Università degli Studi di Napoli Federico II
Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea in Informatica

Implementazione di una procedura di decisione per Binding-Fragments in Vampire

Relatore
Prof. Massimo Benerecetti

Correlatore
Prof. Fabio Mogavero

Candidato
Matteo Richard Gaudino

Matricola
N86003226

Anno Accademico 2022 - 2023

Indice

Introduzione **5**

1 Logica e automazione dei problemi di Decisione **6**
1.1 Logica Proposizionale 6
1.1.1 Formule 6
1.1.2 Assegnamenti 7
1.1.3 Forme Normali 9
1.1.4 Naming 10
1.2 Logica del primo ordine 10
1.2.1 Termini e Formule 10
1.2.2 Unificazione 12
1.2.3 Semantica 14
1.2.4 Skolemizzazione e Forme Normali 16
1.3 Soddisfacibilità e Validità 19
1.4 Resolution 19
1.5 Il formato TPTP 19

2 Algoritmo di decisione di Frammenti Binding **21**
2.1 Classificazione 21
2.2 Algoritmo Astratto 21

3 Il Theorem prover Vampire **22**
3.1 I Termini 22
3.2 Formule e Clausole 22
3.3 Unificazione e Substitution Trees 22
3.4 Preprocessing 22
3.5 Saturazione e Refutazione 22
3.6 Il SAT-Solver 22
3.7 Misurazione dei Tempi 22
3.8 Opzioni 22

4 Implementazione di procedure di decisione per frammenti Binding in Vampire **23**
4.1 Algoritmo di Classificazione 23
4.2 Preprocessing 23
4.2.1 Boolean Top Formula 23
4.2.2 Forall-And 23
4.2.3 SAT-Clausification 23
4.3 Procedura di Decisione 23
4.3.1 Implicants Sorting 23
4.3.2 Maximal Unifiable Subsets 23
4.3.3 Algoritmo Finale 23

5 Analisi Sperimentale **24**
5.1 La libreria TPTP 24
5.2 Analisi dei risultati 24
5.3 Ottimizzazioni 24
5.4 Conclusioni e Possibili Sviluppi futuri 24

Introduzione

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

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 si può definire come formule della logica proposizionale il linguaggio F generato dalla grammatica Context Free seguente:

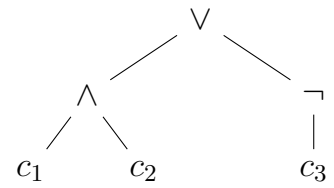
$$\varphi := \top \mid \bot \mid C \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

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:

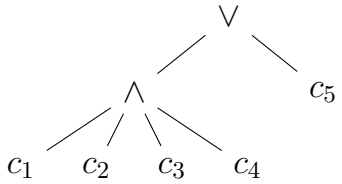
- $(\gamma \Rightarrow \kappa)$ per $(\neg\gamma \vee \kappa)$

- $(\gamma \Leftrightarrow \kappa)$ 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 costante (o $\top/\bot$) 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 $\alpha \models \chi$

Per convenzione si assume che $\alpha(\top) = 1$ e $\alpha(\bot) = 0$ per ogni assegnamento α. 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. In questo caso, tutte e tre le forme presentate preservano la relazione di equivalenza logica, quindi è sempre possibile trasformare una formula in un'altra equivalente in uno di questi tre formati. 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

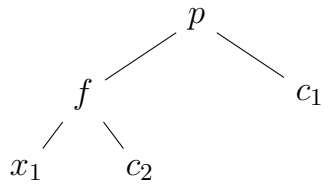
Si definisce la funzione $arity : \Sigma_f \cup \Sigma_p \to \mathbb{N}$ che associa ad ogni simbolo di funzione o predicato la sua arità. I simboli contenuti in $\Sigma_c \cup \Sigma_f \cup \Sigma_p$ sono detti *simboli non logici* e ogni suo sottoinsieme è detto *tipo*. Un *termine* è una stringa generata dalla seguente grammatica:

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

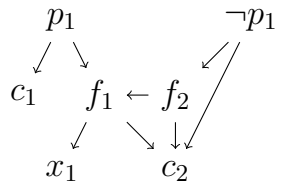
Dove X è un simbolo di variabile, C è un simbolo di costante e f è un simbolo di funzione tale che $arity(f) = n$. In altre parole:

- 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

Si indica 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. Si considerano atomi anche tutti i simboli di costante. Vengono chiamati *Letterali* tutti gli atomi e 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.

