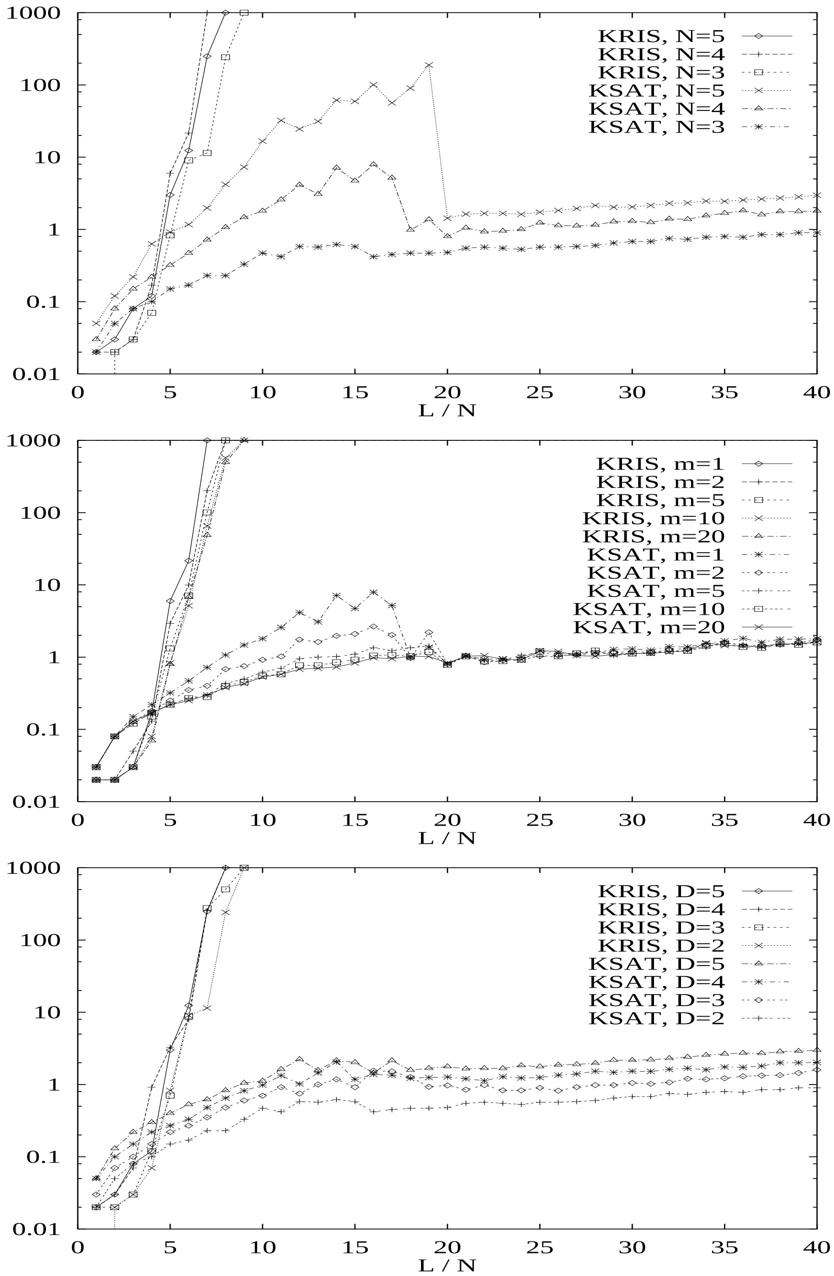


Figura 7.8: Tempi mediani di CPU di KRIS e KSAT$_s$. 1^a riga: Esp. 1, $d = 2$, $m = 1$, $p = 0.5$, $N = 3, 4, 5$. 2^a riga: Esp. 2, $d = 2$, $m = 1, 2, 5, 10, 20$, $p = 0.5$, $N = 4$. 3^a riga: Esp. 3, $d = 2, 3, 4, 5$, $m = 1$, $p = 0.5$, $N = 3$.

tativo di KRIS e KSAT$_s$, e permette un confronto esaustivo sul loro comportamento. Tutte le curve rappresentano 100 formule/punto. Come nelle sezioni precedenti, il range $1 \ldots 40$ per L/N è stato scelto empiricamente per coprire la transizione "100% soddisfacibile – 100% insoddisfacibile". In ogni esperimento analizziamo gli effetti della variazione di un parametro fissati gli altri. Una descrizione più dettagliata, esperimento per esperimento, è riportata nel Capitolo 9, nelle Figure 9.1, 9.2 e 9.3. Nel primo esperimento (prima riga) fissiamo $d = 2$, $m = 1$, $p = 0.5$ e disegniamo le curve risultanti per valori crescenti del numero di variabili $N = 3, 4, 5$. [7] Nel secondo esperimento (seconda riga) fissiamo $d = 2$, $N = 4$, $p = 0.5$ e disegniamo le curve risultanti per valori crescenti del numero di box distinte $m = 1, 2, 5, 10, 20$. Nel terzo esperimento (terza riga) fissiamo $m = 1$, $N = 3$, $p = 0.5$ e disegniamo le curve risultanti per valori crescenti della profondità modale $d = 2, 3, 4, 5$.

I risultati presentati confermano quanto già detto nella sezione precedente riguardo ai risultati di Figura 7.5: le prestazioni di KSAT superano drasticamente quelle di KRIS in tutti i test presentati, indipendentemente dal numero di variabili N, dal numero di modalità m o dalla profondità d considerati. Di nuovo, questo divario non è solo qualitativo — fino a tre o quattro ordini di grandezza, con le solite considerazioni sulle possibili estrapolazioni delle curve — ma soprattutto qualitativo: mentre le curve di KRIS crescono esponenzialmente con L, le curve di KSAT$_s$ crescono polinomialmente (combinando un picco "easy-hard-easy" con una componente lineare).

[7] Se confrontiamo le curve di KSAT$_s$ e KRIS in Figura 7.5 con le corrispondenti curve per $L = 3$, $d = 2$, $m = 1$, $p = 0.5$ in Figura 7.8, notiamo che le curve sono differenti, anche se sono valutate su formule con gli stessi parametri. Questo è dovuto al fatto che (i) i primi sono calcolati su una workstation molto più potente (ii) gli insiemi di campioni sono diversi, in quanto originati da diversi "semi" del generatore random.

Capitolo 8

Spiegazione dei risultati

"I've come to bury Caesar, not to praise him"
[W. Shakespeare, "Anthony and Cleopatra"]

In questo capitolo presentiamo una spiegazione teorica che giustifichi il divario qualitativo e quantitativo tra le prestazioni tra procedure DPLL-based e tableau-based. Concentriamoci sugli algoritmi di base e consideriamo KSAT Versione 2 (Sezione 5.2) e TABLEAU (Sezione 3.3). Entrambi lavorano

(i) enumerando assegnamenti ("rami" per i tableau) che soddisfino proposizionalmente la formula in ingresso, e
(ii) verificando ricorsivamente la K(m)-soddisfacibilità degli assegnamenti trovati.

