Deduzione automatica

Maria Paola Bonacina

Sommario

La deduzione automatica comprende la *dimostrazione automatica di teoremi*, per trovare prove di congetture, e la *costruzione automatica di modelli*, per trovare soluzioni di insiemi di vincoli. In deduzione automatica, la difficoltà del problema è spesso direttamente proporzionale all'espressività del linguaggio logico. La procedura di *Apprendimento di Clausole Guidato da Conflitti*, dall'inglese "Conflict-Driven Clause Learning" (CDCL), è una componente centrale dei sistemi per decidere la *soddisfacibilità in logica proposizionale*. Questa procedura limita le inferenze a quelle necessarie per *spiegare* i conflitti, e usa i conflitti per tagliare lo spazio di ricerca. Le attuali direzioni di ricerca in deduzione automatica vedono una convergenza verso l'obbiettivo di generalizzare questa nozione di *soddisfacibilità guidata da conflitti* dalla logica proposizionale alla logica del primo ordine, realizzando un paradigma di *deduzione guidata da conflitti*. Alcune direzioni di questo tipo si applicano al problema della *soddisfacibilità modulo teorie* in frammenti decidibili di teorie assiomatizzate in logica del primo ordine. In questo ambito la deduzione guidata da conflitti permette perfino una generalizzazione della classe dei problemi considerati da soddisfacibilità modulo teorie a *soddisfacibilità modulo teorie e assegnamenti*. Sviluppi non meno sorprendenti vedono l'applicazione della deduzione guidata da conflitti alla *dimostrazione automatica di teoremi in logica del primo ordine*. Queste direzioni di ricerca sono tanto promettenti quanto foriere di sfide entusiasmanti.

1 Introduzione

La forma tipica dei problemi in deduzione automatica è quella del problema di *validità*, dove ci si chiede se una congettura φ sia conseguenza logica di un insieme H di assunzioni. Poichè i metodi automatici lavorano preferibilmente in modo refutazionale, un problema di validità viene solitamente riformulato *in forma refutazionale*, chiedendosi se $H \cup \{\neg\varphi\}$ sia *insoddisfacibile*. Assunzioni,

congetture, vincoli sono formule logiche che esprimono proprietà di un oggetto di studio, come per esempio un sistema, un programma, un tipo di dati, un circuito, un protocollo, una struttura algebrica. Dal momento che i metodi automatici adottano usualmente *la forma clausale*, $H \cup \{\neg\varphi\}$ viene trasformato in un insieme $\mathcal{S}$ di clausole, interpretato come la congiunzione delle sue clausole. Una clausola è una disgiunzione di letterali con tutte le variabili implicitamente quantificate in modo universale. Alternativamente, si può voler sapere se un vincolo φ possa essere aggiunto a un insieme di vincoli H in modo tale che $H \cup \{\varphi\}$ sia *soddisfacibile*. Una volta che $H \cup \{\varphi\}$ è stato trasformato in un insieme di clausole $\mathcal{S}$, il problema è lo stesso, ovvero si tratta di determinare se $\mathcal{S}$ abbia un modello o sia insoddisfacibile. La *segnatura* del problema è l'insieme dei simboli di costante, funzione, e predicato, che appaiono in $\mathcal{S}$.

La risposta è un modello di $\mathcal{S}$ oppure una dimostrazione $\mathcal{S} \vdash \square$ che $\mathcal{S}$ è inconsistente, quindi insoddisfacibile. Il simbolo $\square$ rappresenta la clausola vuota, ovvero la contraddizione. Una dimostrazione $\mathcal{S} \vdash \square$ si dice *refutazione*. Per un problema formulato in origine come una questione di validità, una refutazione significa che φ è conseguenza logica di H, mentre un modello rappresenta un contro-esempio. Per un problema formulato in origine come una questione di soddisfacibilità, un modello rappresenta una soluzione, mentre una refutazione significa che non c'è soluzione. Questi problemi possono essere *decidibili* (validità e soddisfacibilità sono entrambe decidibili), *semidecidibili* (la validità è semidecidibile, la soddisfacibilità non è decidibile), o *indecidibili* (validità e soddisfacibilità sono entrambe indecidibili). Il primo caso si applica alla logica proposizionale, il secondo alla logica del primo ordine, e il terzo alle logiche di ordine superiore.

Un metodo o strategia di deduzione automatica è tipicamente definita da un *sistema di inferenza* e un *piano di ricerca*. Un sistema di inferenza è un insieme di regole di inferenza. Un piano di ricerca è un algoritmo equipaggiato con euristiche per controllare l'applicazione delle regole di inferenza. Un'applicazione di una regola di inferenza muove il sistema da uno *stato* della *derivazione* al successivo. Quando il problema è decidibile, ci si aspetta che la strategia di deduzione automatica sia una *procedura di decisione*. Per essere una procedura di decisione, una strategia deve essere *corretta* e *completa*, e garantire la *terminazione*, restituendo una refutazione tutte le volte che l'ingresso è insoddisfacibile, e un modello tutte le volte che l'ingresso è soddisfacibile. Quando il problema è semidecidibile, ci si aspetta che la strategia di deduzione automatica sia una *procedura di semidecisione*. Per essere una procedura di semidecisione, una strategia deve essere *corretta* e *completa*, restituendo una refutazione tutte le volte che l'ingresso è insoddisfacibile. In pratica, tuttavia,

istanze di problemi decidibili possono essere troppo difficili per le risorse computazionali disponibili, oppure strategie complete possono essere troppo onerose, così che, indipendentemente dalla decidibilità, i sistemi di deduzione automatica possono rispondere o con una refutazione, o con un modello, o con la risposta "Non so". La misura in cui le risposte "Non so" possano essere tollerate dipende dall'applicazione.

Le applicazioni della deduzione automatica in informatica sono numerose. Per esempio, i *metodi formali* per la specifica, analisi, verifica, e sintesi di sistemi, sia hardware che software, si affidano a procedure di deduzione automatica per dimostrare congetture o trovare soluzioni di sistemi di vincoli. I legami tra la deduzione automatica e i metodi formali sono specialmente profondi, perché la *logica sta all'informatica come l'analisi matematica sta alla fisica*, e perciò si ritiene plausibile che il ragionare su programmi e sistemi possa essere più conducibile all'automazione di altri ambiti di ragionamento [39, 130, 17].

Come in altri campi dell'*intelligenza artificiale*, i problemi in deduzione automatica coinvolgono così tanta conoscenza, che è spesso troppo pesante o inefficiente rappresentarla tutta in H e φ, e quindi nell'insieme $\mathcal{S}$ di clausole. Pertanto, un paradigma comune è quello di ragionare *modulo* $\mathcal{T}$, dove $\mathcal{T}$ è una teoria definita da un insieme di assiomi. La *segnatura* di $\mathcal{T}$ contiene tutti i simboli di costante, funzione e predicato che appaiono negli assiomi. Dunque si restringe l'attenzione ai modelli di $\mathcal{T}$, detti $\mathcal{T}$-modelli, per determinare la $\mathcal{T}$*-soddisfacibilità*, e si cercano refutazioni modulo $\mathcal{T}$, per stabilire la $\mathcal{T}$-insoddisfacibilità. Per esempio, se $\mathcal{T}$ è la *teoria dell'uguaglianza*, abbiamo il *ragionamento equazionale*, dove si considerano solo modelli che soddisfino gli assiomi dell'uguaglianza, e gli assiomi dell'uguaglianza vengono incorporati nel sistema di inferenza. $\mathcal{T}$ può anche contenere altri assiomi che predicano proprietà di simboli diversi dall'uguaglianza, come per esempio l'*associatività* e la *commutatività* di simboli di funzione.

Se $\mathcal{T}$ è una teoria tale che la $\mathcal{T}$-soddisfacibilità è decidibile, il ragionamento modulo $\mathcal{T}$ è noto come *soddisfacibilità modulo teoria* (SMT), e gli assiomi di $\mathcal{T}$ sono incorporati nella procedura di decisione per la $\mathcal{T}$-soddisfacibilità. Per esempio, se $\mathcal{T}$ è il frammento senza quantificatori della teoria dell'uguaglianza, un algoritmo di *chiusura di congruenza* decide la $\mathcal{T}$-soddisfacibilità di un insieme di uguaglianze e ineguaglianze (si vedano [67, 119, 65] per le origini e il capitolo 9 di [39] per una descrizione più recente). Sebbene questa terminologia sia un po' arbitraria, un algoritmo che decide la $\mathcal{T}$-soddisfacibilità di un insieme di letterali senza variabili nella segnatura di $\mathcal{T}$, o $\mathcal{T}$-letterali senza variabili, si dice *procedura di* $\mathcal{T}$*-soddisfacibilità*. Un algoritmo che decide la $\mathcal{T}$-soddisfacibilità di una formula senza quantificatori nella segnatura di $\mathcal{T}$, o $\mathcal{T}$-formula senza quantificatori, si dice *procedura di* $\mathcal{T}$*-decisione*. Una formula

φ senza quantificatori è soddisfacibile se e solo se la sua chiusura esistenziale $\exists\bar{x}.\varphi$ è soddisfacibile, dove $\bar{x}$ contiene tutte le variabili che appaiono in φ. Poi, $\exists\bar{x}.\varphi$ è soddisfacibile se e solo se $\hat{\varphi}$ è soddisfacibile, dove $\hat{\varphi}$ è φ con tutte le variabili rimpiazzate da costanti di Skolem. Quindi, il problema di decidere la $\mathcal{T}$-soddisfacibilità di una $\mathcal{T}$-formula senza quantificatori è equivalente a quello di decidere la $\mathcal{T}$-soddisfacibilità di una $\mathcal{T}$-formula senza variabili.

Il resto di questo capitolo è organizzato come segue. La Sezione 2 descrive brevemente le principali classi di strategie di dimostrazione automatica di teoremi in logica del primo ordine, segnalando ricerche italiane e rinviando a precedenti lavori dell'autrice per rassegne ragionate dello stato dell'arte internazionale. La Sezione 3 è una rassegna ragionata dello stato dell'arte delle strategie guidate da conflitti. La Sezione 4 illustra due metodi guidati da conflitti di recente ideazione. Il primo si chiama *ragionamento Guidato da Semantica e Sensibile al Goal*, dall'inglese "Semantically-Guided Goal-Sensitive reasoning" (SGGS), ed è un metodo guidato da conflitti per la logica del primo ordine [36, 37, 38]. Il secondo si chiama *CDSAT*, dall'inglese "Conflict-Driven SATisfiability", ovvero *Soddisfacibilità Guidata da Conflitti*, ed è un metodo guidato da conflitti per la soddisfacibilità modulo una *generica* combinazione di teorie [26, 27]. Questo metodo porta anche alla definizione di una nuova classe di problemi detta *soddisfacibilità modulo assegnamenti* (SMA). Una versione breve di questo capitolo è stata presentata al sesto workshop internazionale su "Automated Formal Methods" nel maggio 2017 [18].

2 Metodi di dimostrazione automatica di teoremi e applicazioni

La maggior parte delle strategie di dimostrazione automatica di teoremi procede *refutazionalmente*, cioè tenta di stabilire che φ segue da H dimostrando che $\mathcal{S} = H \cup \{\neg\varphi\}$ è insoddisfacibile. Inoltre, i metodi di dimostrazione di teoremi lavorano con formule in *forma clausale*. La semplicità della forma clausale l'ha resa un formato macchina standard, anche se comporta una perdita di leggibilità per gli utenti umani. In logica del primo ordine, una volta che il problema è stato ridotto in forma clausale, è sufficiente considerare le *interpretazioni di Herbrand*, il cui dominio è l'*universo di Herbrand*, cioè l'insieme dei termini senza variabili [44]. Dunque per soddisfare un insieme di clausole $\mathcal{S}$ bisogna soddisfare tutte le sue clausole; per soddisfare una clausola C bisogna soddisfare tutte le sue istanze senza variabili, perché le variabili nelle clausole sono implicitamente quantificate universalmente; e per soddisfare una clausola senza variabili bisogna soddisfarne almeno un letterale. In logica del primo ordine la validità è semidecidibile, grazie al teorema di Herbrand: se $\mathcal{S}$ è insoddisfacibile, esiste un insieme *finito* e insoddisfacibile di istanze *senza*

variabili delle sue clausole. Quindi le strategie di dimostrazione automatica di teoremi in logica del primo ordine sono *procedure di semidecisione*.

Una strategia è definita da un *sistema di inferenza* e un *piano di ricerca*. Poichè si procede in modo refutazionale, la rilevante proprietà di completezza è la *completezza refutazionale*: un sistema di inferenza è refutazionalmente completo, se, ogniqualvolta l'insieme di ingresso $\mathcal{S}$ è insoddisfacibile, il sistema di inferenza è in grado di derivare una contraddizione da $\mathcal{S}$. Tuttavia un sistema di inferenza non è deterministico, nel senso che, dato $\mathcal{S}$, la derivazione non è unica. Per ottenere una strategia deterministica che generi un'unica derivazione da un dato problema in ingresso, occorre affiancare al sistema di inferenza un piano di ricerca. Il piano di ricerca deve essere *equo*, dall'inglese "fair", nel senso di non negligere passi di inferenza che potrebbero essere necessari per completare la refutazione. Un sistema di inferenza refutazionalmente completo e un piano di ricerca equo danno una strategia *completa*.

La maggior parte delle strategie di dimostrazione automatica di teoremi possono essere classificate in tre categorie: *strategie basate su ordinamenti*, *strategie basate su riduzione di goal a sottogoal*, e *strategie basate sulla generazione di istanze*. Questa classificazione appare in diverse rassegne dell'autrice a cui si rimanda per una presentazione più estesa e per la bibliografia, inclusi rimandi ad altre rassegne [14, 15, 34, 24, 19].

Le *strategie basate su ordinamenti* sono così chiamate perché usano *sistemi di inferenza basati su ordinamenti*. Questi sistemi lavorano su un insieme di clausole, detto anche *base di dati* delle clausole o *base di conoscenza* delle clausole, dato inizialmente dall'insieme $\mathcal{S}$. La loro caratteristica fondamentale, quella che dà loro il nome, è l'adozione di un ordinamento *ben fondato* $\succ$ su termini, letterali, e clausole. Il sistema di inferenza comprende regole di inferenza di *espansione* e regole di inferenza di *contrazione*. Le prime generano clausole e le aggiungono all'insieme, che viene così espanso. Le seconde eliminano clausole o le rimpiazzano con clausole più piccole, in modo che l'insieme di clausole viene contratto. La *correttezza* del sistema di inferenza richiede che le clausole generate e aggiunte per espansione siano conseguenze logiche di quelle preesistenti, e le clausole eliminate per contrazione siano conseguenze logiche di quelle restanti. La *risoluzione ordinata*, la *paramodulazione ordinata*, e la *sovrapposizione* sono le principali regole di inferenza di espansione, mentre la *sussunzione* e la *semplificazione* sono le principali regole di inferenza di contrazione [89, 126, 6, 28]. L'ordinamento ben fondato $\succ$ si usa per definire le regole di contrazione e per restringere l'applicabilità di quelle di espansione.

Le clausole che non vengono generate grazie alle restrizioni delle regole di espansione, e quelle che vengono eliminate o rimpiazzate dalle regole di contrazione sono *ridondanti*. La nozione di ridondanza dipende dall'ordinamento,

che può essere un ordinamento su clausole [6] o su prove [28, 20] (si veda [20] per una trattazione generale ed ulteriori riferimenti). Quando la clausola vuota viene generata, si è completata una dimostrazione per refutazione, che viene estratta dallo stato finale della derivazione mediante un processo detto *ricostruzione della dimostrazione* (si vedano [13] per l'introduzione di questo concetto e [19] per una trattazione più generale). Nelle strategie basate su ordinamenti il piano di ricerca è tipicamente l'*algoritmo della clausola data*, che realizza una *ricerca per qualità*, dall'inglese "best-first search", selezionando a ogni iterazione la clausola migliore secondo una funzione euristica (si vedano [112] per il riferimento più recente sul dimostratore OTTER che fu il primo ad avere questo algoritmo, e [128] per gli sviluppi più recenti). Le strategie basate sugli ordinamenti sono la scelta standard per il ragionamento in logica del primo ordine con uguaglianza e hanno trovato numerose applicazioni. Tra quelle sviluppate in Italia si segnalano per esempio in ambito matematico un approccio algebrico alla logica [74], e il ragionamento in teoria degli insiemi [75, 4], e in ambito verifica di programmi le procedure di decisione per teorie di strutture di dati [4, 2, 21, 22, 3] su cui torneremo nella Sezione 3.1.6.

Le clausole che fanno parte della forma clausale di $\neg\varphi$ e i loro letterali sono considerate clausole *goal* e letterali *goal* rispettivamente. Le *strategie basate su riduzione di goal a sottogoal* iniziano scegliendo in $\mathcal{S}$ una clausola goal e procedono applicando inferenze per ridurre il goal corrente a sottogoal. Questa classe di strategie include: strategie basate sulla *risoluzione lineare* (si vedano [125, 99] per le origini e [16] per la rappresentazione dello spazio di ricerca e ulteriori riferimenti); strategie basate sull'*eliminazione di modelli* (si vedano [107] per le origini e [16] per la rappresentazione dello spazio di ricerca e ulteriori riferimenti); e le strategie basate su *tableaux*, che sono divenute il paradigma principale per la riduzione di goal a sottogoal (si vedano [105, 10] per l'avvio della formalizzazione dell'eliminazione di modelli come strategia basata su tableaux, [12, 11, 103, 106] per rassegne, e [16] per la rappresentazione dello spazio di ricerca e ulteriori riferimenti).

Un *tableau* è una struttura a forma di albero, dove i nodi sono etichettati da letterali e i rami rappresentano possibili modelli. Le regole di inferenza estendono un ramo oppure lo chiudono perché contiene due letterali complementari unificabili. Questa è la nozione più semplice di chiusura, ma esistono anche nozioni più elaborate. Quando tutti i rami del tableau sono chiusi, si è completata una dimostrazione per refutazione, e non è necessario ricostruirla, perché è data dal tableau chiuso. In questo senso si può dire che mentre una strategia basata su ordinamenti costruisce *molti* tentativi di dimostrazione contenuti *implicitamente* nell'insieme di clausole generate e tenute, una strategia basata su riduzione di goal a sottogoal costruisce *esplicitamente un* tentativo

di dimostrazione per volta [14].

Per quanto riguarda il piano di ricerca, le strategie basate su riduzione di goal a sottogoal svolgono tipicamente una *ricerca per profondità*, dall'inglese "depth-first search", con *ritorno all'indietro*, dall'inglese "backtracking", e approfondimento iterativo, dall'inglese "iterative deepening" [96, 134]. Il ritorno all'indietro è necessario quando la derivazione non può essere avanzata da alcuna inferenza e quindi è necessario disfare almeno un passo già fatto. Con il ritorno all'indietro una strategia basata sulla riduzione di goal a sottogoal abbandona un tentativo di dimostrazione e passa ad un altro. L'approfondimento iterativo è necessario per avere un piano di ricerca equo, perché la ricerca per profondità non è equa in presenza di uno spazio di ricerca infinito quale quello di un problema di dimostrazione di teoremi in logica del primo ordine. La ricerca per profondità, il ritorno all'indietro e l'approfondimento iterativo sono usati anche da strategie che non appartengono alla classe delle strategie basate sulla riduzione di goal a sottogoal e appariranno anche più avanti in questo capitolo.

Nelle strategie basate su riduzione di goal a sottogoal la ridondanza compare sotto forma di ripetizione di sottogoal. I rimedi includono diverse tecniche quali *C-riduzione* [131], *"caching"* [5, 29], *fusione regressiva*, dall'inglese "regressive merging" [136], *piega all'insù*, dall'inglese "folding-up" [104, 76], *sostituzioni di successo*, dall'inglese "success substitutions" [106], e *tavolatura*, dall'inglese "tabling", o *memorizzazione*, dall'inglese abbreviato "memoing" [138]. C-riduzione, caching, e fusione regressiva si usano in eliminazione di modelli; piega all'insù e sostituzioni di successo si usano nei tableaux; tavolatura e memorizzazione si usano in programmazione dichiarativa, dove hanno trovato numerose applicazioni, dalla pianificazione [144] alla soluzione di giochi [145].

Dal punto di vista logico, tutte queste tecniche discendono dall'idea della *generazione di lemmi* o *lemmatizzazione* (si vedano [107, 5] per le origini, [29] per una trattazione generale della lemmatizzazione come inferenza di meta-livello e ulteriori riferimenti, [16] per un'analisi dell'impatto della lemmatizzazione sullo spazio di ricerca e ulteriori riferimenti). In breve, l'idea della lemmatizzazione è che se la strategia riesce a chiudere un sottotableau la cui radice è etichettata con il letterale L, significa che nessun modello di $\mathcal{S}$ contiene L, cioè $S \models \neg L$, e dunque la strategia può imparare il lemma $\neg L$ e usarlo per chiudere altri rami del tableau. Questa descrizione semplificata si applica solo al caso in cui sia stato possibile chiudere il sottotableau con radice L senza usare letterali che etichettano nodi sul cammino da L alla radice del tableau. Altrimenti, un lemma unitario non è sufficiente (si veda per esempio la Sezione 2.5 di [16] per una trattazione più completa). Le strategie basate

su tableaux sono assai versatili e quindi sono state sviluppate e implementate anche per molte logiche diverse dalla logica classica del primo ordine. Nel panorama italiano si segnalano per esempio metodi per logiche *intuizionistiche* [70, 71, 72], *modali* [79, 73], *condizionali* [121, 81], *descrittive* [43, 80], e *temporali* [82, 86, 41].

