Capitolo 1

Logica e automazione dei problemi di Decisione

In questo capitolo verranno descritte le nozioni di base necessarie per comprendere il lavoro svolto. In particolare, verranno introdotti i concetti di logica proposizionale e del primo ordine, definita come estensione della prima. Nell'ultimo paragrafo del capitolo verrà descritto in che modo le formule di logica del primo ordine possono essere rappresentate in un formato di file, per poi essere processate come input da un theorem prover. Lo scopo di questo capitolo è quello di accennare la teoria logica utilizzata nell'implementazione di vampire e della procedura di decisione per i Binding-Fragments. Perciò, verranno date per scontate nozioni di teoria degli insiemi, algebra e teoria dei linguaggi.

1.1 Logica Proposizionale

1.1.1 Formule

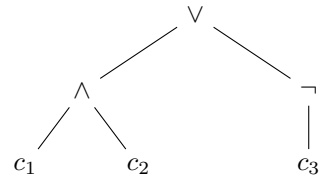
Sia $\Sigma_c = \{c_1, c_2, ...\}$ un insieme di simboli di costante, $\Sigma = \{\wedge, \vee, \neg, (,), \top, \bot\} \cup \Sigma_c$ è detto alfabeto della logica proposizionale. Con queste premesse possiamo definire come formule della logica proposizionale il linguaggio $F \subseteq \Sigma_c$ generato dalla grammatica Context Free seguente:

$$\varphi := \top \mid \bot \mid C \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

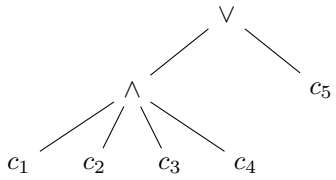
Dove $C \in \Sigma_c$ è un simbolo di costante. Con la funzione $const(\gamma) \to \Sigma_c$ si indica la funzione che associa a ogni formula γ l'insieme dei suoi simboli di costante. Viene chiamato *Letterale*, ogni simbolo di costante c o la sua negazione $\neg c$. Vengono inoltre introdotti i seguenti simboli come abbreviazioni:

- $(\gamma \Rightarrow \kappa)$ per $(\neg\gamma \vee \kappa)$
- $(\gamma \Leftrightarrow \kappa)$ per $((\gamma \Rightarrow \kappa) \wedge (\kappa \Rightarrow \gamma))$
- $(\gamma \oplus \kappa)$ per $\neg(\gamma \Leftrightarrow \kappa)$

È possibile rappresentare una qualunque formula attraverso il proprio albero di derivazione. Questo albero verrà chiamato in seguito anche *albero sintattico* della formula. Ad esempio, la formula $(c_1 \wedge c_2) \vee \neg c_3$ può essere rappresentata dal seguente albero sintattico:



La radice dell'albero è detta *connettivo principale* e i sotto alberi della formula vengono dette *sottoformule*. Per compattezza, grazie alla proprietà associativa di $\wedge$ e $\vee$, è possibile omettere le parentesi, es. $(c_1 \wedge (c_2 \wedge (c_3 \wedge c_4))) \vee c_5$ può essere scritto come $(c_1 \wedge c_2 \wedge c_3 \wedge c_4) \vee c_5$. Allo stesso modo, nell'albero sintattico della formula è possibile compattare le catene di $\wedge$ e $\vee$ come figli di un unico nodo:



Questa è una caratteristica molto importante, in quanto non solo permette di risparmiare inchiostro, ma consente di vedere $\wedge$ e $\vee$ non più come operatori binari ma come operatori n-ari. A livello implementativo, ciò si traduce in un minor impatto in memoria, visite all'albero più veloci e algoritmi di manipolazione più semplici. Si consideri ad esempio di voler ricercare la foglia più a sinistra nell'albero di derivazione della seguente formula $((...(((c_1 \wedge c_2) \wedge c_3) \wedge c_4) \wedge ...) \wedge c_n)$. Senza compattazione, l'algoritmo di ricerca impiegherebbe $O(n)$ operazioni, mentre con la compattazione $O(1)$.