Le due procedure effettuano il secondo passo allo stesso modo. La differenza fondamentale sta nel modo in cui KSAT Versione 2 e TABLEAU gestiscono l'inferenza *proposizionale*. Rispetto a DPLL, i tableau proposizionali presentano due debolezze che li rendono *intrinsecamente* meno efficienti, i cui effetti negativi vengono esponenzialmente amplificati durante l'inferenza modale: l'uso di branching "sintattico" anziché "semantico" e l'incapacità di rivelare "violazione di vincoli". Consideriamole in ordine.

8.1 Branching sintattico contro branching semantico

Nelle procedure tableau-based gli assegnamenti sono implicitamente generati come rami di un tableau proposizionale. Questi ultimi effettuano un "*branching sintattico*", cioè ramificano lo spazio di ricerca in base alla struttura sintattica della formula. In particolare, come discusso in [D'Agostino, 1992; D'Agostino e Mondadori, 1994], ogni applicazione della regola "$\vee$-elimination" genera due sottoalberi *mutuamente consistenti*, [1] cioè che possono condividere modelli. Il numero degli assegnamenti generati cresce esponenzialmente con il numero di disgiunzioni che occorrono positivamente in φ (nei nostri test, questo valore è linearmente proporzionale a L), indipendentemente dal numero effettivo dei modelli della formula. Di conseguenza, l'insieme degli assegnamenti generati da un tableau proposizionale è *intrinsecamente* ridondante, e può contenere fino ad un numero esponenziale di assegnamenti duplicati o sussunti.

Il problema peggiora considerevolmente nel caso modale, dove le regole tableau proposizionali sono usate per generare non *uno* ma *tutti* gli assegnamenti, la cui soddisfacibilità deve essere poi verificata. Si noti che il numero degli assegnamenti può essere enorme, fino a molte migliaia nei nostri test. Questo richiede di verificare ricorsivamente molte sottoformule nella forma $\bigwedge_i \alpha_{ri} \wedge \beta_j$ di profondità $d-1$, per le quali verranno generati assegnamenti, e così via. Ogni assegnamento di profondità k ridondante generato introduce un albero di ricerca ridondante di altezza k. Sempre peggio, tale ridondanza proposizionale si propaga con la profondità d nelle sottoformule di φ.

Nelle procedure DPLL-based, invece, gli assegnamenti sono generati da DPLL. Le procedure DPLL-based effettuano una ricerca

[1] Come sottolineato in [D'Agostino, 1992; D'Agostino e Mondadori, 1994], le regole dei Tableau non sono in grado di rappresentare il *principio di bivalenza*: "ogni proposizione è o vera o falsa, *tertium non datur*". Questo fatto è conseguenza dell'eliminazione della regola del "cut" [Gentzen, 1969] nel calcolo dei sequenti, da cui i tableau sono derivati.

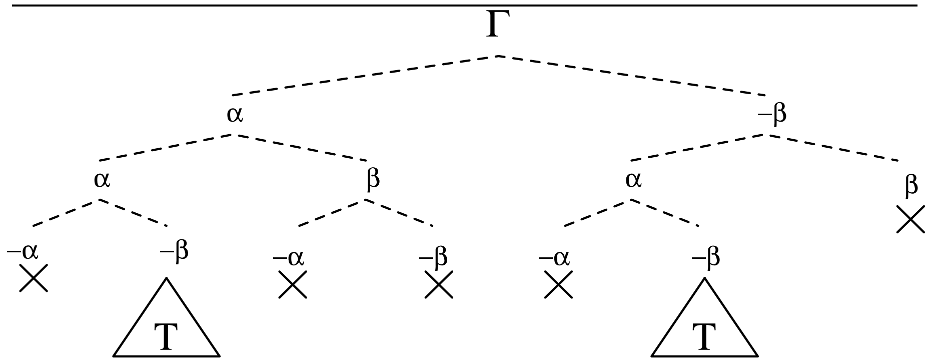


Figura 8.1: Tableau per la formula $\Gamma = (\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$.

basata su quello che chiamiamo "*branching semantico*", cioè ramificano lo spazio di ricerca sui diversi valori di verità da assegnare agli atomi. In questo modo, gli assegnamenti generati sono sempre *mutuamente inconsistenti*. Si evita così ogni ridondanza proposizionale, e conseguentemente ogni propagazione di tale ridondanza sulla struttura modale della formula. Si noti che in KSAT abbiamo utilizzato non-CNF DPLL, ma potremmo indifferentemente usare ogni altra procedura SAT che non risenta dei due problemi qui evidenziati, come OBDD o KE. Tutte le considerazioni fatte in questo capitolo per DPLL valgono anche per tali algoritmi.

Esempio 8.1.1 Si consideri la formula

$$\Gamma = (\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta),$$

dove α e β sono atomi modali, e sia $d = depth(\Gamma)$. L'unico assegnamento possibile che soddisfi Γ è $\mu = \alpha \wedge \neg\beta$, che supponiamo non K(m)-soddisfacibile. Si consideri il tableau rappresentato in Figura 8.1, dove la regola di $\vee$-elimination viene applicata alle clausole di Γ in ordine, da sinistra a destra. Vengono quindi generati due rami aperti identici, entrambi rappresentanti μ, che vengono espansi in due sub-tableau identici T, ciascuno dei quali prova la K(m)-insoddisfacibilità di μ (si noti che i due T possono essere molto grandi). Questo fatto può ripetersi ricorsivamente molte volte ad ogni livello in ciascun sub-tableau T. Per esempio, se

$$\begin{array}{rcl}\alpha & = & \Box_1((\alpha' \vee \neg\beta') \wedge (\alpha' \vee \beta')),\\ \beta & = & \Box_1(\alpha' \wedge \beta'),\end{array}$$

allora al livello immediatamente inferiore abbiamo una formula Γ' di profondità $d-1$ analoga a Γ. Questo propaga la ridondanza esponenzialmente in $depth(d)$. Infine, se consideriamo invece la formula

$$\Gamma^k = \bigwedge_{i=1}^{k} (\alpha_i \vee \neg\beta_i) \wedge (\alpha_i \vee \beta_i) \wedge (\neg\alpha_i \vee \neg\beta_i),$$

il tableau genererebbe 2^k assegnamenti identici $\mu^k = \bigwedge_i \alpha_i \wedge \neg\beta_i$, e le cose andrebbero esponenzialmente peggio.