Le *strategie basate sulla generazione di istanze* intendono implementare il teorema di Herbrand nel modo più diretto possibile. L'idea è di generare insiemi di istanze senza variabili delle clausole in $\mathcal{S}$ e invocare una procedura di decisione per la soddisfacibilità proposizionale per determinarne la soddisfacibilità (si vedano [101] per l'inizio della rinascita di questo tipo di approccio e [15, 24, 19]per ulteriori riferimenti). Se un insieme di istanze senza variabili risulta insoddisfacibile, anche $\mathcal{S}$ è insoddisfacibile, e l'insieme di istanze senza variabili rappresenta la dimostrazione di insoddisfacibilità. Se un insieme di istanze senza variabili risulta soddisfacibile, occorre generarne un altro. Strategie più avanzate invocano una procedura di decisione per la soddisfacibilità modulo teorie per determinare la soddisfacibilità degli insiemi di istanze.

Accanto alla classificazione in strategie basate su ordinamenti, strategie basate sulla riduzione di goal a sottogoal, e strategie basate sulla generazione di istanze, esistono altri criteri per classificare i metodi di deduzione automatica. Un criterio classico in intelligenza artificiale distingue tra strategie che ragionano primariamente *in avanti*, dall'inglese "forward", e strategie che ragionano primariamente *all'indietro*, dall'inglese "backward". Ragionare in avanti significa dedurre primariamente dalle assunzioni, mentre ragionare all'indietro significa dedurre primariamente dalle clausole goal. Le strategie basate sulla riduzione di goal a sottogoal ragionano primariamente all'indietro, mentre le strategie basate su ordinamenti e su generazione di istanze generalmente non distinguono le clausole goal dalle altre clausole e quindi ragionano primariamente in avanti.

Un criterio che ebbe particolare fortuna nel confronto tra la risoluzione e i tableaux considera se la strategia sia *confluente rispetto alla dimostrazione*, dall'inglese "proof confluent". Una strategia di dimostrazione di teoremi si dice *confluente rispetto alla dimostrazione*, se garantisce che si possa completare la dimostrazione senza mai dover disfare un passo di inferenza. In altre parole, una strategia è confluente rispetto alla dimostrazione, se può andare sempre avanti senza aver bisogno di tornare indietro ("backtracking"). Le strategie basate su ordinamenti sono confluenti in questo senso, mentre le strategie basate su riduzione di goal a sottogoal e le strategie basate su generazione di istanze generalmente non lo sono, stante la necessità di disfare qualcosa per passare al tentativo di dimostrazione successivo o al successivo insieme di istanze.

Un criterio di più recente fortuna considera se la strategia sia *basata su*

modelli [24, 19].Una strategia di deduzione automatica si dice *basata su modelli*, se lo stato della derivazione contiene una rappresentazione di un *modello parziale candidato*, e inferenze e ricerca di un modello sono interallacciate, nel senso che le inferenze costruiscono e trasformano il modello, mentre il modello guida le inferenze (si veda [24] per una rassegna di queste strategie in logica del primo ordine). Si parla di *modello candidato*, perché finché non si è stabilita la soddisfacibilità non si sa se sarà davvero un modello. Similmente si parla di *modello parziale*, perché finché non si è stabilita la soddisfacibilità non si sa se si riuscirà a completarlo in un modello totale o modello tout court.

Le strategie basate su ordinamenti non sono basate su modelli, perché non tentano di costruire un modello. Tra le strategie basate su riduzione di goal a sottogoal, l'eliminazione di modelli e le strategie basate su tableaux si possono considerare basate su modelli, in quanto ogni ramo di un tableau rappresenta un modello parziale candidato. Dopo gli inizi, le strategie basate su generazione di istanze sono evolute verso uno schema *guidato da modelli*: quando un insieme di istanze senza variabili risulta soddisfacibile, la procedura di decisione di soddisfacibilità ne produce un modello, e la successiva fase di generazione di istanze mira a generare istanze false in tale modello. Il processo continua fino alla generazione di un insieme di istanze insoddisfacibile. La recente crescita del paradigma della *programmazione con insieme di risposte* [102, 53, 66, 46, 47, 83, 48, 84, 49, 85], dall'inglese "answer set programming", rappresenta per la programmazione dichiarativa una traiettoria analoga a quella verso le strategie basate su modelli in dimostrazione automatica di teoremi.

Le strategie di deduzione automatica *guidate da conflitti* sono una sottoclasse delle strategie basate su modelli. In una strategia basata su modelli, si ha un *conflitto* ogni qualvolta una delle clausole di $\mathcal{S}$ è falsa nel modello candidato corrente. Una strategia si dice *guidata da conflitti*, se usa le inferenze per *spiegare* e *risolvere* il conflitto riparando il modello. Nel resto di questo capitolo consideriamo strategie di deduzione automatica *guidate da conflitti*, iniziando con quelle che sono procedure di decisione per la soddisfacibilità in logica proposizionale o in frammenti decidibili della logica del primo ordine. Il ragionamento in logica del primo ordine ricomparirà nella Sezione 3.1.6 e nella Sezione 4.1. In quest'ultima verrà presentato SGGS, un metodo per la dimostrazione di teoremi in logica del primo ordine, che usa la generazione di istanze, non richiede che siano istanze senza variabili, è confluente rispetto alla dimostrazione, è basato su modelli proprio nel senso di costruire la rappresentazione di modelli parziali candidati in logica del primo ordine, ed è guidato da conflitti [36, 37, 38].

3 Metodi guidati da conflitti

Il ragionamento guidato da conflitti appare per la prima volta nella procedura di *Apprendimento di Clausole Guidato da Conflitti*, dall'inglese "Conflict-Driven Clause Learning" (CDCL), per la *soddisfacibilità proposizionale* [110, 116, 109]. Il problema della *soddisfacibilità proposizionale*, noto in informatica come problema SAT, dall'inglese "SATisfiability", è il problema di decidere la soddisfacibilità di un insieme $\mathcal{S}$ di clausole in logica proposizionale. La procedura CDCL è una componente centrale dei sistemi per SAT, detti *SAT-solutori*, dall'inglese "SAT-solver". Nella Sezione 3.1 trattiamo CDCL e i metodi che incorporano a scatola chiusa un SAT-solutore basato su CDCL. In strategie siffatte la parte di deduzione guidata da conflitti è proposizionale, anche se il metodo si applica a una logica più generale. Nella Sezione 3.2 trattiamo i metodi guidati da conflitti che generalizzano il principio della deduzione guidata da conflitti alla soddisfacibilità modulo teoria.

3.1 Deduzione guidata da conflitti proposizionale

Questa sezione è dedicata a metodi la cui componente guidata da conflitti è ristretta alla logica proposizionale. In altre parole, il modello candidato e le inferenze guidate da conflitto sono proposizionali. Cominciamo tuttavia con una procedura che *non* è guidata da conflitti, ma è *basata su modelli*, per illustrare prima questa nozione più basilare.

3.1.1 DPLL: deduzione proposizionale basata su modelli

La procedura DPLL (Davis-Putnam-Logemann-Loveland) per SAT [55, 54, 44, 143] rappresenta un modello parziale candidato mediante una *sequenza* di letterali, detta *traccia*, e denotata con $\mathcal{M}$. La traccia rappresenta il modello parziale, anch'esso chiamato $\mathcal{M}$, dove tutti i letterali sulla traccia sono veri. Se un letterale L appare sulla traccia $\mathcal{M}$, L è vero nel modello $\mathcal{M}$ e il suo complementare $\neg L$ è falso nel modello $\mathcal{M}$. Se nè L nè $\neg L$ appaiono in $\mathcal{M}$, entrambi i letterali sono *indefiniti*.

La procedura DPLL inizia mettendo in $\mathcal{M}$ tutti letterali che compaiono nell'ingresso come clausole unitarie, se ve ne sono. Una *clausola unitaria* è una clausola formata da un solo letterale. Chiaramente non c'è modo di soddisfare una clausola unitaria se non mettendo nel modello il suo unico letterale. La procedura continua *propagando* le conseguenze del contenuto della traccia sotto forma di *implicazioni* e *conflitti*, un'attività nota come *propagazione clausale*

Booleana, e abbreviata BCP dall'inglese "Boolean Clausal Propagation". Per le implicazioni, si supponga che tutti i letterali di una clausola $C \in \mathcal{S}$ eccetto uno, poniamo L, siano falsi in $\mathcal{M}$. Allora il letterale L è un *letterale implicato*, e viene aggiunto a $\mathcal{M}$ con C come *giustificazione*. Infatti, estendere $\mathcal{M}$ con L è il solo modo per soddisfare C.

Esempio 3.1: Supponiamo che $\mathcal{S}$ contenga le clausole $\{a,\ b,\ \neg a \vee \neg b \vee c\}$. Si inizia mettendo a e b in $\mathcal{M}$: $\mathcal{M} = (\ a,\ b\ )$. A questo punto c è un letterale implicato con giustificazione $\neg a \vee \neg b \vee c$, e $\mathcal{M}$ diventa $\mathcal{M} = (\ a,\ b,\ c\ )$.

La scoperta di un letterale implicato si può vedere in termini di inferenze come il risultato di una serie di passi di *risoluzione unitaria* con i letterali in $\mathcal{M}$ come premesse unitarie. La *risoluzione unitaria* è la risoluzione dove almeno una delle due premesse è una clausola unitaria. Equivalentemente, si può vedere come un singolo passo di *risoluzione a risultato unitario*, dall'inglese "Unit-Resulting resolution" [111], avente come premesse la giustificazione e i letterali in $\mathcal{M}$ e come risolvente il letterale implicato. La *risoluzione a risultato unitario* è una forma di *iperrisoluzione* [124, 44] che risolve simultaneamente una clausola nucleo non unitaria con una o più clausole satelliti unitarie a generare una clausola unitaria o vuota. Per i conflitti, ogniqualvolta tutti i letterali di una clausola $C \in \mathcal{S}$ sono falsi in $\mathcal{M}$, si ha un *conflitto*: C è una *clausola in conflitto* (con $\mathcal{M}$).

Esempio 3.2: Supponiamo che $\mathcal{S}$ contenga le clausole $\{\neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b\}$, e la traccia $\mathcal{M}$ sia la sequenza $(a,\ b,\ c,\ d,\ e,\ \neg f)$: la clausola $\neg b \vee \neg e \vee f$ è in conflitto.

La scoperta di un conflitto si può vedere in termini di inferenze come il risultato di una sequenza di passi di risoluzione unitaria, o come un singolo passo di risoluzione a risultato unitario, che produce la clausola vuota $\square$. Quando tutte le propagazioni possibili sono state fatte e in assenza di un conflitto, la procedura DPLL *decide* che un letterale L sia vero aggiungendolo a $\mathcal{M}$. Un letterale aggiunto alla traccia da una decisione si chiama *letterale deciso*. Una decisione è semplicemente una mossa per avanzare la ricerca di un modello provando a indovinare che ve ne sia uno dove quel letterale è vero. Questa operazione è anche nota come *analisi dei casi* o *suddivisione*, dall'inglese "splitting", perché per un letterale L ci sono due casi, o è vero o è falso.

Esempio 3.3: Riprendiamo l'esempio 3.2 e vediamo come si può essere arrivati al conflitto tra la clausola $\neg b \vee \neg e \vee f$ e la traccia $\mathcal{M} = (\ a,\ b,\ c,\ d,\ e,\ \neg f\ )$. La procedura decide che a sia vero. Segue che b è un letterale implicato con

giustificazione $\neg a \vee b$. La procedura decide che c sia vero. Segue che d è un letterale implicato con giustificazione $\neg c \vee d$. La procedura decide che e sia vero. Segue che $\neg f$ è un letterale implicato con giustificazione $\neg e \vee \neg f$. A questo punto la traccia $\mathcal{M}$ è la sequenza (a, b, c, d, e, $\neg f$) e la clausola $\neg b \vee \neg e \vee f$ è in conflitto.

Dopo ogni decisione la procedura applica la BCP per scoprire nuove implicazioni o un conflitto. Quando si verifica un conflitto, la procedura DPLL ritorna indietro cronologicamente, disfacendo la decisione più recente e tutte le propagazioni che ne conseguono. La procedura restituisce "soddisfacibile" se $\mathcal{M} \models \mathcal{S}$, e "insoddisfacibile" se c'è un conflitto e nessuna decisione da disfare.

3.1.2 CDCL: deduzione proposizionale guidata da conflitti

La procedura di *Apprendimento di Clausole Guidato da Conflitti* (CDCL) [110, 116, 109] eredita dalla procedura DPLL la rappresentazione del modello candidato, la BCP, e le decisioni. Mantiene anche l'inizializzazione della traccia con le clausole unitarie in ingresso, di cui si dice che sono memorizzate al *livello* 0 della traccia. Ogni decisione apre un successivo *livello di decisione* sulla traccia: il livello di decisione n-esimo contiene l'n-esimo letterale deciso sulla traccia corrente e tutti i letterali implicati scoperti dalla BCP come conseguenza.

Esempio 3.4: Nell'esempio 3.3 la traccia $\mathcal{M}$ = (a, b, c, d, e, $\neg f$) vede la sottosequenza (a, b) al primo livello, con a deciso e b implicato; la sottosequenza (c, d) al secondo livello, con c deciso e d implicato; e la sottosequenza (e, $\neg f$) al terzo livello, con e deciso e $\neg f$ implicato.

La procedura CDCL si comporta però in modo marcatamente differente da DPLL quando si manifesta un conflitto. Supponiamo che C sia una clausola in conflitto e contenga un letterale L, tale che $\neg L$ appare in $\mathcal{M}$ con giustificazione D. La procedura CDCL applica la risoluzione a C e D risolvendo i letterali L e $\neg L$. Questa inferenza *spiega* il conflitto: L è falso perché $\neg L$ è vero, e $\neg L$ è vero, perché c'è D in $\mathcal{S}$ e tutti gli altri letterali di D appaiono negati in $\mathcal{M}$. Il risolvente così generato è ancora in conflitto, in quanto tutti gli altri letterali di C e D sono falsi in $\mathcal{M}$. Ogni risolvente è una conseguenza logica di $\mathcal{S}$ e può essere aggiunto a $\mathcal{S}$ come una *clausola appresa* o *lemma*. Tale passo si chiama *apprendimento*. In pratica, $\mathcal{S}$ contiene spesso un numero enorme di clausole, e la procedura impara al più una clausola per conflitto. Quante risoluzioni svolgere e quale risolvente imparare è materia di scelta euristica.

Esempio 3.5: Continuando l'esempio 3.3, perché la clausola $\neg b \vee \neg e \vee f$ è in conflitto con $\mathcal{M} = (\ a,\ b,\ c,\ d,\ e,\ \neg f\ )$? Perché tutti i suoi letterali sono falsi in $\mathcal{M}$. Perché f è falso in $\mathcal{M}$? Perché $\neg f$ è vero in $\mathcal{M}$. Perché $\neg f$ è vero in $\mathcal{M}$? Per soddisfare la sua giustificazione $\neg e \vee \neg f$. Allora si spiega il conflitto risolvendo la clausola in conflitto $\neg b \vee \neg e \vee f$ con la giustificazione $\neg e \vee \neg f$, ottenendo il risolvente $\neg b \vee \neg e$. Si noti che $\neg b \vee \neg e$ è ancora in conflitto. Imparare il lemma $\neg b \vee \neg e$ significa imparare che il dato insieme di clausole non ha un modello dove b ed e siano entrambe vere.

L'euristica nota come *primo punto di implicazione*, dall'inglese "first unique implication point" (1UIP), prescrive di eseguire passi di risoluzione finché si genera un risolvente che è una *clausola di asserzione*. Sia l'n-esimo il livello di decisione corrente. Una clausola in conflitto $C = L_1 \vee \ldots \vee L_k \vee \ldots \vee L_m$ è una *clausola di asserzione*, se per uno solo dei suoi letterali, poniamo L_k, il complementare appare sulla traccia $\mathcal{M}$ all'n-esimo livello di decisione. Per tutti gli altri letterali L_j di C, con $1 \leq j \leq k-1$ o $k+1 \leq j \leq m$, il complementare appare su $\mathcal{M}$ in un livello di decisione di numero più piccolo di n. Al limite, $\neg L_j$ appare al livello 0, se $\neg L_j$ è una clausola unitaria nell'insieme di ingresso $\mathcal{S}$. L'euristica 1UIP prescrive che la procedura *impari* la prima clausola di asserzione generata dal processo di spiegazione. Supponiamo che C sia la prima. Dopo averla imparata, cioè aggiunta a $\mathcal{S}$, la procedura *salta all'indietro*, dall'inglese "backjump", al più piccolo livello di decisione q dove L_k è indefinito, e tutti gli altri letterali di C sono falsi.

Esempio 3.6: Continuando l'esempio 3.5, la clausola $\neg b \vee \neg e$ è una clausola di asserzione relativamente a $\mathcal{M} = (\ a,\ b,\ c,\ d,\ e,\ \neg f\ )$, perché solo $\neg e$ è falso nel livello di decisione corrente. Il livello di decisione corrente è il terzo e contiene la sottosequenza $(e,\ \neg f)$ (cfr. l'esempio 3.4). Dunque l'euristica 1UIP prescrive di imparare $\neg b \vee \neg e$ e saltare al primo livello, portando la traccia allo stato $\mathcal{M} = (\ a,\ b,\ \neg e\ )$, essendo il primo il più piccolo livello dove $\neg e$ è indefinito e $\neg b$ è falso. Per confronto, se avesse applicato il "backtracking" cronologico, la procedura sarebbe tornata a $\mathcal{M} = (\ a,\ b,\ c,\ d\ )$. L'intuizione di CDCL è invece quella di disfare anche la sottosequenza $(c,\ d)$, che non ha avuto parte nel conflitto, e asserire $\neg e$ per soddisfare $\neg b \vee \neg e$ ed evitare di violare di nuovo questo lemma.

Esiste per certo un livello di decisione dove L_k è indefinito e tutti gli altri letterali di C sono falsi: infatti, L_k è indefinito in ogni livello più piccolo di n; e siccome C è una clausola in conflitto, per tutti i suoi letterali il complementare è sulla traccia a un qualche livello. Dunque si tratta semplicemente di prendere come q il più piccolo tra i livelli con queste proprietà. Se q è pari a $n-1$, il

salto all'indietro si riduce al tornare indietro ("backtracking") cronologico. Se q è pari a 0, la procedura torna in uno stato dove solo le clausole unitarie di ingresso sono sulla traccia. In ogni caso almeno una decisione viene sicuramente disfatta. Dopo il salto all'indietro, la procedura aggiunge L_k a $\mathcal{M}$, così che C è soddisfatta, il conflitto risolto, e si può riprendere la ricerca con propagazioni e decisioni.

La procedura CDCL restituisce "soddisfacibile" se $\mathcal{M} \models \mathcal{S}$, e "insoddisfacibile" se c'è un conflitto al livello 0. Operativamente, se c'è un conflitto al livello 0 non esiste un livello più piccolo a cui saltare per risolverlo. Logicamente, un conflitto al livello 0 significa che si sono generate due clausole unitarie complementari come L e $\neg L$, e quindi la clausola vuota. La capacità di spiegare i conflitti e imparare clausole permette alla procedura CDCL di *generare dimostrazioni*, formate precisamente dai passi di spiegazione. Poiché i passi di spiegazione sono inferenze di risoluzione, le dimostrazioni prodotte da CDCL sono dimostrazioni per risoluzione, che si possono rappresentare come *alberi di risoluzione* [44]. Nell'esempio seguente [30] usiamo la notazione $\Pi(C)$ per indicare un albero di risoluzione che genera la clausola C.

Esempio 3.7: Sia $\mathcal{S}$ l'insieme $\{p \vee q, \neg p \vee q, \neg q \vee p, \neg p \vee \neg q\}$. Supponiamo che la derivazione inizi con la decisione che p sia vero: $\mathcal{M} = (\ p\ )$. Segue che q è un letterale implicato con giustificazione $\neg p \vee q$: $\mathcal{M} = (\ p,\ q\ )$. A questo punto $\neg p \vee \neg q$ è in conflitto. Si spiega il conflitto risolvendo $\neg p \vee \neg q$ con la giustificazione $\neg p \vee q$. Il risolvente $\neg p$ è una clausola di asserzione. Si noti che una clausola in conflitto unitaria è banalmente una clausola di asserzione, e una clausola di asserzione unitaria porta inevitabilmente a tornare al livello 0. Quindi si impara $\neg p$ aggiungendolo a $\mathcal{S}$; si salta all'indietro al livello 0; e si mette $\neg p$ sulla traccia: $\mathcal{M} = (\ \neg p\ )$. Inoltre, poiché $\neg p$ è una clausola generata e non di ingresso, si memorizza che la sua dimostrazione $\Pi(\neg p)$ è data dal passo di risoluzione tra $\neg p \vee \neg q$ e $\neg p \vee q$. La derivazione ricomincia con la propagazione del letterale implicato q con giustificazione $p \vee q$: $\mathcal{M} = (\ \neg p,\ q\ )$. Ora $\neg q \vee p$ è in conflitto. Si spiega il conflitto risolvendo $\neg q \vee p$ con la giustificazione $p \vee q$. Si impara il risolvente p aggiungendolo a $\mathcal{S}$ e memorizzando che la sua dimostrazione $\Pi(p)$ è data dal passo di risoluzione tra $\neg q \vee p$ e $p \vee q$. Si noti che $\mathcal{M}$ contiene solo il livello 0 e quindi non si applica l'euristica 1UIP a p e non si salta all'indietro, perché non c'è un livello più piccolo di 0 a cui saltare. Inoltre, non si aggiunge p a $\mathcal{M}$, perché vorrebbe dire rendere il modello candidato inconsistente. L'unica mossa possibile è spiegare ulteriormente il conflitto risolvendo p con la giustificazione $\neg p$ a ottenere la clausola vuota $\square$. La procedura restituisce "insoddisfacibile" con la refutazione $\Pi(\square)$ composta dal passo di risoluzione tra p e $\neg p$ con $\Pi(p)$ and $\Pi(\neg p)$ come sottoalberi.