1.1.2 Assegnamenti

Un *assegnamento* è una qualunque funzione α da un insieme $C \subseteq \Sigma_c$ nell'insieme $\{1, 0\}$ (o $\{True, False\}$).

$$\alpha : C \to \{1, 0\}$$

Un assegnamento α è detto *appropriato* per una formula $\varphi \in F$ se e solo se $const(\varphi) \subseteq dom(\alpha)$.

Si definisce la relazione binaria di *Soddisfacibilità*:

$$\models \subseteq \{1, 0\}^C \times F$$

In modo tale che dato un assegnamento α appropriato a una formula φ, si dice che $\alpha \models \varphi$ (α soddisfa φ) o anche α è un assegnamento per φ o se e solo se:

- Se φ è una variabile c_x allora $\alpha \models \varphi$ sse $\alpha(c_x) = 1$
- Se φ è della forma $\neg\psi$ (dove ψ è una formula) allora $\alpha \models \varphi$ sse $\alpha \not\models \psi$
- Se φ è della forma $(\psi \wedge \chi)$ (con ψ e χ formule) allora $\alpha \models \varphi$ sse $\alpha \models \psi$ e $\alpha \models \chi$
- Se φ è della forma $(\psi \vee \chi)$ (con ψ e χ formule) allora $\alpha \models \varphi$ sse $\alpha \models \psi$ o $\alpha \models \chi$

Una *Tautologia* è una formula φ tale che per ogni assegnamento α appropriato a φ, $\alpha \models \varphi$ (in simboli $\models \varphi$). Una formula è detta soddisfacibile se esiste un assegnamento appropriato che la soddisfa altrimenti è detta insoddisfacibile. Date due formule φ e ψ, si dice che ψ è *conseguenza logica* di φ (in simboli $\varphi \models \psi$) se e solo se per ogni assegnamento α appropriato a entrambe le formule, se $\alpha \models \varphi$ allora $\alpha \models \psi$. Due formule sono dette *equivalenti* sse $\varphi \models \psi$ e $\psi \models \varphi$ (in simboli $\varphi \equiv \psi$). Un'importante proprietà è che se $\varphi \models \psi$ allora la formula $\varphi \Rightarrow \psi$ è una tautologia ($\models \varphi \Rightarrow \psi$).

Due concetti molto simili a quello di equivalenza e conseguenza logica sono l'*equisoddisfacibilità* e la *soundness*. In pratica, due formule sono sound se e solo se, se la prima formula è soddisfacibile allora

lo è anche la seconda. Due formule sono equisoddisfacibili se e solo se sono sound in entrambe le direzioni. Quindi la conseguenza logica implica la soundness ma non il viceversa. Allo stesso modo l'equivalenza logica implica l'equisoddisfacibilità ma non il viceversa. Si consideri ad esempio le due formule $\varphi = c_1$ e $\psi = \neg c_1$. Ovviamente non può esserci conseguenza logica tra le due formule, ma sono equisoddisfacibili, infatti se α è un assegnamento per φ allora è possibile costruire un assegnamento β per ψ tale che $\beta(c_1) = 1 - \alpha(c_1)$ e viceversa.

Un'*inferenza* è una qualunque funzione da F in F. Un'inferenza è detta *corretta* se conserva la soddisfacibilità, ovvero se non può generare una formula insoddisfacibile a partire da una formula soddisfacibile (soundness).

Infine, si definisce *Implicante* di una formula φ un insieme I di letterali di φ che rendono vera φ. Cioè, costruendo una assegnazione α tale che $\alpha \models c$ per ogni letterale $c \in I$, si ha che $\alpha \models \varphi$. In altre parole la formula costruita dalla congiunzione di tutti i letterali di I implica logicamente φ. Spesso con abuso di terminologia gli elementi di I vengono chiamati anch'essi implicanti, di solito è facile intuire dal contesto se si sta parlando dell'insieme o dei letterali. È possibile anche costruire un Implicante a partire da una assegnazione. È sufficiente prendere l'insieme dei letterali della formula soddisfatti dall'assegnamento e si ottiene così un implicante.