Una procedura DPLL-based applicata a Γ, invece, genera due rami mutuamente contraddittori $\alpha = T$ e $\alpha = F$. Il primo ramo genera $\mu = \alpha \wedge \neg\beta$, mentre il secondo genera $\neg\alpha \wedge \neg\beta \wedge \beta$ che viene chiuso in quanto contiene una contraddizione. Viene quindi generata un'unica istanza di μ. Lo stesso accade a μ^k. $\Box$

8.2 Rilevamento della violazione di vincoli

La spiegazione precedente lascia ancora due domande senza risposta. In primo luogo, la Figura 7.4 mostra un divario di prestazioni tra KSAT Versione 2 e MODIFIED TABLEAU (più di un fattore 30 per $L/N = 12$). Questo fatto non si spiega ancora, in quanto MODIFIED TABLEAU genera sottoalberi mutuamente inconsistenti, in quanto basato sulle regole $\frac{\varphi\vee\psi}{\varphi \quad \psi,\neg\varphi}$, $\frac{\varphi\vee\psi}{\varphi,\neg\psi \quad \psi}$ (si veda Sezione 7.1). In secondo luogo, Figura 7.5 (ma si veda anche il Capitolo 9) mostra che l'andamento della dimensione dello spazio di ricerca KSAT dopo un certo valore decresce con la dimensione della formula. Questi due fatti richiedono un'ulteriore spiegazione.

Una formula proposizionale φ può essere vista come un insieme di vincoli per gli assegnamenti che la soddisfano [Williams e Hogg,

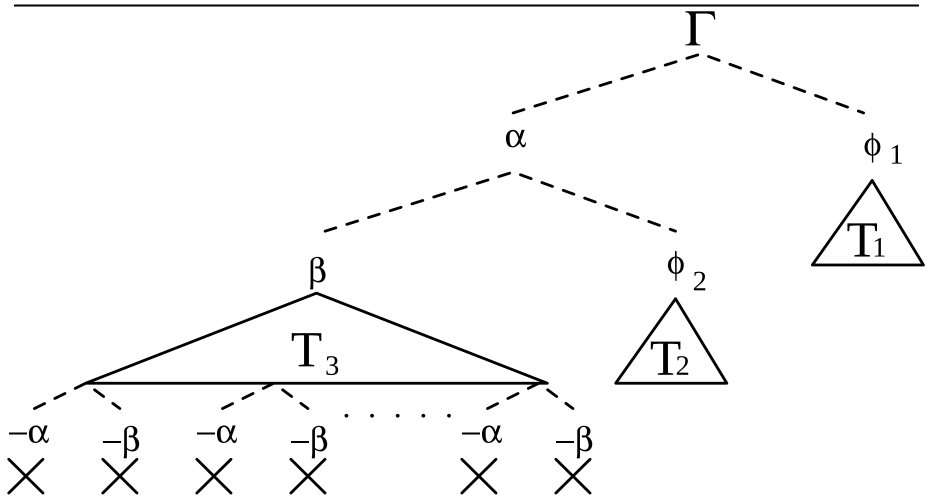


Figura 8.2: Tableau per la formula $\Gamma = (\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta)$.

1994]. Per esempio, una clausola $A_1 \vee A_2$ vincola ogni assegnamento ad assegnare T ad A_1 o ad A_2. Contrariamente ai tableau, DPLL taglia i rami non appena violano qualche vincolo della formula d'ingresso: se μ falsifica una clausola $C \in \varphi$, allora *assign* restituisce *False* e DPLL esegue un backtracking. Più vincolata è la formula, più facilmente un assegnamento viola qualche vincolo (per esempio, più grande è L in una formula CNF, più facilmente un assegnamento genera una clausola vuota). Quando φ diventa fortemente vincolata (cioè per L sufficientemente grande) l'albero di ricerca viene quindi fortemente potato. Di conseguenza, per L maggiore di un certo valore, la dimensione dello spazio di ricerca *decresce* con L.

Esempio 8.2.1 Si consideri la formula

$$\Gamma = (\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta),$$

dove α e β sono atomi, ϕ_1, ϕ_2 e ϕ_3 sottoformule, tali che $\alpha \wedge \beta \wedge \phi_3$ sia soddisfacibile. Si consideri il tableau di Figura 8.2, dove la regola di $\vee$-elimination viene applicata in ordine, da sinistra a destra.

Dopo due passi viene generato il ramo α, β, che viola il vincolo imposto dall'ultima clausola $(\neg\alpha \vee \neg\beta)$. Una procedura tableau-based non è in grado di accorgersi di tale violazione finché non analizza esplicitamente quella clausola, cioè solo dopo aver generato l'intero sub-tableau T_3 per $\alpha \wedge \beta \wedge \phi_3$, che può essere molto grande. Solo allora può chiudere il sub-tableau T_3 creando, per ciascun ramo aperto di T_3, i due sotto-rami $\neg\alpha$ e $\neg\beta$. DPLL, invece, rivela la violazione e immediatamente taglia il ramo. Per esempio, in KSAT questo viene fatto dalla funzione *assign*. □

È interessante osservare che le considerazioni sulla violazione di vincoli forniscono una spiegazione del fatto che, in tutti i fenomeni di transizione di fase, il picco delle curve "easy-hard-easy" si trovi all'*interno* della transizione "100% soddisfacibile - 100% insoddisfacibile". Questa spiegazione è indipendente dalla logica considerata. Consideriamo una formula insoddisfacibile φ. Se le aggiungiamo una nuova clausola/vincolo C, allora $\varphi \wedge C$ resta insoddisfacibile e l'aggiunta di C permette a DPLL di tagliare alcuni rami in anticipo (cioè non appena violino C), riducendo la dimensione dell'albero di ricerca. Conseguentemente (Figura 8.3) all'interno della zona "100% insoddisfacibile", lo spazio di ricerca rimpicciolisce con L/N. Se invece eliminiamo una clausola da una formula soddisfacibile ψ, questo aumenta il numero di rami soddisfacibili, diminuendo quindi la dimensione dell'albero di ricerca "visitato" per trovarne uno. Di conseguenza (Figura 8.3) all'interno della zona "100% soddisfacibile", lo spazio di ricerca cresce con L/N. Questi due fatti, insieme, posizionano il picco all'interno della zona di transizione. (Si noti che il branching semantico, invece, ha l'effetto di ridurre lo spazio di ricerca indipendentemente dalle zone considerate, in quanto riduce il numero degli assegnamenti.)