Si noti che la procedura DPLL (cfr. la Sezione 3.1.1) non genera refutazioni. Al contrario CDCL genera refutazioni proprio grazie al suo essere una procedura guidata da conflitti, con spiegazione dei conflitti mediante risoluzione, apprendimento di risolventi, e salto all'indietro guidato dalla clausola in conflitto. Per chiudere con una nota storica, la procedura di Davis-Putnam (DP) [55] includeva la risoluzione proposizionale, poi sostituita dall'analisi dei casi nella procedura DPLL [54]. In effetti, ci sono ottime ragioni per non adottare la risoluzione come principio d'inferenza in logica proposizionale: la risoluzione duplica i letterali (si veda per esempio l'analisi dell'impatto di questo comportamento in [123]) ed è inutilmente non deterministica per una logica semplice come la logica proposizionale. Un pregio della procedura CDCL dal punto di vista della deduzione consiste proprio nel reintrodurre la risoluzione nel ragionamento proposizionale in modo mirato, solo quel tanto che basta per spiegare i conflitti.

3.1.3 Procedure per SMT che incorporano CDCL: DPLL($\mathcal{T}$)

La soddisfacibilità modulo teoria (SMT) è il problema di decidere la $\mathcal{T}$-soddisfacibilità di un insieme $\mathcal{S}$ di clausole *senza variabili*. Il paradigma DPLL($\mathcal{T}$) per SMT [120, 129, 9] integra una procedura di $\mathcal{T}$-soddisfacibilità in un SAT-solutore basato sulla procedura CDCL. Per analogia con SAT-solutore, il programma che implementa una procedura di $\mathcal{T}$-soddisfacibilità si chiama *$\mathcal{T}$-solutore*. Dal momento che il SAT-solutore accetta solo clausole proposizionali, gli atomi dei $\mathcal{T}$-letterali vengono sostituiti da atomi proposizionali mediante l'applicazione di una *funzione di astrazione* α.

Esempio 3.8: Si consideri l'insieme di clausole $\mathcal{S} = \{x \geq 2,\ \neg(x \geq 1) \vee y \geq 1,\ x^2 + y^2 \leq 1 \vee xy > 1\}$, dove x e y sono variabili libere, ovvero variabili implicitamente quantificate in modo esistenziale in aritmetica non lineare. Dal punto di vista logico, come già detto, x e y possono essere sostituite da simboli di costante o trattate come simboli di costante, ma per chi legge è sovente più intuitivo chiamarle variabili. Determinare la soddisfacibilità di $\mathcal{S}$ significa trovare valori di x e y che soddisfino le tre clausole di $\mathcal{S}$. L'astrazione proposizionale di $\mathcal{S}$ è $\alpha(S) = \{a,\ \neg b \vee c,\ d \vee e\}$, ovvero $\alpha(x \geq 2) = a$, $\alpha(x \geq 1) = b$, e così via. Chiaramente tutta l'informazione viene persa eccetto la struttura proposizionale data dai connettivi. La funzione inversa α^{-1} svolge la traduzione nel verso opposto: $\alpha^{-1}(a) = x \geq 2$, $\alpha^{-1}(b) = x \geq 1$, e così via.

Ovviamente non è possibile risolvere il problema ragionando solo in modo proposizionale. Occorre far collaborare la ricerca di un modello proposizio-

nale svolta dalla procedura CDCL con la conoscenza della teoria specifica $\mathcal{T}$ incorporata nella procedura di $\mathcal{T}$-soddisfacibilità.

Esempio 3.9: Continuando l'esempio 3.8, la procedura CDCL inizia mettendo a sulla traccia: $\mathcal{M} = (\ a\ )$. La procedura proposizionale non sa che a sta per $x \geq 2$. Per la procedura proposizionale a è una clausola unitaria che deve essere soddisfatta. Il $\mathcal{T}$-solutore legge che a è sulla traccia, applica α^{-1}, e quindi interpreta che $x \geq 2$ è vero nel modello corrente. Poiché il problema contiene anche l'atomo $x \geq 1$, il $\mathcal{T}$-solutore determina che $x \geq 2$ implica $x \geq 1$, interpretando il predicato $\geq$ in aritmetica. Quindi il $\mathcal{T}$-solutore segnala al SAT-solutore che $\alpha(x \geq 1) = b$ è vero. Il SAT-solutore mette b sulla traccia: $\mathcal{M} = (\ a,\ b\ )$. A questo punto il SAT-solutore considera la clausola $\neg b \vee c$ e aggiunge c alla traccia come letterale implicato con giustificazione $\neg b \vee c$, da cui $\mathcal{M} = (\ a,\ b,\ c\ )$.

Come illustrato nell'esempio, il SAT-solutore non sa nulla dei predicati interpretati nella teoria $\mathcal{T}$ (e.g., $\geq$ in aritmetica), e simmetricamente il $\mathcal{T}$-solutore non conosce la disgiunzione e la sua interazione con la negazione. Le procedure SMT organizzano la collaborazione tra questi diversi solutori in modo che possano risolvere il problema lavorando ciascuno con il suo linguaggio. In DPLL($\mathcal{T}$), l'interfaccia tra SAT-solutore e $\mathcal{T}$-solutore consiste essenzialmente di due regole di inferenza. La regola di *$\mathcal{T}$-conflitto* permette al $\mathcal{T}$-solutore di segnalare al SAT-solutore che un insieme di letterali $L_1, \ldots, L_k$ in $\mathcal{M}$ è inconsistente in $\mathcal{T}$. Il $\mathcal{T}$-conflitto viene gestito esattamente come un conflitto puramente proposizionale, usando $\neg L_1 \vee \ldots \vee \neg L_k$ come *clausola in $\mathcal{T}$-conflitto*. Una clausola in $\mathcal{T}$-conflitto è una conseguenza logica del problema nella teoria $\mathcal{T}$, e quindi può essere imparata come un *$\mathcal{T}$-lemma*, esattamente come si può imparare un risolvente in logica proposizionale. La regola di *$\mathcal{T}$-propagazione* permette al $\mathcal{T}$-solutore di segnalare al SAT-solutore che nella teoria $\mathcal{T}$ un insieme di letterali $L_1, \ldots, L_k$ in $\mathcal{M}$ implica un letterale L. Il *$\mathcal{T}$-lemma* $\neg L_1 \vee \ldots \vee \neg L_k \vee L$ viene imparato, e il letterale L viene aggiunto a $\mathcal{M}$ come un letterale *$\mathcal{T}$-implicato*, avente il $\mathcal{T}$-lemma $\neg L_1 \vee \ldots \vee \neg L_k \vee L$ come giustificazione.

Esempio 3.10: Nell'esempio 3.9 il letterale b è un letterale $\mathcal{T}$-implicato con giustificazione $\neg a \vee b$. Il $\mathcal{T}$-solutore vede che $\{x \geq 2\} \models_{\mathcal{T}} x \geq 1$, dove $\models_{\mathcal{T}}$ è il simbolo di conseguenza logica nella teoria $\mathcal{T}$, e quindi costruisce il $\mathcal{T}$-lemma $\neg(x \geq 2) \vee x \geq 1$. Il SAT-solutore riceve questo $\mathcal{T}$-lemma nella forma $\neg a \vee b$, e mette b sulla traccia $\mathcal{M}$ con $\neg a \vee b$ come sua giustificazione.

DPLL($\mathcal{T}$) *non permette la creazione di nuovi atomi*, cioè atomi che non compaiano già in $\mathcal{S}$ o nella sua astrazione $\alpha(S)$. La regola di $\mathcal{T}$-propagazione

impone che l'atomo del letterale $\mathcal{T}$-implicato L appaia già nell'insieme esistente di clausole. Le clausole imparate durante il processo di spiegazione di un conflitto o $\mathcal{T}$-conflitto sono risolventi proposizionali formati da letterali i cui atomi provengono da $\mathcal{S}$ o $\alpha(S)$. Quindi non ci sono meccanismi che creino nuovi atomi durante la derivazione. Questa assunzione che non vengano formati nuovi atomi gioca un ruolo fondamentale nella dimostrazione di terminazione di DPLL($\mathcal{T}$) [120]. Dal punto di vista della nostra trattazione, l'aspetto cruciale è che il ragionamento guidato da conflitti in DPLL($\mathcal{T}$) rimane puramente proposizionale, svolto dal SAT-solutore secondo la procedura CDCL, sia per conflitti che per $\mathcal{T}$-conflitti, senza che la teoria $\mathcal{T}$ e la procedura di $\mathcal{T}$-soddisfacibilità abbiano alcun ruolo nello spiegare e risolvere i $\mathcal{T}$-conflitti.

3.1.4 Da una teoria a molte: la condivisione di uguaglianze

Nei problemi prodotti dagli ambiti di applicazione, $\mathcal{T}$ è tipicamente una combinazione di teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$, assiomatizzata dall'unione delle assiomatizzazioni di $\mathcal{T}_1, \ldots, \mathcal{T}_n$. Il metodo di *condivisione di uguaglianze*, dall'inglese "equality sharing" è l'approccio standard per questo problema (si vedano [118, 117] per le origini e il capitolo 10 di [39] per una descrizione più recente). Si suppone che siano note n procedure di $\mathcal{T}_i$-soddisfacibilità, per $i = 1, \ldots n$, implementate in altrettanti $\mathcal{T}_i$-solutori, per $i = 1, \ldots n$, e l'obbiettivo è combinarli in una procedura di $\mathcal{T}$-soddisfacibilità.

Per poter decidere dell'esistenza di un $\mathcal{T}$-modello, le procedure di $\mathcal{T}_i$-soddisfacibilità devono poter concordare l'interpretazione di tutto ciò che è condiviso dalle teorie. Il metodo di condivisione di uguaglianze assume che le teorie siano *disgiunte*, che significa che non hanno simboli di funzione o di predicato in comune, eccetto l'uguaglianza, che si assume presente in tutte le teorie. Teorie disgiunte possono condividere tipi, e simboli di costante, purché siano simboli liberi, cioè non compaiano in assiomi. Come già osservato, per problemi di soddisfacibilità si possono usare indifferentemente variabili libere o costanti libere. Tuttavia, i termini in $\mathcal{S}$ naturalmente mescolano simboli dalle diverse teorie. Quindi, il primo passo è *separare* le occorrenze di simboli da teorie diverse, introducendo *nuove costanti libere*.

Esempio 3.11: Supponiamo che l'insieme $\mathcal{S}$ di ingresso contenga l'uguaglianza $f(g(a)) \simeq b$, dove f e g appartengono a segnature di teorie diverse. L'uguaglianza $f(g(a)) \simeq b$ viene riscritta come $\{f(c) \simeq b,\ g(a) \simeq c\}$ dove c è una costante nuova.

In questo modo $\mathcal{S}$ viene trasformato in insiemi $\mathcal{S}_1, \ldots, \mathcal{S}_n$ tali che $\mathcal{S}_i$ è un insieme di $\mathcal{T}_i$-letterali, per $i = 1, \ldots n$, e per ogni i e j, $1 \leq i \neq j \leq n$, $\mathcal{S}_i$ ed $\mathcal{S}_j$

hanno in comune solo costanti libere. Poiché si introducono solo costanti, il problema rimane senza variabili (quantificate), e poiché le costanti introdotte sono nuove, $\bigcup_{i=1}^{n} \mathcal{S}_i$ è $\mathcal{T}$-soddisfacibile se e solo se lo è $\mathcal{S}$. Ogni $\mathcal{T}_i$-solutore si occupa del sottoproblema $\mathcal{S}_i$. La versatilità del metodo di condivisione di uguaglianze discende dalla sua capacità di minimizzare l'interazione tra i $\mathcal{T}_i$-solutori, che vengono integrati come scatole chiuse.

Tuttavia, poiché le teorie possono avere tipi in comune, i $\mathcal{T}_i$-solutori devono poter concordare sulle cardinalità dei dominii di interpretazione dei tipi condivisi. Per questo il metodo di condivisione di uguaglianze assume che le teorie siano *stabilmente infinite*. Una teoria $\mathcal{T}$ è *stabilmente infinita* se ogni $\mathcal{T}$-formula senza quantificatori che è $\mathcal{T}$-soddisfacibile ha un $\mathcal{T}$-modello infinito numerabile. Grazie a questo requisito, si assume che i dominii di interpretazione dei tipi condivisi siano infiniti numerabili. L'assunzione è implicita, ma gioca un ruolo nella dimostrazione di completezza del metodo di condivisione di uguaglianze (cfr. il capitolo 10 di [39]).

Da questa dimostrazione si vede che quel che occorre scambiare tra le teorie sono *interpolanti*, nel senso dell'interpolazione di Craig [51, 52]. Dati due insiemi di clausole $\mathcal{S}$ e $\mathcal{R}$ tali che $\mathcal{S} \cup \mathcal{R} \vdash \square$, un interpolante di $\mathcal{S}$ e $\mathcal{R}$ è una formula I tale che $\mathcal{S} \vdash I$ e $\mathcal{R}, I \vdash \square$, e tutti i simboli liberi che compaiono in I compaiono sia in $\mathcal{S}$ che in $\mathcal{R}$. Quando si ragiona in una teoria $\mathcal{T}$, $\vdash$ è sostituito da $\vdash_{\mathcal{T}}$, che è il simbolo di derivabilità in $\mathcal{T}$. La restrizione sui simboli che possono apparire nell'interpolante è relativa ai simboli liberi, per cui i simboli della teoria possono apparirvi liberamente. La stabilità infinita assicura che sia sufficiente propagare interpolanti senza quantificatori. Più precisamente, l'offerta infinita di elementi da un dominio infinito numerabile permette di dimostrare un lemma di *eliminazione di quantificatori* che fa parte della dimostrazione di completezza del metodo di condivisione di uguaglianze. Questo lemma garantisce che un interpolante con quantificatori possa essere sostituito da un interpolante equivalente senza quantificatori (si vedano [77] e il capitolo 10 di [39] per maggiori dettagli). In pratica, data l'assunzione che tutte le teorie siano stabilmente infinite, i $\mathcal{T}_i$-solutori non debbono far nulla riguardo alle cardinalità.

Poiché le teorie possono avere costanti libere in comune, i $\mathcal{T}_i$-solutori devono poter concordare un *arrangiamento* delle costanti condivise, che dica quali sono uguali e quali non lo sono. Per costruire un arrangiamento condiviso, il metodo di condivisione di uguaglianze richiede che ciascun $\mathcal{T}_i$-solutore sia in grado di dedurre e propagare agli altri ogni disgiunzione di uguaglianze $c_1 \simeq d_1 \vee \ldots \vee c_n \simeq d_n$ tra costanti condivise, che sia conseguenza logica in $\mathcal{T}_i$ del sottoproblema $\mathcal{S}_i$. L'analisi dei casi per queste disgiunzioni, così come per ogni altra disgiunzione che un $\mathcal{T}_i$-solutore possa generare (si ricordi la nozione di $\mathcal{T}$-lemma nella Sezione 3.1.3) viene svolta dal SAT-solutore.

Quando $\mathcal{T}$ è una combinazione di teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$, il $\mathcal{T}$-solutore integrato in DPLL($\mathcal{T}$) è una combinazione di $\mathcal{T}_i$-solutori secondo il metodo di condivisione di eguaglianze. La nozione che le disgiunzioni $c_1 \simeq d_1 \vee \ldots \vee c_n \simeq d_n$ siano gestite dal SAT-solutore è stata chiamata *suddivisione su richiesta*, dall'inglese "splitting on demand" [8, 100]. La limitazione di DPLL($\mathcal{T}$) per cui non è permesso creare nuovi atomi durante la derivazione viene allentata per permettere l'introduzione di un numero finito di "nuovi" atomi proposizionali, uno per ogni distinta uguaglianza che appaia in una disgiunzione propagata $c_1 \simeq d_1, \ldots, c_n \simeq d_n$ di uguaglianze tra costanti condivise. Poichè vi è un numero finito di costanti, vi è anche un numero finito di uguaglianze tra costanti e quindi questo allentamento non modifica in modo sostanziale la dimostrazione di terminazione di DPLL($\mathcal{T}$).

3.1.5 Da una teoria a molte: combinazione basata su modelli

Un altro approccio alla combinazione di teorie è la *combinazione di teorie basata su modelli*, abbreviato MBTC dall'inglese "Model-Based Theory Combination" [137, 58]. Questo approccio si applica quando i $\mathcal{T}_i$-solutori sono capaci di costruire $\mathcal{T}_i$-modelli candidati parziali. Quindi, ogni $\mathcal{T}_i$-solutore "propaga" ogni uguaglianza $s \simeq t$ tra termini senza variabili che risulti vera nel $\mathcal{T}_i$-modello candidato corrente, invece che disgiunzioni di uguaglianze tra costanti condivise. MBTC permette al $\mathcal{T}_i$-solutore di aggiungere una tale uguaglianza $s \simeq t$, o meglio la sua astrazione $\alpha(s \simeq t)$, alla traccia $\mathcal{M}$, purché $s \simeq t$ sia indefinita in $\mathcal{M}$. In questo modo, MBTC riequilibra un po' il rapporto tra SAT-solutore e T-solutori, riducendo il primato del SAT-solutore, così netto in DPLL($\mathcal{T}$), dove solo il SAT-solutore scrive sulla traccia. Si noti tuttavia che il termine "propaga", sebbene invalso in letteratura, e utile a confrontare MBTC con il metodo di condivisione di uguaglianze, non è adatto. Infatti, non si tratta della propagazione di una conseguenza logica, come nella BCP (cfr. la Sezione 3.1.1), nella $\mathcal{T}$-propagazione (cfr. la Sezione 3.1.3), o nel metodo di condivisione di uguaglianze (cfr. la Sezione 3.1.4), ma di una *decisione*, come nell'analisi dei casi (cfr. la Sezione 3.1.1).

Chiaramente un'uguaglianza siffatta $s \simeq t$ può causare un conflitto, precisamente perché non è necessariamente una conseguenza logica in $\mathcal{T}_i$ del sottoproblema $\mathcal{S}_i$ a cui lavora il $\mathcal{T}_i$-solutore, ma è soltanto vera nel $\mathcal{T}_i$-modello candidato corrente. Se questo accade, il conflitto viene gestito nel solito modo da CDCL, e $\alpha(s \simeq t)$ viene rimosso dalla traccia saltando all'indietro. La ratio di MBTC è di tipo pratico: è generalmente meno costoso enumerare le uguaglianze vere in un particolare $\mathcal{T}_i$-modello piuttosto che enumerare le uguaglianze vere in tutti i $\mathcal{T}_i$-modelli consistenti con $\mathcal{M}$. Nel più degli esperi-

menti, il numero di uguaglianze rilevanti per determinare la soddisfacibilità o insoddisfacibilità del problema risulta essere basso.

Tuttavia, MBTC *non genera atomi nuovi*, perché la propagazione di un'uguaglianza $s \simeq t$ viene permessa solo se s e t appaiono nell'insieme di clausole esistente. MBTC si applica primariamente al frammento senza quantificatori della teoria dell'uguaglianza e a frammenti dell'aritmetica. Per il frammento senza quantificatori della teoria dell'uguaglianza, si interpreta come modello parziale candidato il grafo che l'algoritmo di chiusura di congruenza usa per costruire la chiusura di congruenza delle uguaglianze in ingresso (si vedano [65, 58] e il capitolo 9 di [39]). Per i frammenti dell'aritmetica, dominio di interpretazione e interpretazione dei simboli della teoria sono fissati da un modello inteso (e.g., gli interi), e sono noti algoritmi per costruire un modello parziale e aggiornarlo dopo un conflitto [69, 58].

MBTC è un approccio alla combinazione di teorie nel contesto di un sistema per soddisfacibilità modulo teorie costruito sopra un SAT-solutore basato sulla procedura CDCL. Dunque, il ragionamento sui conflitti avviene solo in logica proposizionale. Tuttavia, con la nozione di permettere la propagazione di uguaglianze vere in un $\mathcal{T}$-modello parziale candidato, ma non necessariamente in tutti, MBTC prepara il terreno per la deduzione guidata da conflitti nella teoria $\mathcal{T}$ e non solo in logica proposizionale.

3.1.6 DPLL(Γ+$\mathcal{T}$): un metodo per la logica del primo ordine che incorpora CDCL

Nei problemi prodotti dalle applicazioni, appaiono spesso anche quantificatori. I quantificatori esistenziali vengono eliminati mediante Skolemizzazione durante la trasformazione del problema in forma clausale. Restano quindi clausole con variabili implicitamente quantificate universalmente. In questa sezione, con variabile si intende una variabile quantificata universalmente. Per esempio, i problemi generati in ambito di verifica di programmi richiedono sovente di determinare la $\mathcal{T}$-soddisfacibilità di un insieme di clausole $\mathcal{S} = \mathcal{P} \uplus \mathcal{R}$, dove $\mathcal{T}$ è una combinazione di teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$, P è un insieme di clausole *senza variabili* con occorrenze di $\mathcal{T}$-simboli, $\mathcal{R}$ è un insieme di clausole *con variabili* senza occorrenze di $\mathcal{T}$-simboli, e $\uplus$ denota l'unione disgiunta. L'insieme $\mathcal{R}$ può essere l'assiomatizzazione di una teoria per la quale non abbiamo una procedura di $\mathcal{R}$-soddisfacibilità. Chiaramente, questo tipo di problema è più generale del decidere la $\mathcal{T}$-soddisfacibilità di un insieme di clausole senza variabili.

Per i quantificatori, ci sono situazioni dove è possibile trovare le istanze delle variabili quantificate universalmente necessarie e sufficienti a determina-

re la $\mathcal{T}$-soddisfacibilità dell'insieme. Questo è il caso, per esempio, in certi frammenti di teorie degli array [40, 23] e dei puntatori [115], o in *estensioni locali* di teorie (per queste ultime si vedano [133, 90] e la Sezione 3 di [24]). Al di fuori di questi casi speciali, si sono investigate tecniche per indovinare come istanziare le variabili, sulla base di euristiche e annotazioni date dall'utente [65, 56, 78]. Per esempio, sebbene ci siano procedure di $\mathcal{T}$-soddisfacibilità per la teoria degli *array con estensionalità* [135], la maggiore parte dei sistemi per la soddisfacibilità modulo teorie legge in ingresso gli assiomi di questa teoria come parte dell'insieme $\mathcal{S}$, e applica tecniche euristiche per istanziare le variabili quantificate universalmente negli assiomi. Queste tecniche possono essere molto efficienti quanto hanno successo, ma richiedono molta cura e tempo all'utente, e restano inerentemente incomplete. L'incompletezza del sistema per la soddisfacibilità modulo teorie fa sì che il sistema di verifica di programmi che lo chiama riporti risultati che sono falsi positivi. In altre parole, il compilatore-verificatore o analizzatore statico segnala un errore che non c'è.