1.1.3 Forme Normali

Una delle strategie più utilizzate dai dimostratori di teoremi automatici è la *normalizzazione* delle formule. Una *forma normale* è essenzialmente un sottoinsieme di F che rispetta determinate proprietà. Una *normalizzazione* invece è il processo di trasformazione di una formula tramite una successione d'inferenze (corrette) in una forma normale. In questo paragrafo verranno descritte le tre forme normali che sono state utilizzate per il preprocessing dell'algoritmo. La prima e l'ultima ossia le forme NNF e CNF sono le più famose e utilizzate, mentre la seconda, la ENNF, non è abbastanza conosciuta da essere definita standard e viene utilizzata per bypassare alcuni problemi di efficienza causati dalla CNF grazie all'utilizzo di tecniche di Naming, che però verranno discusse nella prossima sezione.

La prima tra queste è la *NNF* ossia *Negated Normal Form* (Forma normale negata). Una formula è in formato NNF sse non contiene connettivi semplificati ($\Rightarrow$, $\Leftrightarrow$, $\oplus$) e la negazione è applicata solo a letterali. La classe di formule NNF è generata dalla seguente grammatica:

$$\eta := \top \mid \bot \mid C \mid \neg C \mid (\eta \wedge \eta) \mid (\eta \vee \eta)$$

Dove $C \in \Sigma_c$ è un simbolo di costante. La normalizzazione di una formula in NNF è un processo semplice che consiste nell'applicare opportunamente le regole di De Morgan e le regole di semplificazione dei connettivi.

La seconda forma normale è la *ENNF* ossia *Extended Negated Normal Form* (Forma normale negata estesa). Il formato ENNF è essenzialmente una classe più permissiva della NNF, in quanto conserva il vincolo sulla negazione ma vieta esclusivamente l'uso di '$\Rightarrow$'. La classe di formule ENNF è generata dalla seguente grammatica:

$$\overline{\eta} := \top \mid \bot \mid C \mid \neg C \mid (\overline{\eta} \wedge \overline{\eta}) \mid (\overline{\eta} \vee \overline{\eta}) \mid (\overline{\eta} \Leftrightarrow \overline{\eta}) \mid (\overline{\eta} \oplus \overline{\eta})$$

La terza e ultima forma normale è la *CNF* ossia *Conjunctive Normal Form* (Forma normale congiuntiva). Una formula è in formato CNF sse è una congiunzione di disgiunzioni di letterali. La classe di formule CNF è generata dalla seguente grammatica:

$$\zeta := \xi \mid (\xi \wedge \zeta)$$

$$\xi := \top \mid \bot \mid C \mid \neg C \mid (\xi \vee \xi)$$

La classe CNF è storicamente la più famosa e utilizzata, in quanto è la più semplice da implementare e da manipolare. È possibile vedere le clausole come insiemi di letterali mentre la formula principale è vista come un insieme di clausole. Ad esempio, la CNF $(c_1 \vee \neg c_2) \wedge (c_3)$ può essere rappresentata in termini insiemistici come $\{\{c_1, \neg c_2\}, \{c_3\}\}$. La clausola vuota $\{\}$ è una clausola speciale che rappresenta la formula $\bot$, viene spesso raffigurata dal simbolo $\square$. La normalizzazione di una formula in CNF è un processo più complesso rispetto alle altre due forme normali. Non esiste un unica tecnica di normalizzazione, ma una strategia comune è questa:

1. Si trasforma la formula in NNF.
2. Se la formula è del tipo $\varphi_1 \wedge ... \wedge \varphi_n$ allora la struttura principale è già una congiunzione di formule, quindi si procede applicando l'algoritmo sulle sottoformule $\varphi_1, ..., \varphi_n$.
3. Se la formula è del tipo $(\varphi_1 \wedge \varphi_2) \vee \psi_1$ si applica la proprietà distributiva di $\vee$ su $\wedge$ in modo da spingere i connettivi $\vee$ il più possibile in profondità. Si ottiene così una formula del tipo $(\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_1)$ si procede poi ricorsivamente con il punto 2.