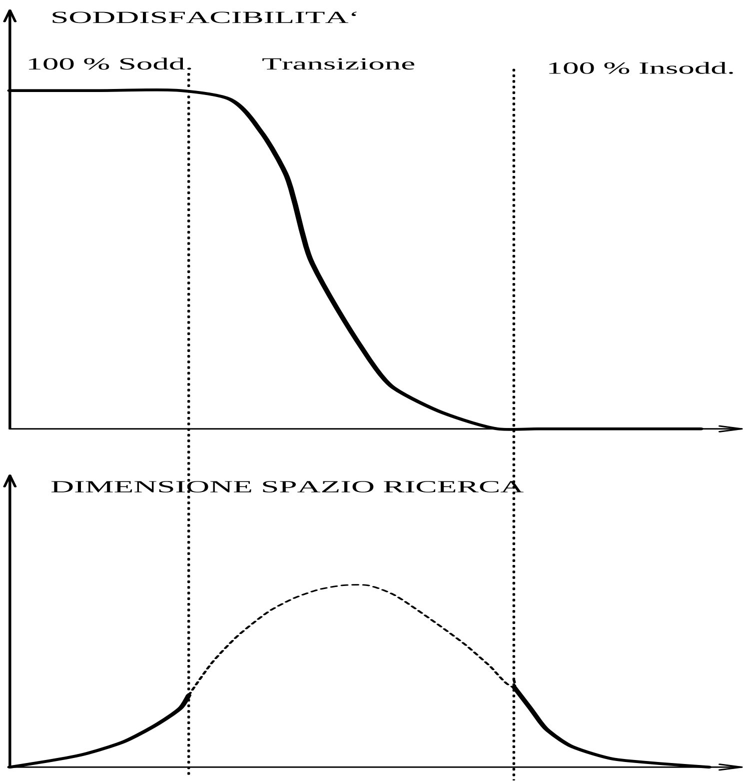


Figura 8.3: Confronto teorico tra la curva di soddisfacibilità (sopra) e la curva della dimensione dello spazio di ricerca (sotto).

Capitolo 9

Un'analisi empirica di K(m)

In questo capitolo ritorniamo a considerare i risultati degli esperimenti descritti nella Sezione 7.3, e utilizziamo tali risultati per fare alcune considerazioni qualitative. I risultati sono raggruppati, esperimento per esperimento, nelle figure 9.1, 9.2 e 9.3. In ciascun esperimento sono riportati le percentuali di formule soddisfacibili (prima riga), i tempi mediani di CPU impiegati da KSAT_s e KRIS[1] (seconda riga) e il numero di chiamate ad *assign* di KSAT_s, cioè la dimensione dello spazio di ricerca (terza riga). Riportiamo sotto (Sezione 9.1) una prima analisi, passo-passo, dei risultati. Questo ci permette di capire come la difficoltà delle formule e la loro soddisfacibilità varino in funzione dei vari parametri. Nella Sezione 9.2 riportiamo un'analisi globale, e per certi aspetti più interessante, dei risultati. Questo rivela l'esistenza di un fenomeno di "transizione di fase" per K(m).

9.1 Analisi "test a test"

I risultati del primo esperimento (Figura 9.1) mostrano come, aumentando il numero di variabili N (e proporzionatamente il numero di clausole L), si ha un rilevante aumento della difficoltà delle formule

[1]In questo capitolo ignoreremo i grafici relativi a KRIS.

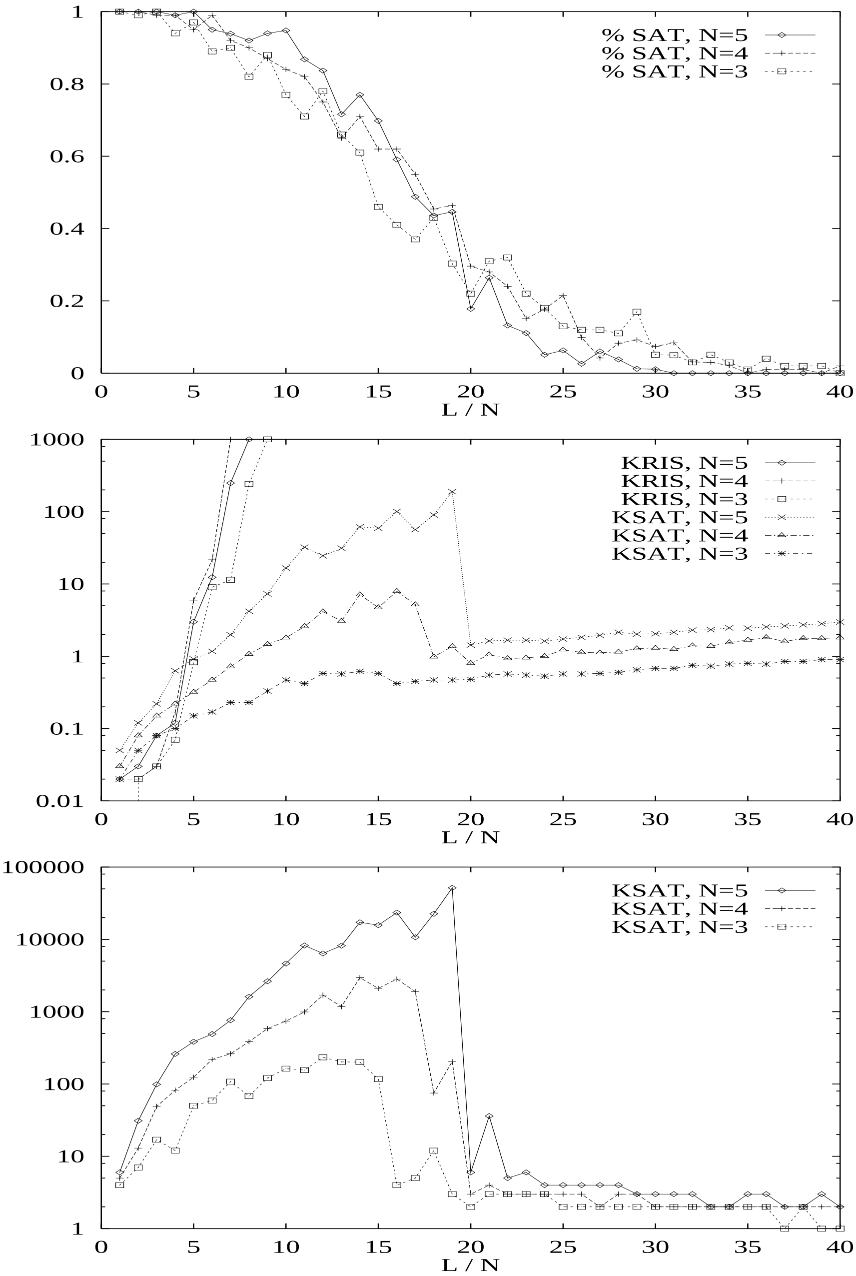


Figura 9.1: Esp. 1: $d = 2$, $m = 1$, $p = 0.5$, $N = 3, 4, 5$. 1^a riga: percentuale di formule soddisfacibili. 2^a riga: tempi mediani di CPU di Kris e Ksat_s. 3^a riga: dimensione mediana dello spazio di ricerca di Ksat_s.