D'altro canto, i *sistemi di inferenza basati su ordinamenti* (cfr. la Sezione 2) sono *refutazionalmente completi* per la logica del primo ordine con uguaglianza; e sono particolarmente versati nel ragionare con l'uguaglianza, le variabili quantificate universalmente, e le clausole di Horn. Equipaggiati con un piano di ricerca *equo*, questi sistemi di inferenza offrono: (1) procedure di semidecisione per la validità in logica del primo ordine con uguaglianza; e (2) procedure di $\mathcal{T}$-soddisfacibilità per i frammenti senza quantificatori della teoria dell'uguaglianza, di parecchie teorie di strutture di dati, incluse strutture di dati ricorsive e array con o senza estensionalità, e delle loro combinazioni [4, 2, 21, 22, 3].

Il sistema DPLL($\Gamma + \mathcal{T}$) [57, 32, 33] integra DPLL($\mathcal{T}$), un sistema di inferenza Γ basato su ordinamenti, ed MBTC, per risolvere problemi della summenzionata forma $\mathcal{S} = \mathcal{P} \uplus \mathcal{R}$. Nel nome DPLL($\Gamma + \mathcal{T}$), Γ e $\mathcal{T}$ sono due parametri: per esempio, alcuni dei risultati in [33] sono ottenuti assumendo che Γ usi l'*iperrisoluzione positiva* [124, 44]. Per $\mathcal{T}$ si assume che sia una combinazione di teorie *disgiunte* e *stabilmente infinite*. Per $\mathcal{R}$ si assume che sia *disgiunta* da $\mathcal{T}$ e *inattiva sulle variabili* [3], una proprietà sintattica e verificabile da Γ, sufficiente a implicare che $\mathcal{R}$ sia stabilmente infinita [25].

L'idea di DPLL($\Gamma + \mathcal{T}$) è di usare DPLL($\mathcal{T}$) per ragionare sulle clausole senza variabili e la teoria $\mathcal{T}$, e Γ per ragionare sulle clausole con variabili. In questo modo, ogni sistema viene applicato alla parte di problema per cui è più adatto. L'integrazione consiste nel fatto che DPLL($\Gamma + \mathcal{T}$) permette a Γ di usare come premesse delle sue inferenze gli $\mathcal{R}$-letterali senza variabili la cui astrazione proposizionale appare sulla traccia $\mathcal{M}$. In questo modo, anche le inferenze di Γ sono guidate in parte dal modello corrente. Il meccanismo

di MBTC partecipa all'integrazione, perché anche le uguaglianze messe sulla traccia da MBTC verranno viste da Γ.

L'integrazione di sistema di inferenza basato su ordinamenti e solutore che sviluppa una ricerca con salti all'indietro presenta difficoltà non banali. Una prima difficoltà è che i letterali che sono sulla traccia $\mathcal{M}$ non vi sono permanentemente, ma possono sparire a causa di salti all'indietro. Dunque DPLL(Γ+$\mathcal{T}$) lavora con *clausole con ipotesi*: se una clausola C viene generata da una inferenza di Γ usando tra le premesse un letterale L dalla traccia $\mathcal{M}$, la clausola viene generata nella forma $L \triangleright C$, per dire che abbiamo derivato C sotto l'ipotesi che L sia vero nel modello candidato. Qualsiasi clausola generata da $L \triangleright C$ eredita a sua volta l'ipotesi L. Se un salto all'indietro rimuove L da $\mathcal{M}$, tutte le clausole le cui ipotesi includono L vengono rimosse. In generale, le clausole ereditano le ipotesi delle premesse che le generano, dando luogo a clausole della forma $(L_1 \wedge \ldots \wedge L_m) \triangleright (N_1 \vee \ldots \vee N_r)$, che viene interpretata come $\neg L_1 \vee \ldots \vee \neg L_m \vee N_1 \vee \ldots \vee N_r$.

Una seconda difficoltà è data dalla presenza in Γ di regole di contrazione, che eliminano, invece che generare, clausole. Bisogna dunque prevenire la situazione scorretta in cui una clausola D viene eliminata, perché viene sussunta o semplificata da una clausola $L \triangleright C$, e poi $L \triangleright C$ sparisce, perché L viene rimosso dalla traccia $\mathcal{M}$ con un salto all'indietro. DPLL(Γ+$\mathcal{T}$) risolve questo problema adattando le regole di contrazione di Γ alle clausole con ipotesi. A una clausola $L_1 \wedge \ldots \wedge L_m \triangleright C$ viene associato il livello di decisione più alto tra quelli di $L_1, \ldots, L_m$. Si ricordi che i letterali $L_1, \ldots, L_m$ appaiono sulla traccia $\mathcal{M}$ e quindi ciascuno appartiene a un livello di decisione. Supponiamo che L_m sia quello con il livello più alto: appena un salto all'indietro rimuove il livello di L_m anche $L_1 \wedge \ldots \wedge L_m \triangleright C$ sparisce. Omettendo per brevità le ipotesi, sia p il livello di D e q quello di C, dove C sussume o semplifica D. Se $q \leq p$, non è possibile che un salto all'indietro rimuova C prima di D. Dunque il passo di sussunzione o semplificazione si può eseguire normalmente, e D viene definitivamente sussunta o semplificata. Se invece $q > p$, la clausola D viene soltanto *disabilitata*, e potrà essere restaurata qualora C venga rimossa da un salto all'indietro. Nel caso della semplificazione, se C riduce D a D', D viene soltanto disabilitata, un salto all'indietro rimuove C, e D viene restaurata, si noti che il salto all'indietro che rimuove C rimuove anche D', perché D' eredita le ipotesi di C e D.

DPLL(Γ+$\mathcal{T}$) è *refutazionalmente completo* [33], e quindi se $\mathcal{S} = \mathcal{P} \uplus \mathcal{R}$ è insoddisfacibile, esistono derivazioni che portano a una refutazione. Per ottenere una procedura di semidecisione per la validità di problemi nella forma $\mathcal{S} = \mathcal{P} \uplus \mathcal{R}$, occorre equipaggiare il sistema di inferenza con un piano di ricerca equo. DPLL($\mathcal{T}$) adotta una ricerca per profondità con salto all'indietro, che è

equa solo per problemi con spazio di ricerca finito. In presenza di $\mathcal{R}$ e Γ, cioè della parte del problema in logica del primo ordine, lo spazio di ricerca non è finito in generale. Dunque occorre adottare un piano di ricerca con *approfondimento iterativo* [96, 134]. La terminazione in caso di ingresso soddisfacibile è naturalmente assai più difficile, come mostra l'esempio seguente [33].

Esempio 3.12: Si assuma che $\mathcal{R}$ contenga l'assioma di monotonia

$$\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$$

che statuisce la monotonia della funzione f rispetto alla relazione $\sqsubseteq$. La risoluzione senza restrizioni risolve l'assioma con sè stesso, ovvero, con una sua variante

$$\neg(x' \sqsubseteq y') \vee f(x') \sqsubseteq f(y')$$

generando il risolvente $\neg(x \sqsubseteq y) \vee f^2(x) \sqsubseteq f^2(y)$, e poi la serie infinita

$$\{\neg(x \sqsubseteq y) \vee f^i(x) \sqsubseteq f^i(y)\}_{i \geq 0},$$

dove a ogni passo l'assioma originale risolve con $\neg(x \sqsubseteq y) \vee f^{i-1}(x) \sqsubseteq f^{i-1}(y)$ generando il suo successore nella serie. L'iperrisoluzione positiva non incorre in questo comportamento, perché permette soltanto di risolvere simultaneamente una clausola nucleo non positiva con una o più clausole satelliti positive a generare una clausola positiva. Tuttavia, anche l'iperrisoluzione positiva genera una serie infinita

$$\{f^i(a) \sqsubseteq f^i(b)\}_{i \geq 0}$$

dall'assioma di monotonia e una clausola unitaria $a \sqsubseteq b$ che si trovi nell'insieme d'ingresso. In pratica, se l'insieme d'ingresso è soddisfacibile, ben raramente occorre andare oltre $f(a) \sqsubseteq f(b)$ o $f^2(a) \sqsubseteq f^2(b)$ per mostrare la soddisfacibilità.

L'integrazione di sistema di inferenza per la logica del primo ordine e solutore che effettua salti all'indietro permette *inferenze speculative reversibili* [32, 33]. Un'*inferenza speculativa* è un'inferenza che aggiunge all'insieme di clausole una clausola arbitraria. Questo forza il sistema a cercare un modello che soddisfi sia l'insieme di ingresso $\mathcal{S}$ che la clausola aggiunta in modo speculativo. L'idea è che può accadere che la ricerca di un modello per $\mathcal{S}$ non termini, ma termini la ricerca di un modello di $\mathcal{S}$ che soddisfi anche le clausole aggiunte speculativamente. Però clausole aggiunte speculativamente possono cambiare il problema, trasformando un insieme soddisfacibile in uno insoddisfacibile. Per questo occorre che le inferenze speculative siano *reversibili*.

A questo scopo DPLL($\Gamma + \mathcal{T}$) tiene traccia in $\mathcal{M}$ delle clausole aggiunte in modo speculativo. Se C viene aggiunta in modo speculativo, la sua aggiunta viene registrata mettendo su $\mathcal{M}$ un atomo proposizionale nuovo $\lceil C \rceil$, e C viene aggiunta nella forma $\lceil C \rceil \triangleright C$. Se $\lceil C \rceil \triangleright C$ causa inconsistenza, l'inconsistenza emergerà come un conflitto, e $\lceil C \rceil$ e $\lceil C \rceil \triangleright C$ verranno ritirate da un salto all'indietro. In questo modo le inferenze speculative sono *reversibili*. Il punto cruciale è aggiungere speculativamente clausole che facilitino la terminazione in caso di ingresso soddisfacibile. Nell'esempio seguente [33], questa funzione è svolta da uguaglianze che limitano la profondità dei termini mediante semplificazione.

Esempio 3.13: Sia $\mathcal{R}$ l'insieme

$$\{\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z,\ \neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)\},$$

che aggiunge all'assioma di monotonia dell'esempio 3.12 l'assioma di transitività della relazione $\sqsubseteq$. Sia P l'insieme

$$\{a \sqsubseteq b,\ a \sqsubseteq f(c),\ \neg(a \sqsubseteq c)\}.$$

Assumiamo che Γ abbia la risoluzione, la sovrapposizione e la semplificazione. Supponiamo che un'inferenza speculativa aggiunga

$$\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$$

all'insieme di clausole, mettendo $\lceil f(x) \simeq x \rceil$ sulla traccia $\mathcal{M}$. Supponendo di essere all'inizio della derivazione, $\lceil f(x) \simeq x \rceil$ è un letterale deciso al primo livello della traccia. La clausola $\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$ riduce l'assioma di monotonia a una tautologia, e $a \sqsubseteq f(c)$ a $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$. L'assioma di monotonia e $a \sqsubseteq f(c)$ non hanno ipotesi, essendo clausole di ingresso, e quindi sono associate al livello 0. Poiché $\lceil f(x) \simeq x \rceil$ appartiene al primo livello, segue che questi passi di semplificazione disabilitano soltanto l'assioma di monotonia e $a \sqsubseteq f(c)$. A questo punto un passo di risoluzione tra $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$ e $\neg(a \sqsubseteq c)$ genera la clausola $\lceil f(x) \simeq x \rceil \triangleright \Box$, che essendo vuota è in conflitto con qualsiasi modello. La clausola $\lceil f(x) \simeq x \rceil \triangleright \Box$ è equivalente a $\neg\lceil f(x) \simeq x \rceil$, che è unitaria e quindi di asserzione. Dunque DPLL($\Gamma + \mathcal{T}$) impara $\neg\lceil f(x) \simeq x \rceil$, salta all'indietro al livello 0, e mette $\neg\lceil f(x) \simeq x \rceil$ sulla traccia $\mathcal{M}$. In questo modo, DPLL($\Gamma + \mathcal{T}$) non potrà ripetere l'inferenza speculativa che propone $f(x) \simeq x$. Supponiamo ora che una successiva inferenza speculativa aggiunga

$$\lceil f(f(x)) \simeq x \rceil \triangleright f(f(x)) \simeq x.$$

In presenza di questa clausola, l'assioma di monotonia e $a \sqsubseteq b$ producono solo $f(a) \sqsubseteq f(b)$. L'assioma di monotonia e $a \sqsubseteq f(c)$ producono solo $f(a) \sqsubseteq f(f(c))$, che viene disabilitata e sostituita da $\lceil f(f(x)) = x \rceil \triangleright f(a) \sqsubseteq c$. Ora nessun'altra inferenza è possibile, e DPLL($\Gamma + \mathcal{T}$) dichiara l'insieme soddisfacibile.

L'esempio successivo [33] mostra una situazione in cui MBTC gioca un ruolo chiave.

Esempio 3.14: Sia Γ l'insieme di inferenza con iperrisoluzione positiva, sovrapposizione, e semplificazione, e sia $\mathcal{T}$ la teoria dell'aritmetica lineare sugli interi. Supponiamo di avere in ingresso l'assioma di transitività di $\sqsubseteq$

$$\mathcal{R} = \{\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z\}$$

e le clausole senza variabili

$$P = \{a \sqsubseteq b_1,\ b_2 \sqsubseteq c,\ \neg(a \sqsubseteq c),\ b_1 \leq b_2,\ b_1 > b_2 - 1\}.$$

DPLL($\Gamma+\mathcal{T}$) inizia mettendo i letterali di P al livello 0 sulla traccia $\mathcal{M}$: come in CDCL, sono clausole unitarie da soddisfare, per cui $\mathcal{M} = (\ a \sqsubseteq b_1,\ b_2 \sqsubseteq c,\ \neg(a \sqsubseteq c), b_1 \leq b_2,\ b_1 > b_2 - 1\ )$. Nel modello costruito dal solutore per l'aritmetica lineare, b_1 e b_2 sono interpretati come uguali. Quindi, la regola di inferenza di DPLL($\Gamma+\mathcal{T}$) che realizza MBTC propone l'uguaglianza $b_1 \simeq b_2$, mettendola sulla traccia come un letterale deciso al primo livello:

$$\mathcal{M} = (\ a \sqsubseteq b_1,\ b_2 \sqsubseteq c,\ \neg(a \sqsubseteq c), b_1 \leq b_2,\ b_1 > b_2 - 1,\ b_1 \simeq b_2\ ).$$

Supponiamo che sia $b_2 \succ b_1$ nell'ordinamento $\succ$ di Γ: la semplificazione applica $b_1 \simeq b_2$ a riscrivere $b_2 \sqsubseteq c$ in

$$(b_1 \simeq b_2) \triangleright b_1 \sqsubseteq c.$$

L'iperrisoluzione usa l'assioma di transitività come nucleo e $a \sqsubseteq b_1$ e $b_1 \simeq b_2 \triangleright b_1 \sqsubseteq c$ come satelliti per derivare

$$(b_1 \simeq b_2) \triangleright a \sqsubseteq c.$$

Un altro passo di iperrisoluzione tra nucleo $\neg(a \sqsubseteq c)$ e satellite $b_1 \simeq b_2 \triangleright a \sqsubseteq c$ genera

$$(b_1 \simeq b_2) \triangleright \square,$$

ovvero $\neg(b_1 \simeq b_2)$. Questa clausola è in conflitto, perché la traccia $\mathcal{M}$ contiene $b_1 \simeq b_2$, ed è di asserzione. DPLL($\Gamma+\mathcal{T}$) salta all'indietro al livello 0, e vi

aggiunge $\neg(b_1 \simeq b_2)$, risolvendo il conflitto. Ora però la regola di $\mathcal{T}$-conflitto segnala che il sottoinsieme

$$\{b_1 \leq b_2,\ b_1 > b_2 - 1,\ \neg(b_1 \simeq b_2)\}$$

di $\mathcal{M}$ è inconsistente nell'aritmetica lineare sugli interi, producendo la clausola

$$\neg(b_1 \leq b_2) \vee \neg(b_1 > b_2 - 1) \vee b_1 \simeq b_2,$$

che è in conflitto con il livello 0 di $\mathcal{M}$. Pertanto DPLL($\Gamma + \mathcal{T}$) restituisce "insoddisfacibile". La clausola vuota viene generata come segue: un passo di iperrisoluzione tra nucleo $\neg(b_1 \leq b_2) \vee \neg(b_1 > b_2 - 1) \vee b_1 \simeq b_2$ e satelliti $b_1 \leq b_2$ e $b_1 > b_2 - 1$, deriva $b_1 \simeq b_2$; un altro passo di iperrisoluzione tra nucleo $\neg(b_1 \simeq b_2)$ e satellite $b_1 \simeq b_2$ produce $\square$. Per avere una dimostrazione completa tuttavia occorre che la procedura di soddisfacibilità per l'aritmetica lineare produca una dimostrazione di $\neg(b_1 \leq b_2) \vee \neg(b_1 > b_2 - 1) \vee b_1 \simeq b_2$.

Come negli esempi, clausole aggiunte in modo speculativo possono indurre la terminazione in caso di ingresso soddisfacibile, permettendo al sistema di scoprire l'esistenza di un modello. DPLL($\Gamma + \mathcal{T}$) è una procedura di $\mathcal{T}$-decisione con inferenze speculative per alcune classi di problemi nella forma $\mathcal{S} = \mathcal{P} \uplus \mathcal{R}$ che soddisfano certe proprietà. Esempi includono casi in cui $\mathcal{R}$ contiene *assiomatizzazioni di sistemi di tipi* [33]. Tuttavia, la parte di deduzione guidata da conflitti è proposizionale in DPLL($\Gamma+\mathcal{T}$) come in DPLL($\mathcal{T}$). Nella sezione successiva consideriamo metodi che portano la deduzione guidata da conflitti *dentro* la teoria.

3.2 Deduzione guidata da conflitti in una teoria

Per avere deduzione guidata da conflitti in una teoria $\mathcal{T}$, occorre che i meccanismi per *spiegare* un $\mathcal{T}$-conflitto, *imparare* un $\mathcal{T}$-lemma, e *risolvere* il $\mathcal{T}$-conflitto, operino all'interno del $\mathcal{T}$-solutore, e non solo a livello proposizionale nel SAT-solutore. In altre parole, occorre che il $\mathcal{T}$-solutore implementi una *procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti*.

3.2.1 Procedure di $\mathcal{T}$-soddisfacibilità guidate da conflitti

Procedure siffatte sono note per l'aritmetica lineare sui reali [114, 97, 50], l'aritmetica lineare sugli interi [140, 137, 94], l'aritmetica non lineare [95], e l'aritmetica binaria a virgola mobile [87]. Queste procedure sono state ottenute

ricreando i meccanismi di CDCL in teorie specifiche. Infatti, queste procedure mantengono una traccia per rappresentare un $\mathcal{T}$-modello parziale candidato, e vi permettono *assegnamenti al primo ordine*, come $x \leftarrow 3$, dove x è una variabile libera che appare nel problema. Inoltre, queste procedure possono *spiegare* un $\mathcal{T}$-conflitto generando una clausola che contiene atomi *nuovi*. L'esempio seguente [26] illustra sia gli assegnamenti al primo ordine che la generazione di atomi nuovi.

Esempio 3.15: Sia dato l'insieme d'ingresso

$$\mathcal{S} = \{l_0 : (-2 \cdot x - y < 0),\ l_1 : (x + y < 0),\ l_2 : (x < -1)\},$$

che è insoddisfacibile in aritmetica lineare sui razionali. Come nell'esempio 3.8, x e y sono variabili libere, ovvero implicitamente quantificate in modo esistenziale. Dal punto di vista logico, come già osservato, x e y possono essere sostituite da simboli di costante o trattate come simboli di costante, ma per chi legge è sovente più intuitivo chiamarle variabili. Determinare la soddisfacibilità di $\mathcal{S}$ significa trovare valori di x e y che soddisfino le tre clausole di $\mathcal{S}$. Una procedura di soddisfacibilità guidata da conflitti tenta di costruire un modello indovinando un valore per una delle variabili, poniamo $y \leftarrow 0$. Con questo assegnamento, l_0 produce l'estremo inferiore $x > 0$ per la variabile x. Dato l'estremo superiore per x rappresentato da l_2, lo spazio dei valori possibili per x è vuoto, rivelando che l'assegnamento $y \leftarrow 0$ e le clausole di $\mathcal{S}$ sono in conflitto. La procedura *spiega* il conflitto derivando la clausola

$$l_3 : (-y < -2),$$

risultato della combinazione lineare di l_0 ed l_2 che elimina la variabile x. Si noti che l'atomo di questa clausola è *nuovo*. Questa spiegazione esclude non solo $y \leftarrow 0$, ma anche tutti gli assegnamenti $y \leftarrow \mathfrak{c}$ tali che $\mathfrak{c} \leq 2$, dove $\mathfrak{c}$ denota un qualsiasi numero razionale. Supponiamo che la procedura ritiri l'assegnamento $y \leftarrow 0$ e tenti $y \leftarrow 4$. Con questo nuovo assegnamento, la clausola l_1 produce l'estremo superiore $x < -4$, mentre la clausola l_0 ci dà l'estremo inferiore $x > -2$. La procedura *spiega* questo conflitto con la clausola

$$l_4 : (y < 0),$$

risultato della combinazione lineare di l_0 ed l_1 che elimina la variabile x. Di nuovo, l'atomo di questa clausola è nuovo. Siccome l_4 è violato dall'assegnamento $y \leftarrow 4$, la procedura lo ritira. Ora nessun assegnamento a y può soddisfare sia l_3 che l_4. Questo terzo conflitto viene *spiegato* con la combinazione lineare di l_3 ed l_4 che elimina y, ovvero $0 < -2$, anche questo un atomo nuovo. Poiché questa conseguenza di $\mathcal{S}$ è una contraddizione nella teoria, la procedura restituisce "insoddisfacibile".