A questo punto si definisce finalmente le formule della logica del primo ordine. Prendendo come punto di partenza le formule proposizionali, si definisce alfabeto delle formule del primo ordine l'insieme: $\Sigma' = \Sigma \cup \Sigma_f \cup \Sigma_p \cup \Sigma_x \cup \{\forall, \exists\}$ e Si definisce come formule del primo ordine il linguaggio F' generato dalla seguente grammatica:

$$\phi := \top \mid \bot \mid A \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid \forall x(\phi) \mid \exists x(\phi)$$

Dove A è un atomo e x è un simbolo di variabile. I simboli $\forall$ e $\exists$ sono detti quantificatori universali ed esistenziali. Una variabile x è detta *vincolata* se è contenuta in una formula del tipo $\forall x(\varphi')$ o $\exists x(\varphi')$ altrimenti è detta *libera*. Una formula è detta *enunciato* se non contiene variabili libere. Una formula è detta *ground* se tutti i suoi letterali sono ground.

Per comodità di scrittura è possibile raggruppare catene di quantificatori dello stesso tipo. Ad esempio, la formula $\forall x_1 \forall x_2 \forall x_3 \exists x_4 \exists x_5 \forall x_6 \forall x_7(\phi)$ può essere scritta come $\forall x_1 x_2 x_3 \exists x_4 x_5 \forall x_6 x_7(\phi)$. Così come visto per $\vee$ e $\wedge$ è possibile vedere $\forall$ e $\exists$ come operatori n-ari che prendono in input n-1 variabili e una formula.

1.2.2 Unificazione

Dato un termine τ (o un letterale), con la scrittura $\tau[x_k/t]$ si indica il termine (o il letterale) ottenuto sostituendo tutte le occorrenze della variabile x_k con il termine t in τ. Ad esempio se $\tau = f(x_1, x_2)$ allora $\tau[x_1/c_1] = f(c_1, x_2)$. Si può estendere questa notazione in modo da poter sostituire più variabili contemporaneamente. Si definisce come *sostituzione* una qualunque funzione da variabili a termini. Dato un termine τ e una sostituzione $\sigma = \{(x_1, t_1), ..., (x_n, t_n)\}$, con la scrittura $\tau[x_1/t_1, ..., x_n/t_n]$ oppure τ^σ si indica il termine ottenuto sostituendo *contemporaneamente* tutte le occorrenze delle variabili $x_1, ..., x_n$ con i termini $t_1, ..., t_n$ in τ. Con contemporaneamente si intende che ogni singola sostituzione viene effettuata sul termine originale, senza essere influenzata dalle sostituzioni precedenti o successive. Ad esempio, se $\tau = f(x_1, x_2)$ allora $\tau[x_1/x_2, x_2/x_1] = f(x_2, x_1)$ e non $f(x_1, x_1)$ che è invece il risultato dell'applicazione sequenziale delle due regole $\tau[x_1/x_2][x_2/x_1]$.

Dati due termini τ_1 e τ_2 si dice che τ_1 è *più generico* di τ_2 o che τ_2 è *più specifico* di τ_1 se e solo se esiste una sostituzione σ tale che $\tau_1^\sigma = \tau_2$. Se esiste una sostituzione σ tale che $\tau_1^\sigma = \tau_2^\sigma$ allora i due termini sono detti *unificabili* e la sostituzione σ è detta *unificatore* dei due termini.