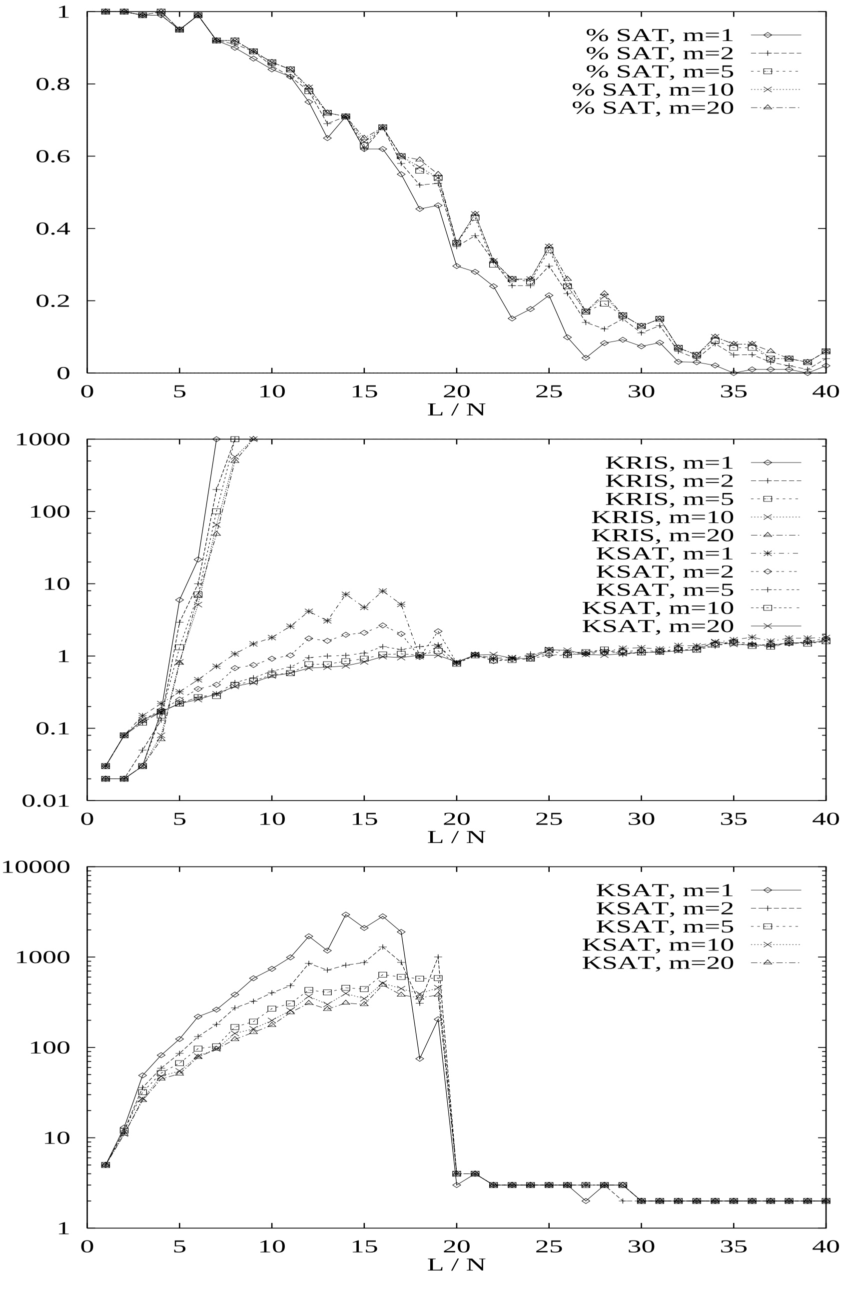


Figura 9.2: Esp. 2: $d = 2$, $m = 1, 2, 5, 10, 20$, $p = 0.5$, $N = 4$. 1^a riga: percentuale di formule soddisfacibili. 2^a riga: tempi mediani di CPU di Kris e Ksat_s. 3^a riga: dimensione mediana dello spazio di ricerca di Ksat_s.

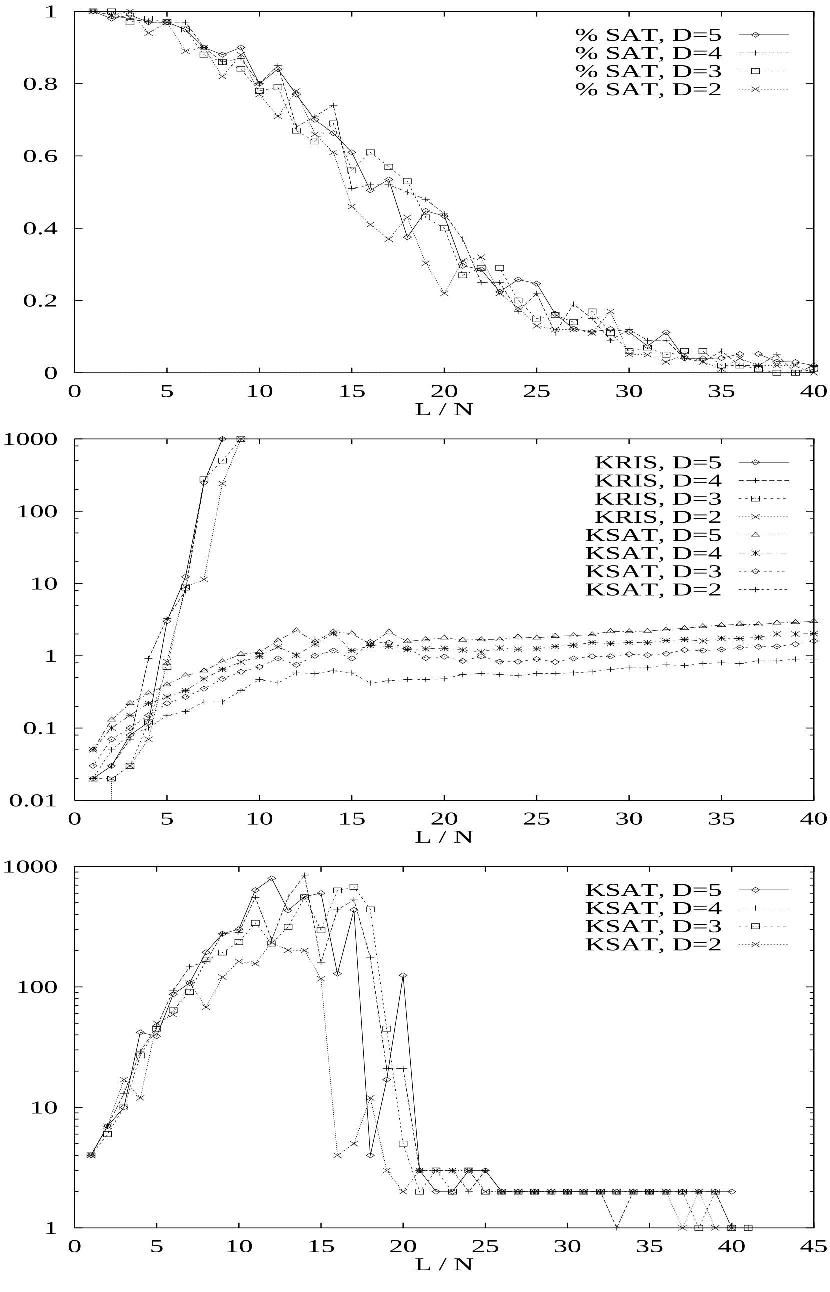


Figura 9.3: Esp. 3: $d = 2, 3, 4, 5$, $m = 1$, $p = 0.5$, $N = 3$. 1^a riga: percentuale di formule soddisfacibili. 2^a riga: tempi mediani di CPU di Kris e Ksat_s. 3^a riga: dimensione mediana dello spazio di ricerca di Ksat_s.