Sebbene i frammenti dell'aritmetica siano stati l'ambito principale di ricerca su procedure di $\mathcal{T}$-soddisfacibilità guidate da conflitti, qualche progresso è stato fatto anche per la teoria degli *array con estensionalità* [42], sviluppando la nozione di *lemma su richiesta* [64]. L'idea dei *lemmi su richiesta* è che il $\mathcal{T}$-solutore generi $\mathcal{T}$-lemmi che *spiegano* perché il contenuto della traccia $\mathcal{M}$ sia inconsistente nella teoria. Questo è qualcosa di più rispetto a generare la clausola in $\mathcal{T}$-conflitto data dalla disgiunzione delle negazioni dei letterali del sottoinsieme di $\mathcal{M}$ che risulta inconsistente in $\mathcal{T}$, come già fatto dalla regola di $\mathcal{T}$-conflitto in DPLL($\mathcal{T}$) e DPLL($\Gamma+\mathcal{T}$) (cfr. la Sezione 3.1.3 e l'esempio 3.14 nella Sezione 3.1.6). In logica proposizionale, i *lemmi su richiesta* sono i risolventi generati dalla spiegazione del conflitto mediante risoluzione. Si vorrebbe che anche la $\mathcal{T}$-propagazione sia basata sul modello e guidata da conflitti, nel senso di generare $\mathcal{T}$-lemmi che spieghino i conflitti. La regola di $\mathcal{T}$-propagazione in DPLL($\mathcal{T}$) è basata sul modello, ma non è guidata da conflitti, perché permette di generare qualsiasi $\mathcal{T}$-lemma consistente con lo stato della traccia (cfr. la Sezione 3.1.3).

La procedura di $\mathcal{T}$-soddisfacibilità per gli array con lemmi su richiesta [42] include regole di inferenza che propagano l'applicazione del simbolo di funzione $read$, o $select$, che rappresenta l'operazione di lettura su un array nella segnatura della teoria degli array. I lemmi generati hanno la forma $\neg L_1 \vee \ldots \vee \neg L_k \vee L$, dove $L_1, \ldots, L_k$ sono veri e L è falso nel modello candidato corrente $\mathcal{M}$, mentre L dovrebbe essere vero secondo gli assiomi della teoria. Questi lemmi rivelano che $\mathcal{M}$ non è un modello per la teoria e dicono perché. Spesso tuttavia i lemmi generati da questa procedura sono istanze degli assiomi, così che si ha in sostanza istanziazione degli assiomi guidata da conflitti. Il problema successivo è come ottenere una *procedura di $\mathcal{T}$-decisione guidata da conflitti*.

3.2.2 Procedure di $\mathcal{T}$-decisione guidate da conflitti

Le procedure di $\mathcal{T}$-soddisfacibilità guidate da conflitti [114, 97, 50, 94, 95, 87] *non* sono compatibili in generale con DPLL($\mathcal{T}$), e quindi non si può ottenere una procedura di $\mathcal{T}$-decisione guidata da conflitti integrando una procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti in DPLL($\mathcal{T}$). Una ragione principale di incompatibilità è precisamente che DPLL($\mathcal{T}$) non permette la creazione di atomi nuovi durante la derivazione, mentre una procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti può spiegare un $\mathcal{T}$-conflitto generando una clausola che contiene atomi nuovi, come illustrato nell'esempio 3.15.

Il problema di integrare CDCL con una procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti è stato risolto dal sistema MCSAT, che sta per "Model-

Constructing SATisfiability" ovvero *Soddisfacibilità con Costruzione di Modello* [60]. MCSAT è un paradigma per ottenere *procedure di $\mathcal{T}$-decisione guidate da conflitti* per una singola generica teoria $\mathcal{T}$ [60]. È stato istanziato per la combinazione del frammento senza quantificatori della teoria dell'uguaglianza e l'aritmetica lineare sui reali [93], l'aritmetica non lineare sugli interi [92], e la teoria dei vettori di bit [141].

MCSAT combina il modello proposizionale di CDCL con il $\mathcal{T}$-modello della procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti, mettendo su un'unica traccia $\mathcal{M}$ sia i letterali che gli assegnamenti a variabili libere del primo ordine. Per esempio, $\mathcal{M}$ può contenere un letterale L, abbreviazione dell'assegnamento $L \leftarrow \mathit{vero}$, e un assegnamento come $x \leftarrow 3$. Quindi la traccia non è più una sequenza di letterali, ma una *sequenza di assegnamenti* che rappresenta un modello parziale candidato. Gli assegnamenti a variabili del primo ordine possono essere decisioni o propagazioni, che MCSAT chiama *decisioni semantiche* e *propagazioni semantiche*, per distinguerle da letterali decisi e letterali implicati in logica proposizionale. MCSAT generalizza CDCL a qualsiasi teoria $\mathcal{T}$ che possa essere equipaggiata con regole di inferenza clausali per *spiegare* $\mathcal{T}$-conflitti. Il prossimo esempio [61, 24] illustra questo concetto nel frammento senza quantificatori della teoria dell'uguaglianza.

Esempio 3.16: Sia $\mathcal{S}$ un insieme che include $\{v \simeq f(a),\ w \simeq f(b)\}$, dove f è un simbolo di funzione, e a, b, v e w sono indifferentemente costanti o variabili libere. Se la traccia $\mathcal{M}$ contiene gli assegnamenti $a \leftarrow \mathfrak{q}$, $b \leftarrow \mathfrak{q}$, $w \leftarrow \mathfrak{c}_1$, $v \leftarrow \mathfrak{c}_2$, dove $\mathfrak{q}$, $\mathfrak{c}_1$, e $\mathfrak{c}_2$ sono distinti valori del tipo appropriato, c'è un conflitto nella teoria dell'uguaglianza. La spiegazione è la formula $a \simeq b \Rightarrow f(a) \simeq f(b)$, che è un'istanza dell'assioma di sostitutività, o di congruenza, per la funzione f: $\forall x.\forall y.\ x \simeq y \Rightarrow f(x) \simeq f(y)$. Si noti che gli atomi $a \simeq b$ e $f(a) \simeq f(b)$ possono non apparire in $\mathcal{S}$, e in tale caso un simile lemma non potrebbe essere generato in DPLL($\mathcal{T}$).

L'esistenza di *regole d'inferenza di spiegazione dei $\mathcal{T}$-conflitti* è un ingrediente chiave per la deduzione guidata da conflitti in $\mathcal{T}$. La possibilità di imparare una clausola generata da una inferenza di spiegazione di un $\mathcal{T}$-conflitto, e usarla per emendare il modello candidato parziale, segue come in CDCL. In MCSAT, le inferenze che spiegano i $\mathcal{T}$-conflitti generano clausole che possono contenere atomi nuovi senza variabili (quantificate) nella segnatura della teoria $\mathcal{T}$, al di là di ciò che è permesso da DPLL($\mathcal{T}$) anche con suddivisione su richiesta. Gli assegnamenti a variabili del primo ordine e i nuovi atomi vengono coinvolti in decisioni, propagazioni, scoperte di conflitti, e spiegazioni. Insomma vengono trattati alla stessa stregua degli assegnamenti di valori di verità e degli atomi presenti nell'insieme $\mathcal{S}$ in ingresso. Questo

significa che la procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti *non viene integrata come una scatola chiusa*, ma coopera con il SAT-solutore allo stesso livello. La procedura CDCL stessa viene vista come una procedura di $\mathcal{T}$-soddisfacibilità guidata da conflitti dove $\mathcal{T}$ è la logica proposizionale. La generazione di atomi nuovi durante la derivazione è tuttavia problematica per la terminazione, come mostra l'esempio seguente [61, 24].

Esempio 3.17: Riprendiamo il problema $\mathcal{S} = \{x \geq 2,\ \neg(x \geq 1) \vee y \geq 1,\ x^2+y^2 \leq 1 \vee xy > 1\}$ degli esempi 3.8, 3.9 e 3.10. Si era rimasti allo stadio in cui la traccia è nello stato $\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1\ )$, dove lasciamo implicita l'applicazione della funzione di astrazione α che interviene quando la traccia viene vista dalla componente CDCL di MCSAT. Questo problema non può essere risolto con DPLL($\mathcal{T}$), ma bensì con MCSAT, purché sia equipaggiato con una procedura di $\mathcal{T}$-soddisfacibilità per l'aritmetica sufficientemente intelligente, e per questo lo riprendiamo a questo punto della trattazione. Supponiamo che la procedura decida che $x^2+y^2 \leq 1$ sia vero. Poi, tenta la decisione semantica $x \leftarrow 2$, così che abbiamo $\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ x^2+y^2 \leq 1,\ x \leftarrow 2\ )$. A questo punto sorge un $\mathcal{T}$-conflitto, perché non c'è nessun valore di y che possa soddisfare $4 + y^2 \leq 1$ in aritmetica. Dedurre ed imparare $\neg(x = 2)$ come spiegazione del conflitto non funziona, perché la procedura potrebbe poi tentare $x \leftarrow 3$, e incorrere in un altro conflitto. Chiaramente, non vogliamo che il sistema impari la sequenza infinita $\neg(x = 2),\ \neg(x = 3),\ \neg(x = 4) \ldots$.

MCSAT risolve questo problema imponendo che i nuovi atomi provengano da una *base* e che questa base sia *finita*. La dimostrazione della terminazione di MCSAT riposa su questa assunzione [60]. A livello teorico si può semplicemente assumere che una base esista e richiedere che sia finita. In pratica, la base è data dalla chiusura dell'insieme degli atomi che appaiono nel problema $\mathcal{S}$ in ingresso rispetto alle regole di inferenza usate per spiegare i conflitti nella teoria $\mathcal{T}$. Quindi, il requisito che la base sia finita diventa un requisito per le regole di inferenza di spiegazione dei $\mathcal{T}$-conflitti. Tuttavia, una procedura che applichi sistematicamente le regole di inferenza a enumerare tutti gli atomi nella base finita sarebbe troppo inefficiente. Il punto cruciale è che le regole di inferenza siano applicate solo a spiegare i $\mathcal{T}$-conflitti e a emendare il modello candidato corrente, proprio come accade in CDCL per la risoluzione. In questo modo, anche la generazione di atomi nuovi è guidata dai conflitti.

L'interpolazione (si vedano [31, 30] per recenti risultati e per lo stato dell'arte sull'interpolazione in deduzione automatica) gioca un ruolo in questo contesto. Infatti, una *spiegazione* di un conflitto tra insieme di clausole $\mathcal{S}$ e traccia $\mathcal{M}$ è una formula che segue da $\mathcal{S}$ ed è inconsistente con $\mathcal{M}$. Nel caso più semplice, quello della logica proposizionale, per spiegare un conflitto

risolviamo la clausola in conflitto con una giustificazione, ed otteniamo un risolvente che è una conseguenza logica di $\mathcal{S}$ ed è ancora in conflitto con $\mathcal{M}$ (cfr. la Sezione 3.1.2). Se una spiegazione è una formula che segue da $\mathcal{S}$ ed è inconsistente con $\mathcal{M}$, un interpolante di $\mathcal{S}$ ed $\mathcal{M}$ (vista quest'ultima come una formula) è una spiegazione. Illustriamo questo nesso continuando l'esempio 3.17.

Esempio 3.18: Abbiamo l'insieme $\mathcal{S} = \{x \geq 2,\ \neg(x \geq 1) \vee y \geq 1,\ x^2 + y^2 \leq 1 \vee xy > 1\}$, la traccia

$$\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ x^2 + y^2 \leq 1,\ x \leftarrow 2\ ),$$

e il $\mathcal{T}$-conflitto tra $x^2 + y^2 \leq 1$ e $x \leftarrow 2$. La soluzione è che la procedura di $\mathcal{T}$-soddisfacibilità per l'aritmetica sappia dedurre che $x^2 + y^2 \leq 1$ implica $-1 \leq x \wedge x \leq 1$, e che $-1 \leq x \wedge x \leq 1$ è inconsistente con $x = 2$. Si noti che $-1 \leq x \wedge x \leq 1$ è un interpolante di $x^2 + y^2 \leq 1$ e $x = 2$, perché x, che qui deve essere considerata una costante, appare in entrambi. Per questo esempio è sufficiente generare $(x^2 + y^2 \leq 1) \Rightarrow x \leq 1$, ovvero $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$ in forma clausale. MCSAT impara questa clausola come un $\mathcal{T}$-lemma e ritira l'assegnamento $x \leftarrow 2$. Poichè $x^2 + y^2 \leq 1$ è sulla traccia, l'apprendimento del $\mathcal{T}$-lemma porta MCSAT ad aggiornare la traccia aggiungendo il letterale implicato $x \leq 1$ con giustificazione $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$:

$$\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ x^2 + y^2 \leq 1,\ x \leq 1\ ).$$

L'aggiunta di $x \leq 1$ a $\mathcal{M}$ previene il sistema dal tentare assegnamenti come $x \leftarrow 3$, $x \leftarrow 4$, e così via, in quanto tutti incompatibili con $x \leq 1$. A questo punto, la procedura di $\mathcal{T}$-soddisfacibilità per l'aritmetica segnala che la sottosequenza $\{x \geq 2,\ x \leq 1\}$ è $\mathcal{T}$-inconsistente, generando la clausola in conflitto $\neg(x \geq 2) \vee \neg(x \leq 1)$. Questa clausola è di asserzione: il livello 0 di $\mathcal{M}$ contiene la sottosequenza $(x \geq 2,\ x \geq 1,\ y \geq 1)$; il primo livello contiene la sottosequenza $(x^2 + y^2 \leq 1,\ x \leq 1)$; e quindi il solo $\neg(x \leq 1)$ nella clausola in conflitto $\neg(x \geq 2) \vee \neg(x \leq 1)$ vede il suo complementare nel primo livello che è quello corrente. Applicando l'euristica 1UIP di CDCL, MCSAT impara questa clausola, e salta al livello 0 con

$$\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ \neg(x \leq 1)\ ).$$

Avendo MCSAT in precedenza imparato il $\mathcal{T}$-lemma $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$, la presenza di $\neg(x \leq 1)$ sulla traccia induce poi MCSAT ad aggiungere anche

$\neg(x^2 + y^2 \leq 1)$, come letterale implicato, con giustificazione $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$:

$$\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ \neg(x \leq 1),\ \neg(x^2 + y^2 \leq 1)\ ).$$

L'euristica 1UIP non è la sola possibile. Una diverso controllo non impara la clausola $\neg(x \geq 2) \vee \neg(x \leq 1)$ e procede invece a spiegare il conflitto che essa rappresenta con la risoluzione. Un primo passo di spiegazione per risoluzione tra la clausola in conflitto $\neg(x \geq 2) \vee \neg(x \leq 1)$ e la giustificazione $\neg(x^2 + y^2 \leq 1) \vee x \leq 1$ di $x \leq 1$ genera il risolvente $\neg(x^2 + y^2 \leq 1) \vee \neg(x \geq 2)$, che è ancora in conflitto. Un secondo passo di spiegazione per risoluzione tra la clausola in conflitto $\neg(x^2 + y^2 \leq 1) \vee \neg(x \geq 2)$ e la clausola unitaria $x \geq 2$ genera il risolvente $\neg(x^2 + y^2 \leq 1)$. MCSAT impara questa clausola, salta al livello 0, e mette $\neg(x^2 + y^2 \leq 1)$ sulla traccia:

$$\mathcal{M} = (\ x \geq 2,\ x \geq 1,\ y \geq 1,\ \neg(x^2 + y^2 \leq 1)\ ).$$

La traccia non contiene $\neg(x \leq 1)$, perché non avendo appreso la clausola $\neg(x \geq 2) \vee \neg(x \leq 1)$, non è necessario avere $\neg(x \leq 1)$ sulla traccia per soddisfarla. Con entrambe le euristiche il sistema rimedia alla decisione che ha causato il conflitto, ovvero l'aver tentato di soddisfare la clausola $x^2 + y^2 \leq 1 \vee xy > 1$ decidendo che $x^2 + y^2 \leq 1$ sia vero.

In conclusione, MCSAT è stato il primo sistema a generalizzare CDCL alla soddisfacibilità modulo una teoria, con inferenze al primo ordine per la spiegazione di conflitti, al di là della spiegazione mediante risoluzione proposizionale.

4 Metodi generali guidati da conflitti

In questa sezione illustriamo due strategie guidate da conflitti che generalizzano, rispettivamente, CDCL ed MCSAT. La Sezione 4.1 presenta le caratteristiche principali di un metodo chiamato SGGS, che "solleva" (dall'inglese "lifting") la procedura CDCL alla logica del primo ordine [36, 37, 38]. Al momento della stesura di questo capitolo SGGS è il solo metodo che realizzi una generalizzazione siffatta. Altri approcci [122, 1] sollevano la procedura CDCL solo al frammento di Bernays-Schönfinkel, che permette soltanto formule della forma $\exists^*\forall^*\varphi$, dove φ è priva di quantificatori e può contenere occorrenze di simboli di costante, ma non occorrenze di simboli di funzione. La *conflitto-risoluzione* [132, 91], dall'inglese "conflict resolution", solleva alla logica del primo ordine non l'intera procedura CDCL come qui descritta nella

Sezione 3.1.2, ma solo il meccanismo di generazione di risolventi guidato da conflitto, e senza affrontare il problema della rappresentazione di un modello candidato in logica del primo ordine. La Sezione 4.2 illustra per sommi capi un metodo chiamato CDSAT, che per primo generalizza MCSAT a *generiche* combinazioni of teorie [26, 27] Infine, la Sezione 4.3 considera come il problema della soddisfacibilità modulo teoria (SMT) possa essere generalizzato al problema della *soddisfacibilità modulo assegnamenti* (SMA).

4.1 SGGS

SGGS, dall'inglese "Semantically-Guided Goal-Sensitive reasoning", porta la deduzione guidata da conflitti alla logica del primo ordine [36, 37, 38]. È un metodo che gode di una combinazione di caratteristiche assai rara in dimostrazione automatica di teoremi: è simultaneamente *guidato da conflitti*, *guidato dalla semantica*, *sensibile al goal*, e *confluente rispetto alla dimostrazione*.

In questa sezione con "variabile" si intende variabile quantificata universalmente, e con "interpretazione" si intende *interpretazione di Herbrand*. Com'è noto, una volta che il problema è stato ridotto in forma clausale, è sufficiente considerare le interpretazioni di Herbrand [44]. Si ricorda che per soddisfare un insieme di clausole $\mathcal{S}$ bisogna soddisfare tutte le sue clausole; per soddisfare una clausola C bisogna soddisfare tutte le sue istanze senza variabili, perché le variabili sono implicitamente quantificate universalmente e il dominio delle interpretazioni di Herbrand è l'*universo di Herbrand*, cioè l'insieme dei termini senza variabili; e per soddisfare una clausola senza variabili bisogna soddisfarne almeno un letterale.

Se la segnatura contiene almeno un simbolo di funzione, gli atomi e le clausole hanno infinite istanze senza variabili, e ci sono infinite interpretazioni. In queste condizioni iniziare la costruzione di un modello decidendo il valore di verità degli atomi, come nel meccanismo di decisione in DPLL o CDCL, sarebbe un procedimento troppo cieco. Pertanto, SGGS adotta un'*interpretazione iniziale* $\mathcal{I}$ per averne una *guida semantica*. Per la rappresentazione dei modelli al primo ordine, SGGS mette sulla traccia $\mathcal{M}$ intere clausole. Dunque in SGGS la traccia è una sequenza di clausole. SGGS impone che ogni clausola in $\mathcal{M}$ abbia un *letterale selezionato*. Una traccia $\mathcal{M}$ rappresenta l'interpretazione $\mathcal{I}[\mathcal{M}]$, che è $\mathcal{I}$ modificata per soddisfare i letterali selezionati nelle clausole di $\mathcal{M}$.

Esempio 4.1: Supponiamo che $\mathcal{S}$ includa le clausole $su(a, b)$, $su(b, c)$, $verde(a)$, e $\neg verde(c)$, e $\mathcal{I}$ sia l'interpretazione *tutta negativa*, cioè quella

che rende veri tutti i letterali negativi e falsi tutti i letterali positivi. Allora SGGS inizia la derivazione con $\mathcal{M} = (\ su(a,b),\ su(b,c),\ verde(a)\ )$. In una clausola unitaria il solo letterale presente è ovviamente quello selezionato. $\mathcal{I}[\mathcal{M}]$ è l'interpretazione che rende tutti i letterali positivi falsi eccetto $su(a,b)$, $su(b,c)$, e $verde(a)$. Supponiamo invece che $\mathcal{I}$ sia l'interpretazione *tutta positiva*, cioè quella che rende veri tutti i letterali positivi e falsi tutti i letterali negativi. Allora SGGS inizia la derivazione con $\mathcal{M} = (\ \neg verde(c)\ )$, e $\mathcal{I}[\mathcal{M}]$ è l'interpretazione che rende tutti i letterali negativi falsi eccetto $\neg verde(c)$. Per confronto, visto che le clausole in questo esempio non hanno variabili, CDCL metterebbe tutte le clausole unitarie sulla traccia. SGGS assume un'interpretazione guida e la modifica solo quel tanto che basta, perché costruire modelli al primo ordine è molto più difficile che costruire modelli proposizionali.