Date due sostituzioni σ_1 e σ_2 si dice che σ_1 è *più generica* di σ_2 o che σ_2 è *più specifica* di σ_1 se e solo se per ogni termine θ, σ_2 è sussunta da σ_1, ossia sse esiste una sostituzione θ tale che $\tau^{\sigma_2} = (\tau^{\sigma_1})^\theta$. Dati due termini unificabili τ_1 e τ_2 si dice che σ_1 è un *MGU* (Most General Unifier) di τ_1 e τ_2 se è la sostituzione più generica tra tutti gli unificatori dei due termini.

È possibile generalizzare il concetto di unificazione per insiemi di termini, letterali e insiemi di letterali. Dato un insieme di termini T, si dice che T è unificabile se esiste una sostituzione σ tale che $\tau_1^\sigma = \tau_2^\sigma$ per ogni coppia di termini $\tau_1, \tau_2 \in T$. In questo caso σ è detto unificatore di T. Due letterali della stessa arità $L_1 = p_1(\tau_1, ..., \tau_n)$ e $L_2 = p_2(\tau'_1, ..., \tau'_n)$ sono unificabili se e solo se esiste una sostituzione che li eguaglia ignorando il simbolo di predicato. In altre parole sia f una funzione di arità n allora L_1 e L_2 sono unificabili se e solo se sono unificabili i termini $f(\tau_1, ..., \tau_n)$ e $f(\tau'_1, ..., \tau'_n)$. Letterali di diversa arità non sono mai unificabili. Un insieme di letterali è unificabile se e solo se esiste una sostituzione che unifica a due a due tutti i letterali dell'insieme.

Un importante risultato è questo:

Proposizione 1. *Se due termini non sono unificabili allora vale una delle seguenti affermazioni:*

1. *I due termini hanno arità diverse*
2. *I due termini presentano una function obstruction*
3. *I due termini presentano una variable obstruction*

Una function obstruction è una situazione in cui visitando allo stesso modo l'albero sintattico dei termini si incontrano due simboli di funzione diversi. Ad esempio, i termini $f_1(x_1, f_2(x_2))$ e $f_1(x_1, f_3(x_2))$ non sono unificabili in quanto presentano una function obstruction, $f_2 \neq f_3$. Una variable obstruction invece è una situazione in cui visitando allo stesso modo l'albero sintattico dei termini si incontra una variabile x nel primo termine e si incontra un sottotermine t ($\neq x$) che contiene x nel secondo termine. Ad esempio i termini $f_1(x_1, f_2(x_2))$ e $f_1(f_2(x_1), x_1)$ non sono unificabili in quanto presentano una variable obstruction, x_1 è contenuta in $f_2(x_1)$.

Un noto algoritmo per la ricerca di un MGU di due termini è l'algoritmo di unificazione di *Robinson*. Il collo di bottiglia di questo algoritmo è la occurrence-check, ovvero la ricerca di una variable obstruction. Se si assume che i termini in

input non contengono variabili in comune è possibile ignorare questa situazione. Di seguito viene riportato l'algoritmo di Robinson senza occurrence-check:

Algorithm 1: Algoritmo di unificazione di Robinson senza occurrence-check

Firma: unify(τ_1, τ_2)
Input: τ_1, τ_2 due termini
Output: σ un MGU di τ_1 e τ_2 o $\perp$ se non esiste
$S :=$ Empty Stack of pair of terms;
$\sigma :=$ Empty Substitution;