— fino a un ordine di grandezza per variabile per KSAT$_s$, fino a due ordini di grandezza per KRIS. Questo fatto non dovrebbe sorprendere, in quanto in K(m) l'aggiunta di poche variabili può causare una crescita esponenziale dello spazio di ricerca. Ogni variabile, infatti, assume valori diversi nei diversi stati/mondi delle strutture di Kripke, cioè ogni variabile deve venire considerata con una "molteplicità implicita" equivalente al numero di stati delle possibili strutture di Kripke (il cui numero può essere esponenziale nella dimensione della formula [Ladner, 1977; Halpern e Moses, 1992]).

I risultati del secondo esperimento (Figura 9.2) presentano due risultati interessanti. In primo luogo, la difficoltà delle formule *decresce* con il numero m delle modalità, sia per KSAT$_s$ che per KRIS. Questo fatto, che a prima vista può sorprendere, ha una facile spiegazione. Ogni assegnamento μ è partizionato in m assegnamenti indipendenti μ^r, ristretti alle singole $\Box_r$ (si vedano le Equazioni (4.1) e (4.2)). Questo significa applicare il principio del "divide et impera" allo spazio di ricerca, cioè dividere l'albero di ricerca in m sottoalberi indipendenti. Più grande è m, più partizionabile è lo spazio di ricerca, più semplice da risolvere è il problema. In secondo luogo, una lettura attenta dei grafici rivela che la percentuale di formule soddisfacibili *aumenta* con m. Anche questo fatto è facilmente spiegabile. Non essendoci mutua dipendenza tra la soddisfacibilità dei singoli assegnamenti μ^r, più grande è m, meno vincolato è il problema, più probabilmente soddisfacibile è φ.

I risultati del terzo esperimento (Figura 9.3) mostrano come la difficoltà delle formule cresca con la profondità d, sia per KSAT che per KRIS. Anche questo fatto è intuitivo: maggiore è d, maggiore è la profondità dei modelli di Kripke da cercare, maggiore è la complessità della ricerca.

9.2 Analisi globale

Le curve di KSAT (seconda riga) evidenziano l'esistenza di due componenti, un picco "easy-hard-easy", dipendente dallo spazio di ricerca, e una componente lineare, dovuta alla funzione *assign* che viene invocata ad ogni passo ricorsivo. Infatti, se aumentiamo N da 3 a 5 e L di conseguenza (Figura 9.1), la dimensione dello spazio di ricerca ha un aumento rilevante. Mentre con $N = 3$ la componente lineare prevale, per $N = 5$, la componente easy-hard-easy domina. Inoltre, variando il numero di modalità mantenendo inalterata la dimensione delle formule (Figura 9.2), l'effetto della componente easy-hard-easy scompare ($L/N > 20$), e le curve del tempi di CPU collassano insieme, poichè la componente lineare, causata da *assign*, non dipende da m. Si noti che la posizione delle zone easy/hard/easy sembrano non variare significativamente nè con il numero di variabili N (Figura 9.1), nè con il numero di modalità m (Figura 9.2), nè con la profondità d (Figura 9.3).

Consideriamo ora le curve di soddisfacibilità nelle tre figure (prima riga). Nonostante il rumore e le approssimazioni dovute ai timeout, è facile notare che i punti "50% soddisfacibile" sono centrati intorno ai valori $L/N = 15 \sim 20$ in tutti gli esperimenti. Inoltre, in modo simile a quanto rivelato da [Mitchell *et al.*, 1992; Kirkpatrick e Selman, 1994], nel primo esperimento un occhio attento rivela che la transizione tra soddisfacibilità e insoddisfacibilità diventa più ripida all'aumentare di L (per esempio, si confrontino le curve $N = 3$ e $N = 5$). Infine nelle curve della dimensione dello spazio di ricerca di tutti gli esperimenti (terza riga) i picchi si trovano nell'area di transizione di soddisfacibilità (anche se, contrariamente a [Mitchell *et al.*, 1992; Kirkpatrick e Selman, 1994], sembrano anticipare un po' i punti "50%".

Da questi fatti possiamo congetturare (da verificare ulteriormente) l'esistenza del fenomeno di "transizione di fase" per K(m)/$\mathcal{ALC}$, simile a quelli ben noti per SAT [Mitchell *et al.*, 1992; Kirkpatrick e Selman, 1994] e molti altri problemi NP-hard [Cheeseman *et al.*, 1991; Gent e Walsh, 1996]. Per quanto a nostra conoscenza, questa è la prima volta che tale fenomeno viene rivelato per logiche modali. (Si ricordi che la K(m)-soddisfacibiltà è un problema PSPACE-complete). Questa congettura è supportata anche dall'analisi presentata nella Sezione 8.2 (Figura 8.3).

Capitolo 10

Conclusioni e lavoro futuro

Questa tesi presenta una serie di nuovi risultati, che possono essere riassunti come segue.

1. Introduce un nuovo approccio, chiamandolo "SAT-based", per costruire procedure di decisione *efficienti* per logiche modali e terminologiche (ed eventualmente altre logiche). L'idea è quella di costruire tali procedure "sopra" procedure di decisione per logica proposizionale, sostituendo con queste ultime le regole tableau proposizionali all'interno delle "classiche" procedure tableau-based.

2. Restringendosi al caso di studio della logica K(m), componente comune di tutte le logiche considerate, descrive KSAT, una nuova procedura SAT-based per tale logica. Presenta poi una serie di test estensivi, da cui risulta che le prestazioni di KSAT superano di vari ordini di grandezza quelle di un decisore tableau-based allo stato dell'arte, KRIS.

3. Presenta una spiegazione dei risultati empirici ottenuti. In particolare, dimostra che le procedure SAT-based sono intrinsecamente più efficienti delle procedure tableau-based, evidenziando due grosse cause di inefficienza di queste ultime.

4. Presenta l'evidenza sperimentale, seppure ancora parziale, della presenza del fenomeno di "transizione di fase" per K(m). Se

tale evidenza verrà confermata, questa è la prima volta che tale fenomeno, ben noto per SAT e per altri problemi NP-hard, viene rivelato nelle logiche modali.

Il lavoro presentato suggerisce inoltre futuri sviluppi, lungo il piano delineato da due strade mutuamente ortogonali.