SGGS generalizza la BCP di CDCL in una sorta di *propagazione clausale al primo ordine*. La BCP di CDCL si basa sulla simmetria dei valori di verità in logica proposizionale: se L è vero, $\neg L$ è falso, e se L è falso, $\neg L$ è vero. Dal momento che le variabili in letterali di clausole al primo ordine sono implicitamente quantificate universalmente, se L è vero, $\neg L$ è falso, ma se L è falso, sappiamo soltanto che almeno un'istanza senza variabili di L è falsa, cioè che almeno un'istanza senza variabili di $\neg L$ è vera. Per colmare questa discrepanza, SGGS introduce la *falsità uniforme*: un letterale al primo ordine è *uniformemente falso*, se *tutte* le sue istanze senza variabili sono false. Questa nozione più forte di falsità restaura la simmetria: se L è vero, $\neg L$ è uniformemente falso, e se L è uniformemente falso, $\neg L$ è vero.

SGGS richiede che ogni letterale di ogni clausola in $\mathcal{M}$ sia o $\mathcal{I}$*-vero*, cioè vero in $\mathcal{I}$, o $\mathcal{I}$*-falso*, dove, per definizione, essere $\mathcal{I}$*-falso* vuol dire essere uniformemente falso in $\mathcal{I}$. In questo modo, ogni letterale sulla traccia rappresenta il valore di verità in $\mathcal{I}$ di tutte le sue istanze senza variabili. SGGS impone inoltre che ogni clausola C in $\mathcal{M}$ abbia un *letterale selezionato* L: la clausola C con L selezionato si scrive $C[L]$. I letterali $\mathcal{I}$-falsi sono preferiti per la selezione. Un letterale $\mathcal{I}$-vero viene selezionato solo in una clausola i cui letterali siano tutti $\mathcal{I}$-veri; una clausola siffatta è detta clausola *tutti-*$\mathcal{I}$*-veri*. SGGS mira a costruire un modello di $\mathcal{S}$: se $\mathcal{I} \models \mathcal{S}$, la ricerca termina immediatamente; se $\mathcal{I} \not\models \mathcal{S}$, SGGS tenta di costruire un'interpretazione $\mathcal{I}[\mathcal{M}]$ che differisce da $\mathcal{I}$ dove occorre per soddisfare $\mathcal{S}$. Da qui discende la preferenza per i letterali $\mathcal{I}$-falsi nella selezione.

Esempio 4.2: $\mathcal{S} = \{R(x, f(x)),\ \neg R(x,x),\ \neg R(x,y) \vee R(y,x)\}$ assiomatizza una relazione di raggiungibilità R che è irriflessiva, simmetrica, e tale che ogni stato x abbia un successore $f(x)$. Data $\mathcal{I}$ tutta negativa, SGGS costruisce la

sequenza

$$\mathcal{M} = (\ [R(x, f(x))],\ \neg R(x, f(x)) \vee [R(f(x), x)]\ ),$$

dove le parentesi quadre denotano selezione. Nella seconda clausola, che ha due letterali e quindi permette una scelta nella selezione, il letterale positivo viene preferito, perché è $\mathcal{I}$-falso. Quindi SGGS termina perché $\mathcal{I}[\mathcal{M}] \models \mathcal{S}$.

In SGGS una clausola è *in conflitto* se tutti i suoi letterali sono uniformemente falsi in $\mathcal{I}[\mathcal{M}]$. Un letterale L è uniformemente falso in $\mathcal{I}[\mathcal{M}]$, se tutte le sue istanze senza variabili appaiono negate tra quelle che un precedente letterale selezionato N rende vere in $\mathcal{I}[\mathcal{M}]$. In questo senso, L *dipende* da N.

Esempio 4.3: Dati $\mathcal{S} = \{P(x),\ \neg P(x) \vee R(a, x),\ \neg P(x) \vee \neg R(x, b)\}$ e $\mathcal{I}$ tutta negativa, SGGS costruisce la sequenza

$$\mathcal{M} = (\ [P(x)],\ \neg P(x) \vee [R(a, x)],\ \neg P(a) \vee [\neg R(a, b)]\ ),$$

dove i letterali $\neg P(x)$ e $\neg P(a)$ sono uniformemente falsi in $\mathcal{I}[\mathcal{M}]$, perché $P(x)$ è selezionato; il letterale $\neg R(a, b)$ è uniformemente falso in $\mathcal{I}[\mathcal{M}]$, perché $R(a, x)$ è selezionato; e l'ultima clausola in $\mathcal{M}$ è in conflitto con $\mathcal{I}[\mathcal{M}]$. Si noti che questa clausola è una clausola tutti-$\mathcal{I}$-veri.

In SGGS un letterale L è *implicato*, con la clausola C come *giustificazione*, se L è il solo letterale di C che non è uniformemente falso in $\mathcal{I}[\mathcal{M}]$. SGGS assicura che ogni clausola tutti-$\mathcal{I}$-veri in $\mathcal{M}$ sia o una clausola in conflitto o la giustificazione del suo letterale selezionato. A questo scopo, SGGS usa *funzioni di assegnamento* per tenere traccia delle dipendenze di letterali $\mathcal{I}$-veri da letterali selezionati $\mathcal{I}$-falsi: se L dipende da N, la funzione di assegnamento assegna L alla clausola di N. Dunque, una clausola tutti-$\mathcal{I}$-veri i cui letterali siano tutti assegnati a clausole con letterali selezionati $\mathcal{I}$-falsi è una clausola in conflitto; una clausola tutti-$\mathcal{I}$-veri i cui letterali siano tutti assegnati, eccetto quello selezionato, è una giustificazione del suo letterale selezionato.

Esempio 4.4: Continuando l'esempio 4.3, i letterali $\neg P(x)$ e $\neg P(a)$ sono assegnati a $[P(x)]$; il letterale $\neg R(a, b)$ è assegnato a $[R(a, x)]$; e l'ultima clausola in $\mathcal{M}$ è in conflitto con $\mathcal{I}[\mathcal{M}]$ siccome tutti i suoi letterali sono assegnati.

Tutte le sequenze di clausole SGGS negli esempi precedenti sono generate da applicazioni della regola di inferenza di *SGGS-estensione*. Questa regola aggiunge alla sequenza corrente un'istanza E di una clausola $C \in \mathcal{S}$ e seleziona uno dei suoi letterali. Questa selezione è l'analogo in logica del primo ordine della decisione in logica proposizionale. L'istanza E viene costruita in modo da catturare le istanze senza variabili di C per cui $\mathcal{I}[\mathcal{M}] \not\models C$, così che aggiungere E alla traccia le soddisferà.

Esempio 4.5: Nell'esempio 4.2, la clausola $\neg R(x, f(x)) \vee [R(f(x), x)]$ è un'istanza della clausola in ingresso $\neg R(x, y) \vee R(y, x)$. SGGS la genera unificando il letterale $\neg R(x, y)$ della clausola in ingresso con il letterale $[R(x, f(x))]$ già selezionato. Si ricordi che ogni clausola ha le sue variabili. Ridenominiamo le variabili della clausola in ingresso: $\neg R(u, v) \vee R(v, u)$. Quindi l'unificatore più generale applicato è $\alpha = \{u \leftarrow x, v \leftarrow f(x)\}$. Il significato dell'operazione è il seguente. Inzialmente, siccome $\mathcal{I}$ è tutta negativa, la seconda e la terza clausola di $\mathcal{S}$ sono soddisfatte da $\mathcal{I}$, ma la prima non lo è. Dunque, SGGS genera $\mathcal{M} = ([R(x, f(x))])$ mediante un passo di SGGS-estensione con unificatore più generale vuoto. A questo punto, $\mathcal{I}[\mathcal{M}]$ soddisfa la prima e la seconda clausola di $\mathcal{S}$, ma non la terza. Quali istanze senza variabili della terza clausola abbiamo perso? Precisamente quelle dove $\neg R(u, v)$ unifica con $[R(x, f(x))]$. Pertanto, SGGS estende il modello per ricatturare queste istanze aggiungendo $\neg R(x, f(x)) \vee [R(f(x), x)]$.

Tuttavia, non si dà sempre il caso che una SGGS-estensione aggiunga una clausola E il cui letterale selezionato estende $\mathcal{I}[\mathcal{M}]$, perché E può essere una clausola in conflitto. In tal caso, SGGS *spiega* il conflitto mediante una forma molto ristretta di risoluzione al primo ordine [125], detta *SGGS-risoluzione*. L'SGGS-risoluzione risolve un letterale $\mathcal{I}$-falso L in E con il letterale $\mathcal{I}$-vero implicato N, la cui selezione in $\mathcal{M}$ rende L uniformemente falso in $\mathcal{I}[\mathcal{M}]$. Quindi, l'SGGS-risoluzione risolve la clausola in conflitto E con la clausola tutti-$\mathcal{I}$-veri D che è la giustificazione di N in $\mathcal{M}$. Il risolvente è ancora in conflitto. Questa serie di *inferenze di spiegazione* mediante SGGS-risoluzione termina quando o la clausola vuota $\square$ o una clausola in conflitto tutti-$\mathcal{I}$-veri viene generata.

La generazione di $\square$ segnala che si è completata una refutazione: l'insieme in ingresso $\mathcal{S}$ è insoddisfacibile. Altrimenti, SGGS *muove* la clausola in conflitto tutti-$\mathcal{I}$-veri, poniamo $E[L]$, alla sinistra della clausola $D[N]$ il cui letterale $\mathcal{I}$-falso selezionato N rende il letterale $\mathcal{I}$-vero L selezionato in E uniformemente falso in $\mathcal{I}[\mathcal{M}]$. L'effetto di questa regola di inferenza, detta *SGGS-mossa*, è quello di *imparare* $E[L]$ e *risolvere* il conflitto, *rovesciando* in un colpo solo il valore di verità di *tutte* le istanze senza variabili di L. A questo punto, $D[N]$ è in conflitto, così che la SGGS-risoluzione interviene a risolvere $E[L]$ e $D[N]$ su L ed N.

Ci sono tuttavia due considerazioni da fare. La prima è la seguente: poiché $E[L]$ è una clausola in conflitto tutti-$\mathcal{I}$-veri, i suoi letterali sono tutti assegnati a clausole con letterali selezionati $\mathcal{I}$-falsi. Cosa succede a questi assegnamenti quando SGGS muove $E[L]$? SGGS prescrive che in una clausola in conflitto tutti-$\mathcal{I}$-veri, si selezioni il letterale assegnato alla clausola più a destra, ovvero a quella di indice massimo. In questo modo, quando la clausola muove verso

sinistra per risolvere il conflitto, il suo solo letterale non più assegnato è quello selezionato, e la clausola cambia stato da clausola in conflitto a giustificazione di un letterale implicato, il suo letterale selezionato. La seconda considerazione è che SGGS può *partizionare* $D[N]$ con $E[L]$ per evitare di cambiare $\mathcal{I}[\mathcal{M}]$ là dove non è necessario. La filosofia di SGGS è di essere *guidato da conflitti* e cambiare $\mathcal{I}[\mathcal{M}]$ solo quel tanto che basta per risolvere il conflitto. Entrambe le considerazioni sono illustrate nell'esempio seguente.

Esempio 4.6: Continuando l'esempio 4.4, eravamo rimasti con

$$\mathcal{M} = (\ [P(x)],\ \neg P(x) \vee [R(a,x)],\ \neg P(a) \vee [\neg R(a,b)]\ ).$$

Ora possiamo capire perché $\neg R(a,b)$ sia selezionato nella clausola in conflitto $\neg P(a) \vee [\neg R(a,b)]$: $\neg P(a)$ è assegnato alla prima clausola; $\neg R(a,b)$ è assegnato alla seconda clausola, e quindi $\neg R(a,b)$ è selezionato. La mossa per risolvere il conflitto sembrerebbe consistere nel muovere $\neg P(a) \vee [\neg R(a,b)]$ alla sinistra di $\neg P(x) \vee [R(a,x)]$. Tuttavia, SGGS non lo fa, perché cambiare il valore di verità di tutte le istanze senza variabili di $[R(a,x)]$ è troppo per soddisfare $[\neg R(a,b)]$. SGGS partiziona $\neg P(x) \vee [R(a,x)]$ con $\neg P(a) \vee [\neg R(a,b)]$ producendo

$$\mathcal{M} = (\ [P(x)],\ x \not\equiv b \rhd \neg P(x) \vee [R(a,x)],\ \neg P(b) \vee [R(a,b)],\ \neg P(a) \vee [\neg R(a,b)]\ ).$$

Poi, un'applicazione di SGGS-mossa produce la sequenza

$$\mathcal{M} = (\ [P(x)],\ x \not\equiv b \rhd \neg P(x) \vee [R(a,x)],\ \neg P(a) \vee [\neg R(a,b)],\ \neg P(b) \vee [R(a,b)]\ ).$$

Ora un passo di SGGS-risoluzione risolve $\neg P(a) \vee [\neg R(a,b)]$ e $\neg P(b) \vee [R(a,b)]$ generando

$$\mathcal{M} = (\ [P(x)],\ x \not\equiv b \rhd \neg P(x) \vee [R(a,x)],\ \neg P(a) \vee [\neg R(a,b)],\ \neg P(b) \vee [\neg P(a)]\ ),$$

dove il risolvente $\neg P(b) \vee [\neg P(a)]$ è un'altra clausola tutti-$\mathcal{I}$-veri in conflitto. In questo caso la selezione di $\neg P(a)$ è arbitraria, dal momento che sia $\neg P(b)$ che $\neg P(a)$ sono assegnati a $[P(x)]$.

Come mostrato nell'esempio precedente, la regola di SGGS-risoluzione prescrive che il risolvente *rimpiazzi* la premessa del passo di SGGS-risoluzione che non è tutti-$\mathcal{I}$-veri. Tutte le clausole che hanno letterali assegnati alla premessa rimpiazzata vengono eliminate. In altre parole, il risolvente rimpiazza la clausola in conflitto, non la giustificazione, proprio come in CDCL.

L'operazione di partizionare una clausola $D[N]$ con una clausola $E[L]$ rimpiazza $D[N]$ con una *partizione*, $D_1[M_1], \ldots, D_n[M_n]$, cioè una sequenza

di clausole che insieme rappresentano le stesse istanze senza variabili di $D[N]$, ma hanno letterali selezionati *disgiunti*. Due letterali si dicono *disgiunti* se i loro atomi non hanno istanze senza variabili in comune. Inoltre, esiste un j, $1 \leq j \leq n$, tale che l'insieme delle istanze senza variabili di $atom(L)$ è uguale all'insieme delle istanze senza variabili di $atom(M_j)$, dove $atom(L)$ denota l'atomo del letterale L. In altre parole, partizionare $D[N]$ con $E[L]$ frammenta $D[N]$ in modo tale da esporre l'intersezione non vuota tra l'insieme delle istanze senza variabili di L e l'insieme delle istanze senza variabili di N, dove l'intersezione è calcolata sugli atomi e quindi ignora il segno. L'operazione di partizionare clausole introduce *vincoli* sulle clausole nella traccia, e per questo SGGS lavora con una traccia di clausole possibilmente vincolate con *vincoli di Herbrand* [35, 38].

Esempio 4.7: Continuando l'esempio 4.6, la clausola $\neg P(b) \vee [\neg P(a)]$ partiziona la clausola $[P(x)]$, producendo

$$\mathcal{M} = (x \not\equiv a \rhd [P(x)],\ [P(a)],\ \neg P(a) \vee [\neg R(a, b)],\ \neg P(b) \vee [\neg P(a)]),$$

dove $x \not\equiv b \rhd \neg P(x) \vee [R(a, x)]$ è stata eliminata: SGGS permette di eliminare una clausola che ha un letterale (qui $\neg P(x)$) assegnato a una clausola (qui $[P(x)]$) che viene partizionata. L'alternativa è partizionare ricorsivamente $x \not\equiv b \rhd \neg P(x) \vee [R(a, x)]$ in $\neg P(a) \vee [R(a, a)]$ e $x \not\equiv b, x \not\equiv a \rhd \neg P(x) \vee [R(a, x)]$, assegnando $\neg P(a)$ a $[P(a)]$ e $x \not\equiv b, x \not\equiv a \rhd \neg P(x)$ a $x \not\equiv a \rhd [P(x)]$. Si noti che $x \not\equiv b \rhd \neg P(x) \vee [R(a, x)]$ non può semplicemente rimanere in $\mathcal{M}$, perché non c'è più una clausola a cui assegnare $x \not\equiv b \rhd \neg P(x)$ dopo che $[P(x)]$ è stata partizionata. Con una SGGS-mossa si ottiene

$$\mathcal{M} = (x \not\equiv a \rhd [P(x)],\ \neg P(b) \vee [\neg P(a)],\ [P(a)],\ \neg P(a) \vee [\neg R(a, b)]).$$

Poi un'inferenza di SGGS-risoluzione risolve $\neg P(b) \vee [\neg P(a)]$ e $[P(a)]$ generando

$$\mathcal{M} = (\ x \not\equiv a \rhd [P(x)],\ \neg P(b) \vee [\neg P(a)],\ [\neg P(b)]\ ),$$

dove $\neg P(a) \vee [\neg R(a, b)]$ è stata eliminata, perché il suo letterale $\neg P(a)$ era assegnato alla premessa del passo di risoluzione che è stata rimpiazzata.

Un'altra ragione per eliminare $\neg P(a) \vee [\neg R(a, b)]$ in questo esempio è che è una clausola *disponibile*: in SGGS una clausola C in $\mathcal{M} = QCQ'$, dove Q e Q' sono sottosequenze di $\mathcal{M}$, è *disponibile*, se è soddisfatta da $\mathcal{I}[Q]$, cioè dall'interpretazione indotta dalle clausole alla sua sinistra. La clausola C è ridondante visto che le clausole che la precedono sulla traccia sono sufficienti

a costruire un modello che soddisfa C. SGGS ha una regola di inferenza, detta *SGGS-eliminazione*, che elimina tutte le clausole disponibili. Questa regola viene applicata con la priorità più alta per contenere la lunghezza della sequenza di clausole, data la ben nota prolificità della deduzione automatica in logica del primo ordine.

Esempio 4.8: Continuando l'esempio 4.7, la clausola $[\neg P(b)]$ partiziona la clausola $x \not\equiv a \rhd [P(x)]$, generando

$$\mathcal{M} = (\ x \not\equiv a, x \not\equiv b \rhd [P(x)],\ [P(b)],\ \neg P(b) \vee [\neg P(a)],\ [\neg P(b)]\ ).$$

Con un'applicazione di SGGS-mossa si ottiene

$$\mathcal{M} = (\ x \not\equiv a, x \not\equiv b \rhd [P(x)],\ [\neg P(b)],\ [P(b)],\ \neg P(b) \vee [\neg P(a)]\ ).$$

Poi un passo di SGGS-risoluzione produce

$$\mathcal{M} = (\ x \not\equiv a, x \not\equiv b \rhd [P(x)],\ [\neg P(b)],\ \square,\ \neg P(b) \vee [\neg P(a)]\ )$$

e chiude la refutazione.

In SGGS, l'equità del piano di ricerca assicura che inferenze che sono infinitamente possibili non vengano neglette, e che ogni conflitto sia risolto prima di altre applicazioni di SGGS-estensione. SGGS è *refutazionalmente completo* e *completo al limite* per la costruzione di modelli [38]. La prima proprietà significa che se l'insieme in ingresso $\mathcal{S}$ è insoddisfacibile, ogni derivazione generata da una strategia SGGS con un piano di ricerca equo è una refutazione. La seconda proprietà significa che se $\mathcal{S}$ è soddisfacibile, la *sequenza-limite* $\mathcal{M}_\infty$ di ogni derivazione generata da una strategia SGGS con un piano di ricerca equo rappresenta un modello di $\mathcal{S}$. In altre parole, $\mathcal{I}[\mathcal{M}_\infty]$ è un modello di $\mathcal{S}$. Naturalmente, data la semidecidibilità della logica del primo ordine, sia la sequenza-limite che la derivazione possono essere infinite. Entrambi i risultati di completezza valgono per qualsiasi interpretazione iniziale $\mathcal{I}$.

Dato un insieme di clausole $\mathcal{S}$ ottenuto trasformando in forma clausale $H \cup \{\neg\varphi\}$ (cfr. le Sezioni 1 e 2), le clausole ottenute dalla trasformazione di $\neg\varphi$ si dicono *clausole goal*. Una strategia di dimostrazione di teoremi si dice *sensibile al goal*, se genera solo clausole discendenti dalle clausole goal [123]. SGGS è flessibile rispetto a questa caratteristica: è sensibile al goal, se l'interpretazione iniziale $\mathcal{I}$ soddisfa le clausole provenienti dalla trasformazione di H in forma clausale, ma non le clausole goal.

Una strategia di dimostrazione di teoremi si dice *confluente rispetto alla dimostrazione*, se garantisce che si possa completare la dimostrazione senza

mai dover disfare un passo di inferenza (cfr. la Sezione 2). In altre parole, una strategia è confluente rispetto alla dimostrazione, se può andare sempre avanti senza aver bisogno di tornare indietro ("backtracking") o saltare all'indietro. SGGS è confluente rispetto alla dimostrazione, perché esce dai conflitti muovendo una clausola in $\mathcal{M}$, senza necessità di disfare passi di inferenza. Questo aspetto differenzia SGGS da CDCL e suggerisce che una ricerca con "backtracking" o salto all'indietro non sia un ingrediente essenziale della deduzione guidata da conflitti.

4.2 CDSAT

CDSAT, dall'inglese "Conflict-Driven Satisfiability", è un metodo guidato da conflitti [26, 27] che generalizza sia MCSAT (cfr. la Sezione 3.2.2) che il metodo di condivisione di uguaglianze (cfr. la Sezione 3.1.4). CDSAT estende MCSAT a *generiche* combinazioni di teorie *disgiunte* e si riduce a MCSAT in presenza di una sola teoria. CDSAT generalizza il metodo di condivisione di uguaglianze, perché permette di combinare teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$ dotate di procedure di $\mathcal{T}_i$-soddisfacibilità guidate da conflitti, facendo in modo che la risultante procedura di decisione per $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$ sia anch'essa guidata da conflitti. CDSAT si riduce al metodo di condivisione di uguaglianze se nessuna teoria $\mathcal{T}_i$ ha una procedura di $\mathcal{T}_i$-soddisfacibilità guidata da conflitti.