Il processo di generazione delle clausole prende il nome di *clausificazione*. Questa tecnica di clausificazione nella peggiore delle ipotesi porta a una generazione di un numero di clausole esponenziale rispetto alla dimensione della formula originale. Ad esempio la formula $(c_1 \wedge c_2) \vee (c_3 \wedge c_4) \vee ... \vee (c_{n-1} \wedge c_n)$ genera esattamente 2^n clausole diverse tutte da n letterali.

1.1.4 Naming

1.2 Logica del primo ordine

1.2.1 Termini e Formule

Oltre al solito insieme Σ_c di simboli di costante, vengono introdotti tre nuovi insiemi di simboli:

- $\Sigma_f = \{f_1, f_2, ...\}$ insieme di simboli di funzione
- $\Sigma_p = \{p_1, p_2, ...\}$ insieme di simboli di predicato (o relazione)
- $\Sigma_x = \{x_1, x_2, ...\}$ insieme di simboli di variabile

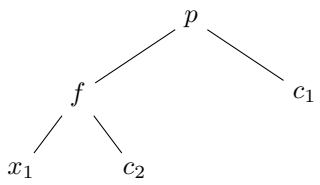
Definiamo la funzione $arity : \Sigma_f \cup \Sigma_p \to \mathbb{N}$ che associa ad ogni simbolo di funzione o predicato la sua arità. Un *termine* è una stringa generata dalla seguente grammatica:

$$\tau := X \mid C \mid f(\tau_1, ..., \tau_n)$$

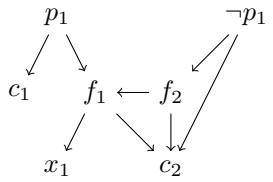
Dove X è un simbolo di variabile, C è un simbolo di costante e f è un simbolo di funzione tale che $arity(f) = n$. In altre parole:

- Ogni variabile è un termine
- Ogni costante è un termine
- Se $\tau_1, ..., \tau_n$ sono termini e f è un simbolo di funzione di arità n allora $f(\tau_1, ..., \tau_n)$ è un termine

Indichiamo con T l'insieme di tutti i termini generati dalla grammatica precedente. Chiameremo *Atomo* tutte le stringhe del tipo $p(\tau_1, ..., \tau_n)$ dove p è un simbolo di relazione di arità n e $\tau_1, ..., \tau_n$ sono termini. Vengono chiamati *Letterali* tutti gli atomi o la loro negazione. Termini e Letterali sono detti *ground* se non contengono variabili. Come già visto per le formule proposizionali, è possibile rappresentare un termine o un letterale attraverso il proprio albero di derivazione. Ad esempio, il letterale $p_1(f(x_1, c_2), c_1)$ può essere rappresentato dal seguente albero sintattico:



Come intuibile i sottoalberi di un termine sono detti *sottotermini*. Si assuma di avere due letterali $p_1(c_1, f_1(x_1, c_2))$ e $\neg p_1(f_2(f_1(x_1, c_2), c_2), c_2)$ di volerli rappresentare in un unico grafo. Al posto di creare una foresta con due alberi indipendenti, è possibile creare un unica struttura condividendo i sottotermini comuni:



Una struttura del genere è detta *Prfectly Shared* (Perfettamente condivisa). Nella pratica questa tecnica di condivisione di sottotermini è indispensabile dato che, anche se a un costo per la creazione e la gestione non indifferente, permette un risparmio di memoria e di tempo considerevole. Per effettuare ad esempio un controllo di uguaglianza tra due sottotermini è sufficiente controllare che le due frecce che partono dai termini padre puntino allo stesso sottotermine, senza dover visitare l'intera sotto struttura, rendendo così tale operazione a tempo costante.

1.2.2 Semantica

1.2.3 Forme Normali

1.2.4 Skolemizzazione

1.2.5 Unificazione

1.3 Soddisfacibilità e Validità

1.4 Resolution

1.5 Il formato TPTP