- Descrivere e realizzare procedure SAT-based per altre logiche, seguendo le linee guida presentate nei Capitoli 4 e 5. Particolarmente interessante, a nostro avviso, è il caso delle logiche dinamiche e temporali (in particolare della logica CTL), in virtù delle loro applicazioni nel campo dei metodi formali di verifica di sistemi e protocolli.
- Sperimentare l'uso di decisori proposizionali alternativi a DPLL. Particolarmente interessante, a nostro avviso, è l'uso degli OBDD, a causa delle proprietà evidenziate nella Sezione 3.1.3. Può essere inoltre interessante considerare la realizzazione di procedure incomplete, sulla base di GSAT o di procedure analoghe.

È inoltre possibile creare sistemi di ragionamento automatico *modulari*, dove la scelta della logica e/o del decisore proposizionale da utilizzare venga lasciata all'utente.

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.

[Auffray e Enjalbert, 1992] Y. Auffray e P. Enjalbert. Modal Theorem Proving: An Equational Viewpoint. *Journal of Logic and Computation*, 2(3):247–295, 1992.

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, e H.J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[Baader e Hollunder, 1991] F. Baader e B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proceedings of the First International Workshop on Processing Declarative Knowledge*, volume 572 of *Lecture Notes in Computer Science*, pp. 67–85, Kaiserslautern (Germany), 1991. Springer–Verlag.

[Borga, 1984] M. Borga. *Elementi di logica matematica.* La Goliardica, 1984.

[Boyer e Moore, 1983] R.S. Boyer e J.S. Moore. Proof checking, theorem proving and program verification. In Bledsoe e Loveland, editors, *Automated Theorem Proving: After 25 Years*, pp. 119–132. American Mathematical Society, 1983.

[Brachman e Levesque, 1984] R. J. Brachman e H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 5th National Conference on Artificial Intelligence*, Austin, TX, 1984.

[Bresciani *et al.*, 1995] P. Bresciani, E. Franconi, e S. Tessaris. Implementing and testing expressive Description Logics: a preliminary report. In *Proc. International Workshop on Description Logics*, Rome, Italy, 1995.

[Bryant, 1992] R. E. Bryant. Symbolic Boolean manipolation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Bull e Segerberg, 1984] R. Bull e K. Segerberg. Basic Modal Logic. In D. Gabbay e F. Guenthner, editors, *Handbook of Philosophical Logic*, volume III, pp. 1–88. D. Reidel Publishing Company, 1984.

[Bundy *et al.*, 1996a] A. Bundy, F. Giunchiglia, R. Sebastiani, e T. Walsh. Calculating Criticalities. *Artificial Intelligence*, 88(1-2):39–67, December 1996.

[Bundy *et al.*, 1996b] A. Bundy, F. Giunchiglia, R. Sebastiani, e T. Walsh. Computing Abstraction Hierarchies by Numerical Simulation. In *Proc. of the 13th National Conference on Artificial Intelligence*, August 1996.

[Buro e Buning, 1992] M. Buro e H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Camilleri *et al.*, 1987] A. Camilleri, M. Gordon, e T. Melham. hardware verification using higher-order logic. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, pp. 43–67. Elsevier Science Publishers B.V. (North-Holland), 1987.

[Catach, 1991] L. Catach. TABLEAUX: a general theorem prover for modal logics. *Journal of Automated Reasoning*, 7, 1991.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefski, e W. M. Taylor. Where the really hard problem are. In *Proc. of the 12th International Joint Conference on Artificial Intelligence*, pp. 163–169, 1991.

[Chellas, 1980] B. F. Chellas. *Modal Logic – an Introduction.* Cambridge University Press, 1980.

[Cook, 1971] S. A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on the Theory of Computation*, pp. 151–158, 1971.

[Crawford e Auton, 1993] J. Crawford e L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proc. of the 11th National Conference on Artificial Intelligence*, pp. 21–27, 1993.

[Crawford e Auton, 1996] J. Crawford e L. Auton. Experimental results on the crossover point in 3SAT. *Artificial Intelligence*, 81, 1996.

[D'Agostino e Mondadori, 1994] M. D'Agostino e M. Mondadori. The Taming of the Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[D'Agostino, 1992] M. D'Agostino. Are Tableaux an Improvement on Truth-Tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.

[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.

[de la Tour, 1990] T. Boy de la Tour. Minimizing the Number of Clauses by Renaming. In *Proc. of the 10th Conference on Automated Deduction*, pp. 558–572. Springer-Verlag, 1990.

[DeGiacomo e Lenzerini, 1994] G. DeGiacomo e M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th National Conference on Artificial Intelligence*, pp. 205–212, 1994.

[DeGiacomo e Massacci, 1996] G. DeGiacomo e F. Massacci. Tableaux and Algorithms for Propositional Dynamic Logic with Converse. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996.

[Demri, 1995] S. Demri. Uniform and Nonuniform strategies for tableaux calculi for modal logics. *Journal of Applied Nonclassical Logics*, 5(1):77–98, 1995.

[DIM, 1993] DIMACS. *The Second DIMACS International Algorithm Implementation Challenge*, Rutgers University, USA, 1993.

[Donini *et al.*, 1992] F. Donini, M. Lenzerini, D. Nardi, B. Hollunder, W. Nutt, e A. Marchetti Spaccamela. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53:309–327, 1992.

[Enjalbert e del Cerro, 1989] P. Enjalbert e L. Farinas del Cerro. Modal Resolution in Clausal Form. *Theoretical Computer Science*, 65:1–33, 1989.

[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.

[Fitting, 1988] M. Fitting. First-Order Modal Tableaux. *Journal of Automated Reasoning*, 4:191–213, 1988.

[Franco e Paull, 1983] J. Franco e Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[Garey e Johnson, 1979] M. R. Garey e D. S. Johnson. *Computers and Intractability.* Freeman and Company, New York, 1979.

[Gent e Walsh, 1993] Ian P. Gent e Toby Walsh. Towards an Understanding of Hill-climbing Procedures for SAT. In *Proceedings AAAI-11*, 1993.

[Gent e Walsh, 1996] I. P. Gent e T. Walsh. The TSP phase transition. *Artificial Intelligence*, 88(1-2):349–358, December 1996.

[Gentzen, 1969] G. Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pp. 68–128. North Holland Publishing Company, 1969.

[Ginsberg, 1993] M. L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[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 *et al.*, 1997] F. Giunchiglia, M. Roveri, e R. Sebastiani. A new method for testing decision procedures in modal and terminological logics. In *Proc. of the 14th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Townsville, Australia, July 1997. Springer Verlag. In print.

[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. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). Technical Report 9611-06, IRST, Trento, Italy, 1996. A shorter version is submitted to "Information and Computation".

[Giunchiglia e Sebastiani, 1996c] 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.

[Goré *et al.*, 1995] R. Goré, W. Heinle, e A. Heuerding. Relations Between Propositional Normal Logics: An Overview. Technical report, TR-ARP-16-95, Automated Reasoning Project, Australian National University, 1995.