Il metodo di condivisione di uguaglianze combina procedure di $\mathcal{T}_i$-soddisfacibilità trattandole come scatole chiuse che comunicano con l'esterno solo propagando disgiunzioni di uguaglianze. Al contrario una procedura di $\mathcal{T}_i$-soddisfacibilità guidata da conflitti lavora sulla traccia alla pari con CDCL come si è visto in MCSAT. Dunque CDSAT presenta un paradigma più generale di combinazione di teorie che permette di integrare sia procedure di $\mathcal{T}_i$-soddisfacibilità guidate da conflitti che procedure di $\mathcal{T}_i$-soddisfacibilità che non lo sono. CDSAT raggiunge questo risultato trattando la combinazione di teorie come *combinazione di sistemi di inferenza* invece che come combinazione di procedure di $\mathcal{T}_i$-soddisfacibilità. Una procedura di $\mathcal{T}_i$-soddisfacibilità, come ogni strategia di deduzione automatica, è data da un insieme di inferenza e un piano di ricerca (cfr. la Sezione 1). Se tutte le procedure di $\mathcal{T}_i$-soddisfacibilità hanno lo stesso piano di ricerca guidato da conflitti (quello di CDCL), quel che occorre far cooperare sono i sistemi di inferenza, mentre il comune controllo guidato da conflitti viene gestito centralmente da CDSAT. Di CDSAT si sono dimostrate correttezza, completezza, e terminazione, determinando i requisiti che le teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$ e i loro sistemi di inferenza devono soddisfare per assicurare queste proprietà [26, 27]

CDSAT porta alle estreme conseguenze la filosofia di MCSAT di rimuovere la gerarchia stabilita da DPLL($\mathcal{T}$) per cui la logica proposizionale e CDCL stanno al centro mentre le teorie e le loro procedure sono satelliti. In CDSAT, anche la logica proposizionale è una teoria e la procedura CDCL è una delle procedure di $\mathcal{T}_i$-soddisfacibilità guidate da conflitti che collaborano a costruire un modello sulla traccia. Coerentemente, CDSAT tratta atomi, letterali, clausole e formule come termini di tipo prop (da proposizione). Il tipo prop deve essere incluso nella segnatura di tutte le teorie considerate.

Come in MCSAT, l'asserzione di un letterale L sulla traccia è vista come l'assegnamento $L \leftarrow \mathit{vero}$, assegnamenti quali $L \leftarrow \mathit{vero}$ e $x \leftarrow 3$, dove x è una variabile libera del primo ordine, sono trattati in modo completamente uniforme, e la traccia $\mathcal{M}$ è definita come una sequenza di assegnamenti. Naturalmente, $L \leftarrow \mathit{vero}$ e $L \leftarrow \mathit{falso}$ si abbreviano L e $\neg L$. CDSAT generalizza la nozione di assegnamento permettendo assegnamenti anche a termini che non sono variabili, come $L \vee N \leftarrow \mathit{vero}$ o $(x \cdot y) \leftarrow 3$, dove $\cdot$ è la moltiplicazione in aritmetica. Dunque, anche il problema in ingresso è visto come un insieme di assegnamenti: un insieme di clausole $\mathcal{S} = \{C_1, \ldots, C_m\}$ è letto come l'assegnamento $\{C_1 \leftarrow \mathit{vero}, \ldots, C_m \leftarrow \mathit{vero}\}$.

Viene spontaneo scrivere assegnamenti come $x \leftarrow 3$, ma 3 non è necessariamente un simbolo nella segnatura della teoria. Quindi, CDSAT assume di avere per ogni teoria un'*estensione* che aggiunge alla segnatura tanti simboli di costante nuovi quanti sono necessari per nominare tutti gli elementi nel dominio di interpretazione (e.g., tutti gli interi). I simboli c e q già usati negli esempi 3.16 e 3.15 sono siffatti simboli di costante. Di tale estensione si richiede che sia *conservativa*, in modo che non cambi il problema. Un'estensione $\mathcal{T}_k^+$ di una teoria $\mathcal{T}_k$ è *conservativa*, se ogni insieme $\mathcal{S}$ di $\mathcal{T}_k$-clausole che è $\mathcal{T}_k^+$-insoddisfacibile è anche $\mathcal{T}_k$-insoddisfacibile. Così, se CDSAT scopre che $\mathcal{S}$ è $\mathcal{T}_k^+$-insoddisfacibile, sappiamo che $\mathcal{S}$ è $\mathcal{T}_k$-insoddisfacibile; se $\mathcal{S}$ è $\mathcal{T}_k$-soddisfacibile, esiste un $\mathcal{T}_k^+$-modello che CDSAT può costruire.

La combinazione delle teorie $\mathcal{T}_1, \ldots, \mathcal{T}_n$ viene realizzata in CDSAT dalla cooperazione di $\mathcal{T}_k$-*moduli* $I_1, \ldots, I_n$, uno per ogni teoria. Un $\mathcal{T}_k$-modulo è un sistema di inferenza per la teoria $\mathcal{T}_k$ che lavora su assegnamenti: un passo di inferenza deduce un assegnamento Booleano, cioè $L \leftarrow \mathit{vero}$ o $L \leftarrow \mathit{falso}$, dove L è un qualsiasi termine di tipo prop, da un insieme di assegnamenti di qualsiasi tipo. Gli assegnamenti di valori altri dai valori di verità non sono visti in CDSAT come il risultato di inferenze, ma sempre come decisioni, anche quando sono forzati dal contenuto della traccia: per esempio, se $\mathcal{M}$ contiene $x \leftarrow 1$ e $(x + y) \leftarrow 3$, l'assegnamento $y \leftarrow 2$ è l'unico possibile. $\mathcal{T}_k$-moduli per la logica proposizionale, e per i frammenti senza quantificatori della teoria dell'uguaglianza, dell'aritmetica lineare sui razionali, e della teoria

degli array con estensionalità sono stati definiti per istanziare le operazioni di CDSAT a casi concreti [26, 27]. Ogni $\mathcal{T}_k$-modulo I_k è responsabile per decidere assegnamenti, per dedurre assegnamenti da quelli che sono sulla traccia, per riconoscere e spiegare $\mathcal{T}_k$-conflitti.

Tutti i $\mathcal{T}_k$-moduli contribuiscono a trasformare la traccia $\mathcal{M}$. Poichè le teorie hanno segnature diverse che si mescolano nei termini che compaiono in $\mathcal{M}$, ogni $\mathcal{T}_k$-modulo ha la sua $\mathcal{T}_k$-*vista* di $\mathcal{M}$. CDSAT sviluppa ulteriormente l'intuizione, già presente in MCSAT ed SGGS, che l'essenza della deduzione guidata da conflitti sia la *spiegazione* dei conflitti. In CDSAT un conflitto è un insieme di assegnamenti. Al fine di poter spiegare i conflitti, ogni assegnamento A sulla traccia $\mathcal{M}$ che non sia una decisione è associato a un insieme di assegnamenti J che appaiono prima di A in $\mathcal{M}$ e dai quali A è stato dedotto: l'insieme J è detto la *giustificazione* di A. La nozione di giustificazione in CDSAT generalizza quella di CDCL. Quindi, se A diventa parte di un conflitto, può essere spiegato rimpiazzandolo con J. La risoluzione proposizionale che spiega i conflitti in CDCL è un caso particolare di questo meccanismo, come illustrato nell'esempio seguente.

Esempio 4.9: Riprendiamo il problema degli esempi 3.2, 3.3, 3.4, 3.5 e 3.6, con l'insieme di clausole $\mathcal{S} = \{\neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b\}$. CDSAT inizia con

$$\mathcal{M} = (\ \neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b\ ),$$

dove tutti questi assegnamenti appartengono al livello 0 e non hanno giustificazione. La decisione che a sia vero e la deduzione che b sia vero in conseguenza portano CDSAT a aggiungere il primo livello ($a,\ b$):

$$\mathcal{M} = (\ \neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b,\ a,\ b\ ),$$

dove a non ha giustificazione perché è una decisione, mentre b ha giustificazione $\{a,\ \neg a \vee b\}$. La decisione che c sia vero e la deduzione che d sia vero in conseguenza portano CDSAT a aggiungere il secondo livello ($c,\ d$):

$$\mathcal{M} = (\ \neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b,\ a,\ b,\ c,\ d\ ),$$

dove c non ha giustificazione perché è una decisione, mentre d ha giustificazione $\{c,\ \neg c \vee d\}$. La decisione che e sia vero e la deduzione che $\neg f$ sia vero in conseguenza portano CDSAT a aggiungere il terzo livello ($e,\ \neg f$):

$$\mathcal{M} = (\ \neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b,\ a,\ b,\ c,\ d,\ e,\ \neg f\ ),$$

dove e non ha giustificazione perché è una decisione, mentre $\neg f$ ha giustificazione $\{e,\ \neg e \vee \neg f\}$. A questo punto il conflitto è dato dall'insieme

$$E_1 = \{f \vee \neg e \vee \neg b,\ \neg f,\ e,\ b\},$$

dove $f \vee \neg e \vee \neg b$ viene dal livello 0, $\neg f$ ed e vengono dal terzo livello, e b viene dal primo livello, per cui il livello di E_1 è il terzo. Il livello di un conflitto è definito come il massimo tra i livelli dei suoi elementi. Un passo di spiegazione sostituisce $\neg f$ con la sua giustificazione $\{e,\ \neg e \vee \neg f\}$, così che ora il conflitto ha la forma

$$E_2 = \{f \vee \neg e \vee \neg b,\ e,\ \neg e \vee \neg f,\ b\},$$

dove $f \vee \neg e \vee \neg b$ e $\neg e \vee \neg f$ vengono dal livello 0, e viene dal terzo livello, e b viene dal primo livello, per cui il livello di E_2 è ancora il terzo. CDSAT incorpora l'euristica 1UIP riformulandola in modo che si applichi quando il conflitto contiene un solo elemento che appartiene allo stesso livello del conflitto stesso. Quindi a questo punto si applica l'euristica 1UIP, perché E_2 contiene un solo elemento, ovvero e, che appartiene al terzo livello. CDSAT salta all'indietro al livello dell'insieme

$$E_3 = E_2 \setminus \{e\} = \{f \vee \neg e \vee \neg b,\ \neg e \vee \neg f,\ b\},$$

cioè il conflitto E_2 tolto e. Poiché il livello di E_3 è il primo, CDSAT salta all'indietro al primo livello, e risolve il conflitto mettendo sulla traccia $\neg e$:

$$\mathcal{M} = (\ \neg a \vee b,\ \neg c \vee d,\ \neg e \vee \neg f,\ f \vee \neg e \vee \neg b,\ a,\ b,\ \neg e\ ),$$

dove $\neg e$ ha come giustificazione E_3. CDSAT non è stato ancora equipaggiato con l'apprendimento: per questo la spiegazione del conflitto si è svolta senza imparare il lemma $\neg b \vee \neg e$, e quindi la giustificazione di $\neg e$ non è il lemma $\neg b \vee \neg e$, ma l'insieme E_3, da cui $\neg e$ si può derivare con due passi di risoluzione.

Esempi in combinazioni di teorie sono disponibili nei riferimenti bibliografici [26, 27]. CDSAT è stato dimostrato *corretto* e *completo* per combinazioni di teorie *disgiunte*, assumendo che almeno uno dei $\mathcal{T}_k$-moduli, poniamo I_1, abbia informazioni sulle cardinalità dei dominii di interpretazione dei tipi condivisi. Assumere che tutte le teorie siano stabilmente infinite è un modo particolare di soddisfare questo requisito. Il sistema di inferenza di CDSAT è parametrizzato rispetto a una *base globale*, che è la fonte dei termini nuovi che i $\mathcal{T}_k$-moduli possono introdurre con le loro inferenze. L'aggettivo *globale* suggerisce che questa base è comune a tutte le teorie, e non è semplicemente l'unione di

basi locali alle teorie [26, 27]. Come per MCSAT, la finitezza di questa base assicura la terminazione.

Poiché non c'è ragione di restringere CDSAT a ingressi della forma $\{C_1 \leftarrow \mathit{vero}, \ldots, C_m \leftarrow \mathit{vero}\}$, CDSAT accetta in ingresso problemi contenenti sia assegnamenti di valori di verità che assegnamenti a termini del primo ordine. Per esempio, si può dover decidere la soddisfacibilità di una formula senza quantificatori φ in una combinazione di teorie, dati assegnamenti di certi valori ad alcune delle variabili libere o atomi proposizionali che appaiano in φ. Pertanto, CDSAT risolve problemi in una categoria più generale di SMT che chiamiamo SMA, per *soddisfacibilità modulo assegnamenti*. Per problemi di questo tipo, già la stesura del problema in ingresso presuppone le estensioni conservative delle teorie.

4.3 Soddisfacibilità modulo assegnamenti

Durante la ricerca di un modello, un sistema deduttivo guidato da conflitti mantiene un modello parziale candidato rappresentato da un assegnamento. Questo suggerisce, anche indipendentemente da CDSAT (cfr. la Sezione 4.2), il problema della *soddisfacibilità modulo assegnamenti* (SMA), definito come il problema di decidere la soddisfacibilità di un insieme $\mathcal{S}$ di clausole, modulo una teoria $\mathcal{T}$, rispetto a un assegnamento iniziale J ad alcuni dei termini che compaiono in $\mathcal{S}$, indifferentemente proposizionali o del primo ordine. Se J è vuoto, SMA si riduce a SMT; se J e $\mathcal{T}$ sono entrambe vuote, SMA si riduce a SAT, mentre uno stato intermedio di una ricerca SAT o SMT è un'istanza di SMA. Come si è visto, in CDSAT non c'è distinzione tra $\mathcal{S}$ e J, uniti a formare l'assegnamento in ingresso.

La risposta a un problema di soddisfacibilità modulo assegnamenti è o "soddisfacibile" con un modello di $\mathcal{S}$ che estende J, o "insoddisfacibile" con un insieme di clausole $\mathcal{F}$ che è conseguenza logica di $\mathcal{S}$ ed è falso in J. L'insieme $\mathcal{F}$ è una *spiegazione*, perché spiega come mai $\mathcal{S}$ è insoddisfacibile dato J. Il concetto di *spiegazione* generalizza quello di *nucleo insoddisfacibile*. Nei problemi di soddisfacibilità proposizionale (SAT), un *nucleo insoddisfacibile* di $\mathcal{S}$ è un insieme di clausole che segue logicamente da $\mathcal{S}$ ed è insoddisfacibile. Un nucleo insoddisfacibile spiega perché $\mathcal{S}$ è insoddisfacibile, e più è piccolo rispetto all'ordinamento dato dalla relazione di sottoinsieme $\subseteq$, più è considerato preciso. Come già discusso, il concetto di *spiegazione* generalizza anche quello di *interpolante* (cfr. la Sezione 3.2.2).

La soddisfacibilità modulo assegnamenti appare in svariati contesti, come i problemi di *enumerazione* di modelli, il progetto di SAT-solutori paralleli, e

i problemi di *ottimizzazione*. A volte non si vuole semplicemente trovare un modello, ma *enumerare* i modelli di un insieme di clausole. Un approccio a questo problema è risolvere una serie di problemi di soddisfacibilità modulo assegnamenti dove ogni assegnamento iniziale esclude i modelli già trovati.

Approcci alla parallelizzazione di SAT-solutori mediante *ricerca distribuita* (cfr. la Sezione 6.4.1 di [19]) risolvono un problema $\mathcal{S}$ di soddisfacibilità proposizionale, risolvendo in parallelo multiple istanze di soddisfacibilità modulo assegnamenti con lo stesso ingresso $\mathcal{S}$, e diversi assegnamenti iniziali, ciascuno dei quali contiene un distinto *cammino-guida* [142] o *cubo* [88]. Logicamente, un cammino-guida o cubo è un assegnamento che assegna *vero* a ogni letterale in un insieme o congiunzione di letterali. Diversi assegnamenti iniziali inducono i solutori che operano in parallelo a esplorare parti diverse dello spazio di ricerca.

Infine, un problema di *ottimizzazione* può essere affrontato risolvendo iterativamente una serie di problemi di soddisfacibilità modulo assegnamenti, dove ogni assegnamento iniziale contiene informazione generata dalle esecuzioni precedenti, in modo tale che la serie converga verso una soluzione ottima. Si è appena iniziato ad esplorare questo approccio iterativo all'ottimizzazione [63] adattandovi la procedura di soddisfacibilità per la teoria dei numeri reali algebrici [95, 62].

5 Discussione

Il quadro generale della ricerca in deduzione automatica vede vari approcci che mirano ad estendere la *deduzione guidata da conflitti* alla logica del primo ordine. Una motivazione comune a queste direzioni di ricerca è lo stupefacente successo avuto dalla procedura di *Apprendimento di Clausole Guidato da Conflitti*, dall'inglese "Conflict-Driven Clause Learning" (CDCL), per la soddisfacibilità proposizionale [108]. CDCL è uno degli ingredienti fondamentali del successo dei sistemi per la soddisfacibilità proposizionale, detti SAT-solutori, che vengono applicati ai problemi più svariati, risultando spesso più conveniente ridurre un problema a SAT, e risolverlo con un SAT-solutore basato su CDCL, che progettare un algoritmo ad hoc.

Dal lato della soddisfacibilità modulo teorie (SMT), il processo è iniziato con le generalizzazioni della procedura CDCL dalla logica proposizionale a frammenti dell'aritmetica [140, 137, 114, 97, 50, 94, 95, 87]. Il metodo di *Soddisfacibilità con Costruzione di Modello*, o MCSAT, dall'inglese "Model-Constructing SATisfiability", generalizza questi predecessori, e integra la deduzione guidata da conflitti in logica proposizionale con la deduzione guidata da conflitti nella teoria [60, 93, 141, 92]. A sua volta, il metodo di *Soddisfacibilità*

Guidata da Conflitti, o CDSAT, dall'inglese "Conflict-Driven SATisfiability", generalizza MCSAT alla combinazione di teorie e alla *soddisfacibilità modulo assegnamenti* (SMA), dove un assegnamento parziale fa parte del problema in ingresso [26, 27].

Dal lato della dimostrazione automatica di teoremi in logica del primo ordine, DPLL(Γ+$\mathcal{T}$) integra il paradigma DPLL($\mathcal{T}$) per la soddisfacibilità modulo teorie, che incorpora CDCL, con un sistema di inferenza basato su ordinamenti per la logica del primo ordine con uguaglianza, in modo da combinare i rispettivi punti di forza. Successivamente, il metodo di *ragionamento Guidato da Semantica e Sensibile al Goal*, o SGGS, dall'inglese "Semantically-Guided Goal-Sensitive reasoning", generalizza CDCL alla logica del primo ordine [36, 37, 38].

La ricerca sui metodi presentati in questa rassegna è appena agli inizi, come testimonia il fatto che nè DPLL(Γ+$\mathcal{T}$), nè SGGS, nè CDSAT sono stati ancora implementati, e di MCSAT esistono solo implementazioni che ne fanno un satellite di implementazioni di DPLL($\mathcal{T}$), quali CVC4 [7] e Yices [68], invece che il fondamento di un nuovo dimostratore. Pertanto il confronto con sistemi in sviluppo da parecchi anni, come i solutori MathSAT [45], CVC4 [7], Z3 [59], o Yices [68], per la soddisfacibilità modulo teorie, o i dimostratori Prover9 [113], SPASS [139], E [127], o Vampire [98], per la logica del primo ordine, è da ritenersi prematuro. In generale, la finalità della concezione di nuovi metodi di deduzione automatica non è ridimostrare teoremi che i sistemi esistenti già dimostrano velocemente, ma piuttosto esplorare nuovi dominii di applicazione o problemi difficili, dove la deduzione guidata da conflitti possa fare la differenza, così come è stato con CDCL per la soddisfacibilità proposizionale. L'identificazione di tali classi di problemi è anch'essa parte delle ricerche in atto o future.

Bibliografia

[1] Gábor Alagi and Christoph Weidenbach. NRCL – a model building approach to the Bernays-Schönfinkel fragment. In Carsten Lutz and Silvio Ranise, editors, *Proceedings of the Tenth International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 9322 of *Lecture Notes in Artificial Intelligence*, pages 69–84. Springer, 2015.

[2] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal. In Bernhard Gramlich, editor, *Proceedings of the Fifth International Workshop*

on Frontiers of Combining Systems (FroCoS), volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 65–80, Berlin, Germany, EU, 2005. Springer.

[3] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 10(1):129–179, 2009.

[4] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.

[5] Owen L. Astrachan and Mark E. Stickel. Caching and lemmaizing in model elimination theorem provers. In Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 224–238. Springer, 1992.

[6] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[7] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the Twenty-Third International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[8] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the Thirteenth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 512–526, Berlin, Germany, EU, 2006. Springer.

[9] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marjin Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 26, pages 825–886. IOS Press, Amsterdam, The Netherlands, EU, 2009.

[10] Peter Baumgartner and Ulrich Furbach. Consolution as a framework for comparing calculi. *Journal of Symbolic Computation*, 16(5):445–477, 1993.

[11] Peter Baumgartner and Ulrich Furbach. Variants of clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume I: Foundations - Calculi and Methods, chapter 3, pages 73–102. Kluwer Academic Publishers, Amsterdam, The Netherlands, 1998.

[12] Wolfgang Bibel and Elmer Eder. Methods and calculi for deduction. In Dov M. Gabbay, Christopher J. Hogger, and John Alan Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume I: Logical Foundations, pages 68–183. Oxford University Press, Oxford, England, 1993.

[13] Maria Paola Bonacina. On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method. *Journal of Symbolic Computation*, 21(4–6):507–522, December 1996.

[14] Maria Paola Bonacina. A taxonomy of theorem-proving strategies. In Michael J. Wooldridge and Manuela Veloso, editors, *Artificial Intelligence Today – Recent Trends and Developments*, volume 1600 of *Lecture Notes in Artificial Intelligence*, pages 43–84. Springer, Berlin, Germany, EU, 1999.