[Halpern e Fagin, 1985] J.Y. Halpern e R. Fagin. A formal model of knowledge, action, and communication in distributed systems. In *Proc. of the Fourth ACM Symposium on the Principles of Distributed Computing*, pp. 224–236, 1985.

[Halpern e Moses, 1992] J.Y. Halpern e Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[Halpern, 1995] J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(3):361–372, 1995.

[Heuerding *et al.*, 1996] A. Heuerding, G. Jager, S. Schwendimann, e M. Seyfried. The logics workbench lwb: A snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, e M. Schmidt-Schauß. Subsumption Algorithms for Concept Description Languages. In *Proc. 8th European Conference on Artificial Intelligence*, pp. 348–353, 1990.

[Hughes e Cresswell, 1972] G. E. Hughes e M. J. Cresswell. *An Introduction to Modal Logic*. Methuen, 1972.

[Hunt, 1989] W.A. Hunt. Microprocessor design verification. *Journal of Automated Reasoning*, 5:429–460, 1989.

[Jeroslow e Wang, 1990] R.G. Jeroslow e J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, 1(1-4):167–187, 1990.

[Kirkpatrick e Selman, 1994] S. Kirkpatrick e B. Selman. Critical behaviour in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[Kleene, 1952] S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[Kleene, 1967] S.C. Kleene. *Mathematical Logic.* John Wiley and Sons Inc., 1967.

[Koshimura e Hasegawa, 1994] M. Koshimura e R. Hasegawa. Modal Propositional Tableaux in a Model Generation Theorem Prover. In *Proc. Third Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pp. 145–151, UK, 1994.

[Kowalski, 1979] R. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, 1979.

[Ladner, 1977] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comp.*, 6(3):467–480, 1977.

[Long, 1993] D. E. Long. *Model Checking, Abstraction, anc compositional Verification.* PhD thesis, School of Computer Science, Carnegie Mellon University, PA, USA, 1993.

[MacGregor, 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):93–97, june 1991.

[Manna e Pnueli, 1995] Z. Manna e A. Pnueli. *Temporal Verification of Reactive Systems.* Springer-Verlag, 1995.

[Massacci, 1994] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. of the 12th Conference on Automated Deduction*, 1994.

[Massacci, 1995] Fabio Massacci. A bridge between modal logics and contextual reasoning. In *IJCAI-95 International Workshop on Modeling Context in Knowledge Representation and Reasoning*, 1995.

[McCune, 1990] W. W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Maths and CS. Division, Argonne National Laboratory, Argonne, Illinois, 1990.

[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.

[Moore, 1994] J. S. Moore. Introduction to the OBDD Algorithm for the ATP Community. *Journal of Automated Reasoning*, 6(1):33–45, 1994.

[Morgan, 1976] C.G. Morgan. Methods of Automated Theorem Proving in Nonclassical Logics. *IEEE Transactions on Computers*, C–25(8):852–862, 1976.

[Ohlbach, 1988a] H. J. Ohlbach. A Resolution Calculus for Modal Logics. In *CADE9*, volume 310, pp. 500–516. Springer-Verlag, 1988.

[Ohlbach, 1988b] H. J. Ohlbach. Translation methods for non-classical logics - an overview. *Bulletin of the Interest Group in Pure and Applied Logic - IGPL*, 1(1), 1988.

[Owre *et al.*, 1995] S. Owre, J. Rushby, N. Shankar, e F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 20(2):107–125, February 1995.

[Patel-Schneider *et al.*, 1991] P. F. Patel-Schneider, D. L. McGuinnes, e R. J. Brachman. The CLASSIC knowledge representation system: guiding principles and implementation rational. *SIGART Bulletin*, 2(3):114–119, june 1991.

[Patel-Schneider, 1996] P. F. Patel-Schneider, editor. *Proceedings International Workshop on Description Logics - DL'96*, Cambridge, MA, USA, November 1996. AAAI Press.

[Pellettier, 1986] F. J. Pellettier. Seventy-Five Problems for Testing Authomatic Theorem Provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[Plaisted e Greenbaum, 1986] D.A. Plaisted e S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Prawitz, 1965] D. Prawitz. *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[Schild, 1991] K. D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. of the 12th International Joint Conference on Artificial Intelligence*, pp. 466–471, Sydney, Australia, 1991.

[Schmidt-Schauß e Smolka, 1991] M. Schmidt-Schauß e G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48:1–26, 1991.

[Schmidt, 1997] R. A. Schmidt. Resolution is a Decision Procedure for Many Propositional Modal Logics. Technical Report MPI-I-97-2-002, Max-Planck-Institut für Informatik, Saarbrücken, Germany, January 1997. Submitted for publication in *J. Automat. Reasoning.* The extended abstract version is to appear in Kracht, M., de Rijke, M., Wansing, H. and Zakharyaschev, M. (eds) (1997), *Advances in Modal Logic '96*, CSLI Publications.

[Schumann, 1994] J. Schumann. Tableaux-Based Theorem Provers: Systems and Implementations. *Journal of Automated Reasoning*, 13:409–421, 1994.

[Sebastiani, 1994] R. Sebastiani. Applying GSAT to Non-Clausal Formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994. Also DIST-Technical Report 94-0018, DIST, University of Genova, Italy.

[Selman *et al.*, 1992] B. Selman, H. Levesque., e D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pp. 440–446, 1992.

[Selman e Kautz, 1993] B. Selman e H. Kautz. Domain-Independent Extension to GSAT: Solving Large Structured Satisfiability Problems. In *Proc. of the 13th International Joint Conference on Artificial Intelligence*, pp. 290–295, 1993.

[Smullyan, 1968] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, NY, 1968.

[Stallmann e Sussman, 1977] R. M. Stallmann e G. J. Sussman. Forward reasoning and dependency-directed backtracking. *Artificial Intelligence*, 9(2):135–196, 1977.

[Stalmarck e Saflund, 1990] G. Stalmarck e M. Saflund. *Modelling and veryfying systems and software in propositional logic.* Pergamon Press, 1990.

[Suttner e Sutcliffe, 1995] C. B. Suttner e G. Sutcliffe. The TPTP Problem Library. Technical Report TR 95/6, James Cook University, Australia, August 1995.

[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.

[van Gelder, 1988] A. van Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79:1–21, October 1988.

[Wallen, 1987] L. Wallen. Matrix proof methods for modal logics. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pp. 917–923, 1987.

[Williams e Hogg, 1994] C. P. Williams e T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[Yugami, 1995] T. Yugami. Theoretical Analysis of Davis-Putnam Procedure and Propositional Satisfiability. In *Proc. of the 14th*

International Joint Conference on Artificial Intelligence, pp. 282–288, 1995.

[Zhang e Stickel, 1994] H. Zhang e M. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical report, University of Iowa, August 1994.