[15] Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000.

[16] Maria Paola Bonacina. Towards a unified model of search in theorem proving: subgoal-reduction strategies. *Journal of Symbolic Computation*, 39(2):209–255, 2005.

[17] Maria Paola Bonacina. On theorem proving for program checking – historical perspective and recent developments. In Maribel Fernández, editor, *Proceedings of the Twelfth International Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 1–11, New York, New York, USA, 2010. ACM.

[18] Maria Paola Bonacina. On conflict-driven reasoning. In Natarajan Shankar and Bruno Dutertre, editors, *Proceedings of the Sixth Workshop*

on Automated Formal Methods (AFM), Ninth NASA Formal Methods Symposium (NFM), May 2017, volume 5 of *Kalpa Publications*, pages 31–49. EasyChair, 2018.

[19] Maria Paola Bonacina. Parallel theorem proving. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, chapter 6, pages 179–235. Springer, Berlin, Germany, EU, May 2018.

[20] Maria Paola Bonacina and Nachum Dershowitz. Abstract canonical inference. *ACM Transactions on Computational Logic*, 8(1):180–208, January 2007.

[21] Maria Paola Bonacina and Mnacho Echenim. Rewrite-based satisfiability procedures for recursive data structures. In Byron Cook and Roberto Sebastiani, editors, *Proceedings of the Fourth Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR), held at the Fourth Federated Logic Conference (FLoC), August 2006*, volume 174(8) of *Electronic Notes in Theoretical Computer Science*, pages 55–70. Elsevier, Amsterdam, The Netherlands, EU, 2007.

[22] Maria Paola Bonacina and Mnacho Echenim. On variable-inactivity and polynomial $\mathcal{T}$-satisfiability procedures. *Journal of Logic and Computation*, 18(1):77–96, 2008.

[23] Maria Paola Bonacina and Mnacho Echenim. Theory decision by decomposition. *Journal of Symbolic Computation*, 45(2):229–260, 2010.

[24] Maria Paola Bonacina, Ulrich Furbach, and Viorica Sofronie-Stokkermans. On first-order model-based reasoning. In Narciso Martí-Oliet, Peter Olveczky, and Carolyn Talcott, editors, *Logic, Rewriting, and Concurrency: Essays Dedicated to José Meseguer*, volume 9200 of *Lecture Notes in Computer Science*, pages 181–204. Springer, Berlin, Germany, EU, 2015.

[25] Maria Paola Bonacina, Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 513–527, Berlin, Germany, EU, 2006. Springer.

[26] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. Satisfiability modulo theories and assignments. In Leonardo de Moura, editor, *Proceedings of the Twenty-Sixth Conference on Automated Deduction (CADE)*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 42–59, Berlin, Germany, EU, 2017. Springer.

[27] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. CDSAT: transition system and completeness. Submitted for publication and available at `http://profs.sci.univr.it/~bonacina/cdsat.html`, March 2018.

[28] Maria Paola Bonacina and Jieh Hsiang. Towards a foundation of completion procedures as semidecision procedures. *Theoretical Computer Science*, 146:199–242, 1995.

[29] Maria Paola Bonacina and Jieh Hsiang. On semantic resolution with lemmaizing and contraction and a formal treatment of caching. *New Generation Computing*, 16(2):163–200, 1998.

[30] Maria Paola Bonacina and Moa Johansson. Interpolation systems for ground proofs in automated deduction: a survey. *Journal of Automated Reasoning*, 54(4):353–390, 2015.

[31] Maria Paola Bonacina and Moa Johansson. On interpolation in automated theorem proving. *Journal of Automated Reasoning*, 54(1):69–97, 2015.

[32] Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by DPLL($\Gamma+\mathcal{T}$) and unsound theorem proving. In Renate Schmidt, editor, *Proceedings of the Twenty-second International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 35–50, Berlin, Germany, EU, 2009. Springer.

[33] Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by theorem proving with speculative inferences. *Journal of Automated Reasoning*, 47(2):161–189, 2011.

[34] Maria Paola Bonacina and Alberto Martelli. Automated reasoning. *Intelligenza Artificiale*, 3(1–2):14–20, 2006. Special issue on Artificial Intelligence 50th Anniversary 1956–2006.

[35] Maria Paola Bonacina and David A. Plaisted. Constraint manipulation in SGGS. In Temur Kutsia and Christophe Ringeissen, editors, *Proceedings of the Twenty-Eighth Workshop on Unification (UNIF) at the Sixth Federated Logic Conference (FLoC)*, Technical Reports of the Research Institute for Symbolic Computation, pages 47–54, Linz, Austria, EU, 2014. Johannes Kepler Universität Linz.

[36] Maria Paola Bonacina and David A. Plaisted. SGGS theorem proving: an exposition. In Stephan Schulz, Leonardo De Moura, and Boris Konev, editors, *Proceedings of the Fourth Workshop on Practical Aspects in Automated Reasoning (PAAR) at the Sixth Federated Logic Conference (FLoC), July 2014*, volume 31 of *EasyChair Proceedings in Computing (EPiC)*, pages 25–38, Manchester, England, UK, 2015. EasyChair.

[37] Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive reasoning: model representation. *Journal of Automated Reasoning*, 56(2):113–141, 2016.

[38] Maria Paola Bonacina and David A. Plaisted. Semantically-guided goal-sensitive reasoning: inference system and completeness. *Journal of Automated Reasoning*, 59(2):165–218, 2017.

[39] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation - Decision Procedures with Applications to Verification*. Springer, Berlin, Germany, EU, 2007.

[40] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In E. Allen Emerson and Kedar S. Namjoshi, editors, *Proceedings of the Seventh International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442, Berlin, Germany, EU, 2006. Springer.

[41] Davide Bresolin, Valentin Goranko, Angelo Montanari, and Pietro Sala. Tableau-based decision procedures for the logics of subinterval structures over dense orderings. *Journal of Logic and Computation*, 20:133–166, 2010.

[42] Robert Brummayer and Armin Biere. Lemmas on demand for the extensional theory of arrays. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:165–201, 2009.

[43] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[44] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Cambridge, England, UK, 1973.

[45] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In Nir Piterman and Scott Smolka, editors, *Proceedings of the Nineteenth Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107, Berlin, Germany, EU, 2013. Springer.

[46] Stefania Costantini and Andrea Formisano. Answer set programming with resources. *Journal of Logic and Computation*, 20(2):533–571, 2010.

[47] Stefania Costantini and Andrea Formisano. Nested weight constraints in ASP. *Fundamenta Informaticae*, 124(4):449–464, 2013.

[48] Stefania Costantini and Andrea Formisano. Negation as a resource: a novel view on answer set semantics. *Fundamenta Informaticae*, 140(3–4):279–305, 2015.

[49] Stefania Costantini and Andrea Formisano. Query answering in resource-based answer set semantics. *Theory and Practice of Logic Programming*, 16(5–6):619–635, 2016.

[50] Scott Cotton. Natural domain SMT: A preliminary assessment. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6246 of *Lecture Notes in Computer Science*, pages 77–91, Berlin, Germany, EU, 2010. Springer.

[51] William Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[52] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.

[53] Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: answer set programming with lazy grounding. *Fundamenta Informaticae*, 96(3):297–322, 2009.

[54] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[55] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[56] Leonardo de Moura and Nikolaj Bjørner. Efficient E-matching for SMT-solvers. In Frank Pfenning, editor, *Proceedings of the Twenty-first Conference on Automated Deduction (CADE)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 183–198, Berlin, Germany, EU, 2007. Springer.

[57] Leonardo de Moura and Nikolaj Bjørner. Engineering DPLL(T) + saturation. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proceedings of the Fourth International Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 475–490, Berlin, Germany, EU, 2008. Springer.

[58] Leonardo de Moura and Nikolaj Bjørner. Model-based theory combination. In Sava Krstić and Albert Oliveras, editors, *Proceedings of the Fifth International Workshop on Satisfiability Modulo Theories (SMT 2007)*, volume 198(2) of *Electronic Notes in Theoretical Computer Science*, pages 37–49, Amsterdam, The Netherlands, EU, 2008. Elsevier.

[59] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the Fourteenth Conference on Tools and algorithms for the construction and analysis of systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Berlin, Germany, EU, 2008. Springer.

[60] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Proceedings of the Fourteenth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, volume 7737 of *Lecture Notes in Computer Science*, pages 1–12, Berlin, Germany, EU, 2013. Springer.

[61] Leonardo de Moura, Dejan Jovanović, and Grant Olney Passmore. Arithmetic and optimization @ MCSAT. Talk given by Leonardo de Moura at a Schloss Dagsthul Seminar on Deduction and Arithmetic, October 2013.

[62] Leonardo de Moura and Grant Olney Passmore. Computation over real closed infinitesimal and transcendental extensions of the rationals. In Maria Paola Bonacina, editor, *Proceedings of the Twenty-Fourth Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 177–191, Berlin, Germany, EU, 2013. Springer.

[63] Leonardo de Moura and Grant Olney Passmore. Exact global optimization on demand (presentation only). In Silvio Ghilardi, Viorica Sofronie-Stokkermans, and Ashish Tiwari, editors, *Notes of the Third Workshop on Automated Deduction: Decidability, Complexity, Tractability (ADDCT)*, pages 50–50, 2013. Available at `https://userpages.uni-koblenz.de/~sofronie/addct-2013/`, last seen on May 9, 2017.

[64] Leonardo de Moura and Harald Rueß. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Application of Satisfiability Testing (SAT)*, Lecture Notes in Computer Science, pages 244–251, Berlin, Germany, EU, 2002. Springer.

[65] David L. Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.

[66] Agostino Dovier, Andrea Formisano, and Enrico Pontelli. An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. *Journal of Experimental and Theoretical Artificial Intelligence*, 21(2):79–121, 2009.

[67] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, 1980.

[68] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Proceedings of the Twenty-Sixth International Conference on Computer-Aided Verification (CAV)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744, Berlin, Germany, EU, 2014. Springer.

[69] Bruno Dutertre and Leonardo de Moura. A fast linear arithmetic solver for DPLL(T). In Tom Ball and R. B. Jones, editors, *Proceedings of the Eighteenth International Conference on Computer Aided Verification (CAV)*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94, Berlin, Germany, EU, 2006. Springer.

[70] Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. Simplification rules for intuitionistic propositional tableaux. *ACM Transactions on Computational Logic*, 13(2):14:1–14:23, 2012.

[71] Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models. *Journal of Automated Reasoning*, 51(2):129–149, 2013.

[72] Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. JTabWb: a Java framework for implementing terminating sequent and tableau calculi. *Fundamenta Informaticae*, 150(1):119–142, 2017.

[73] Camillo Fiorentini. Terminating sequent calculi for proving and refuting formulas in S4. *Journal of Logic and Computation*, 25(1):179–205, 2015.

[74] Andrea Formisano, Eugenio G. Omodeo, and Marco Temperini. Goals and benchmarks for automated map reasoning. *Journal of Symbolic Computation*, 29(2):259–297, 2000.

[75] Andrea Formisano, Eugenio G. Omodeo, and Marco Temperini. Instructing equational set-reasoning with Otter. In Rajeev P. Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 152–167, Berlin, Germany, EU, 2001. Springer.

[76] Marc Fuchs. Controlled use of clausal lemmas in connection tableau calculi. *Journal of Symbolic Computation*, 29(2):299–341, 2000.

[77] Harald Ganzinger, Harald Rueß, and Natarajan Shankar. Modularity and refinement in inference systems. Technical Report CSL-SRI-04-02, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, 2004.

[78] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In Frank Pfenning, editor, *Proceedings of the Twenty-first Conference on Automated Deduction (CADE)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Berlin, Germany, EU, 2007. Springer.

[79] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Analytic tableaux calculi for KLM logics of nonmonotonic reasoning. *ACM Transactions on Computational Logic*, 10(3):18:1–18:47, 2009.

[80] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. A non-monotonic description logic for reasoning about typicality. *Artificial Intelligence*, 195:165–202, 2013.

[81] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Camilla B. Schwind. Tableau calculus for preference-based conditional logics: PCL and its extensions. *ACM Transactions on Computational Logic*, 10(3):21:1–21:50, 2009.

[82] Laura Giordano and Alberto Martelli. Tableau-based automata construction for dynamic linear time temporal logic. *Annals of Mathematics and Artificial Intelligence*, 46(3):289–315, 2006.

[83] Laura Giordano, Alberto Martelli, and Daniele Theseider Dupré. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 13(2), 2013.

[84] Laura Giordano, Alberto Martelli, and Daniele Theseider Dupré. Achieving completeness in the verification of action theories by bounded model checking in ASP. *Journal of Logic and Computation*, 25(6):1307–1330, 2015.

[85] Laura Giordano and Daniele Theseider Dupré. ASP for minimal entailment in a rational extension of SROEL. *Theory and Practice of Logic Programming*, 16(5–6), 2016.

[86] Valentin Goranko, Angelo Montanari, Pietro Sala, and G. Sciavicco. A general tableau method for propositional interval temporal logics: theory and implementation. *Journal of Applied Logic*, 4(3):305–330, 2006.

[87] Leopold Haller, Alberto Griggio, Martin Brain, and Daniel Kroening. Deciding floating-point logic with systematic abstraction. In Gianpiero

Cabodi and Satnam Singh, editors, *Proceedings of the Twelfth International Conference on Formal Methods in Computer Aided Design (FMCAD)*, New York, New York, USA, 2012. ACM and IEEE.

[88] Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: guiding CDCL SAT solvers by lookaheads. In K. Eder, J. Lourenço, and O. Shehory, editors, *Proceedings of the Seventh Haifa Verification Conference (HVC)*, volume 7261 of *Lecture Notes in Computer Science*, pages 50–65, Berlin, Germany, EU, 2012. Springer.

[89] Jieh Hsiang and Michaël Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, 1991.

[90] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the Fourteenth Conference on Tools and algorithms for the construction and analysis of systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 265–281, Berlin, Germany, EU, 2008. Springer.

[91] Daniyar Itegulov, John Slaney, and Bruno Woltzenlogel Paleo. Scavenger 0.1: a theorem prover based on conflict resolution. In Leonardo de Moura, editor, *Proceedings of the Twenty-Sixth Conference on Automated Deduction (CADE)*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 344–356, Berlin, Germany, EU, 2017. Springer.

[92] Dejan Jovanović. Solving nonlinear integer arithmetic with MCSAT. In Ahmed Bouajjani and David Monniaux, editors, *Proceedings of the Eighteenth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, volume 10145 of *Lecture Notes in Computer Science*, pages 330–346, Berlin, Germany, EU, 2017. Springer.

[93] Dejan Jovanović, Clark Barrett, and Leonardo de Moura. The design and implementation of the model-constructing satisfiability calculus. In Barbara Jobstman and Sandip Ray, editors, *Proceedings of the Thirteenth Conference on Formal Methods in Computer Aided Design (FMCAD)*, New York, New York, USA, 2013. ACM and IEEE.

[94] Dejan Jovanović and Leonardo de Moura. Cutting to the chase: solving linear integer arithmetic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Proceedings of the Twenty-Third International*

Conference on Automated Deduction (CADE), volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 338–353, Berlin, Germany, EU, 2011. Springer.

[95] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Ulrike Sattler, editors, *Proceedings of the Sixth International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 339–354, Berlin, Germany, EU, 2012. Springer.

[96] Richard E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[97] Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. Conflict resolution. In Ian P. Gent, editor, *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *Lecture Notes in Computer Science*, pages 509–523, Berlin, Germany, EU, 2009. Springer.

[98] Laura Kovàcs and Andrei Voronkov. First order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the Twenty-Fifth International Conference on Computer-Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, Berlin, Germany, EU, 2013. Springer.

[99] Robert Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.

[100] Sava Krstić and Amit Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In Frank Wolter, editor, *Proceedings of the Sixth International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 1–27, Berlin, Germany, EU, 2007. Springer.

[101] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

[102] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[103] Reinhold Letz. Clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume I: Foundations - Calculi and Methods, chapter 2, pages 43–72. Kluwer Academic Publishers, Amsterdam, The Netherlands, 1998.

[104] Reinhold Letz, Klaus Mayr, and Christian Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–338, 1994.

[105] Reinhold Letz, Johann Schumann, Stephan Bayerl, and Wolfgang Bibel. SETHEO: a high performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[106] Reinhold Letz and Gernot Stenz. Model elimination and connection tableau procedures. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 28, pages 2015–2114. Elsevier, Amsterdam, The Netherlands, 2001.

[107] Donald W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363, 1969.

[108] Sharad Malik and Lintao Zhang. Boolean satisfiability: from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[109] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marjin Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, Amsterdam, The Netherlands, EU, 2009.

[110] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[111] J. McCharen, Ross Overbeek, and Larry Wos. Complexity and related enhancements for automated theorem proving programs. *Computers and Mathematics with Applications*, 2(1):1–16, 1976.

[112] William W. McCune. OTTER 3.3 reference manual. Technical Report TM-263, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, August 2003.

[113] William W. McCune. Prover9 and Mace4, 2005–2010. `http://www.cs.unm.edu/~mccune/prover9/`, last seen on May 10, 2017.

[114] Kenneth L. McMillan, A. Kuehlmann, and Mooly Sagiv. Generalizing DPLL to richer logics. In Ahmed Bouajjani and Oded Maler, editors, *Proceedings of the Twenty-First International Conference on Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 462–476, Berlin, Germany, EU, 2009. Springer.

[115] Scott McPeak and George C. Necula. Data structure specifications via local equality axioms. In Kousha Etessami and Sriram. K. Rajamani, editors, *Proceedings of the Seventeenth International Conference on Computer Aided Verification (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 476–490. Springer, 2005.

[116] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In David Blaauw and Luciano Lavagno, editors, *Proceedings of the Thirty-Ninth Design Automation Conference (DAC)*, pages 530–535, New York, New York, USA, 2001. ACM and IEEE.

[117] Greg Nelson. Combining satisfiability procedures by equality sharing. In Woodrow W. Bledsoe and Donald W. Loveland, editors, *Automatic Theorem Proving: After 25 Years*, pages 201–211. American Mathematical Society, Providence, Rhode Island, USA, 1983.

[118] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[119] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[120] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[121] Nicola Olivetti, Gian Luca Pozzato, and Camilla B. Schwind. A sequent calculus and a theorem prover for standard conditional logics. *ACM Transactions on Computational Logic*, 8(4):22:1–22:51, 2007.

[122] Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *Journal of Automated Reasoning*, 44(4):401–424, 2010.

[123] David A. Plaisted and Yunshan Zhu. *The Efficiency of Theorem Proving Strategies*. Friedr. Vieweg & Sohns, 1997.

[124] John Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.

[125] John Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[126] Michaël Rusinowitch. Theorem-proving with resolution and superposition. *Journal of Symbolic Computation*, 11(1 & 2):21–50, 1991.

[127] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the Nineteenth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 8312 of *Lecture Notes in Artificial Intelligence*, pages 735–743, Berlin, Germany, EU, 2013. Springer.

[128] Stephan Schulz and Martin Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In Nicola Olivetti and Ashish Tiwari, editors, *Proceedings of the Eighth International Conference on Automated Reasoning (IJCAR)*, volume 9706 of *Lecture Notes in Artificial Intelligence*, pages 330–345. Springer, 2016.

[129] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal of Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[130] Natarajan Shankar. Automated deduction for verification. *ACM Computing Surveys*, 41(4):40–96, 2009.

[131] Robert E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.

[132] John Slaney and Bruno Woltzenlogel Paleo. Conflict resolution: a first-order resolution calculus with decision literals and conflict-driven clause learning. *Journal of Automated Reasoning*, 60(2):133–156, 2018.

[133] Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In Robert Nieuwenhuis, editor, *Proceedings of the Twentieth Conference on Automated Deduction (CADE)*, volume 3632 of *Lecture Notes in Artificial Intelligence*, pages 219–234, Berlin, Germany, EU, 2005. Springer.

[134] Mark E. Stickel and W. Mabry Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1073–1075. Morgan Kaufmann Publishers, 1985.

[135] Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy Levitt. A decision procedure for an extensional theory of arrays. In Joseph Halpern, editor, *Proceedings of the Sixteenth IEEE Symposium on Logic in Computer Science (LICS)*, Los Alamitos, California, USA, 2001. IEEE Computer Society Press.

[136] Kevin Wallace and Graham Wrightson. Regressive merging in model elimination tableau-based theorem provers. *Journal of the IGPL*, 3(6):921–937, 1995.

[137] Chao Wang, Franjo Ivančić, Malay Ganai, and Aarti Gupta. Deciding separation logic formulae by SAT and incremental negative cycle elimination. In Geoff Sutcliffe and Andrei Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic, Programming and Automated Reasoning (LPAR)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 322–336, Berlin, Germany, EU, 2005. Springer.

[138] David S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):94–111, 1992.

[139] Christoph Weidenbach, Dylana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate Schmidt, editor, *Proceedings of the Twenty-Second International Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 140–145, Berlin, Germany, EU, 2009. Springer.

[140] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine and its application to resource planning. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*

(IJCAI), volume 1, pages 310–316, San Francisco, California, USA, 1999. Morgan Kaufmann Publishers.

[141] Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. Deciding bit-vector formulas with mcSAT. In Nadia Creignou and Daniel Le Berre, editors, *Proceedings of the Nineteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *Lecture Notes in Computer Science*, pages 249–266, Berlin, Germany, EU, 2016. Springer.

[142] Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21(4–6):543–560, 1996.

[143] Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24(1/2):277–296, 2000.

[144] Neng-Fa Zhou, Roman Barták, and Agostino Dovier. Planning as tabled logic programming. *Theory and Practice of Logic Programming*, 15(4–5):543–558, 2015.

[145] Neng-Fa Zhou and Agostino Dovier. A tabled Prolog program for solving Sokoban. *Fundamenta Informaticae*, 124:1–15, 2013.

Maria Paola Bonacina è Professoressa Ordinaria di Informatica presso l'Università degli Studi di Verona, Dipartimento di Informatica.
mariapaola.bonacina@univr.it