$S.push(\tau_1, \tau_2)$;
while *S is not Empty* **do**
  $(\tau_1, \tau_2) := S.pop()$;
  while τ_1 *is a variable* $\wedge\ \tau_1 \neq \tau_1^\sigma$ **do**
    $\tau_1 := \tau_1^\sigma$;
  end
  while τ_2 *is a variable* $\wedge\ \tau_2 \neq \tau_2^\sigma$ **do**
    $\tau_2 := \tau_2^\sigma$;
  end

  if $\tau_1 \neq \tau_2$ **then**
    switch $\tau_1\tau_2$ **do**
      case τ_1 is a variable x and τ_2 is a variabile $y \Rightarrow \sigma := \sigma \cup \{x/y\}$;
      case τ_1 is a variable $x \Rightarrow \sigma := \sigma \cup \{x/\tau_2\}$;
      case τ_2 is a variable $y \Rightarrow \sigma := \sigma \cup \{y/\tau_1\}$;
      case $\tau_1 = f(s_1, ..., s_n)$ and $\tau_2 = f(t_1, ..., t_n) \Rightarrow$
       $S.push(s_1, t_1), ..., S.push(s_n, t_n)$;
      case $\tau_1 = f(s_1, ..., s_n)$ and $\tau_2 = g(t_1, ..., t_m) \Rightarrow$ **return** $\perp$;
    end
  end
end
return σ;

1.2.3 Semantica

Si indica con $type : F' \rightarrow \Sigma_c \cup \Sigma_f \cup \Sigma_p$ la funzione che associa ogni formula φ al suo *tipo*, cioè l'insieme di tutti i simboli non logici presenti nella formula (costanti, funzioni e predicati). Si definisce *Modello* di un tipo Γ una coppia $\mathcal{M} = (D, I)$ dove D è un insieme non vuoto detto *Dominio* e I è una funzione: $I : \Gamma \rightarrow D$ detta d'*Interpretazione* che associa ogni simbolo non logico del tipo a un elemento del dominio in modo tale che per ogni simbolo non logico s:

- Se s è un simbolo di funzione n-aria allora $I(s)$ è una funzione n-aria su D: $I(s) : D^n \to D$.
- Se s è un simbolo di predicato n-ario allora $I(s)$ è una relazione n-aria su D: $I(s) \subseteq D^n$.

Per l'interpretazioni delle costanti si assume che nel dominio siano presenti almeno due oggetti distinti $\{0, 1\} \in D$:

- Se s è un simbolo di costante allora $I(s) \in \{0, 1\}$
- Se s è $\top$ allora $I(s) = 1$
- Se s è $\bot$ allora $I(s) = 0$

Per semplicità di lettura, d'ora in poi l'applicazione della funzione I ad un simbolo s verrà indicata con s^I. Si definisce *Contesto*, una qualunque mappatura $\gamma : \Sigma_x \to D$ che associa variabili a un elemento del dominio. Con la scrittura $\gamma[x/t]$ si indica il contesto ottenuto sostituendo il valore della variabile x con l'elemento t. Ad esempio se $\gamma = \{x_1 \to a, x_2 \to b\}$ allora $\gamma[x_1/b] = \{x_1 \to b, x_2 \to b\}$. Per l'applicazione di un contesto a una variabile si utilizza la stessa notazione usata per le funzioni, ovvero $\gamma(x)$. Per l'esempio precedente $\gamma(x_1) = a$, $\gamma(x_2) = b$ e $\gamma[x_1/b](x_1) = b$. Se una variabile x non è presente nel contesto allora si assume che $\gamma(x) = x$. Si definisce l'interpretazione di un termine o predicato τ nel contesto γ secondo l'interpretazione I:

- Se τ è una variabile x allora $\tau_\gamma^I = \gamma(x)$
- Se τ è una costante c allora $\tau_\gamma^I = c^I$ (Le costanti non dipendono dal contesto)
- Se τ è una funzione $f(\tau_1, ..., \tau_n)$ allora $\tau_\gamma^I = f^I({\tau_1}_\gamma^I, ..., {\tau_n}_\gamma^I)$
- se τ è un predicato $p(\tau_1, ..., \tau_n)$ allora $\tau_\gamma^I = p^I({\tau_1}_\gamma^I, ..., {\tau_n}_\gamma^I)$

Data una formula del primo ordine φ si dice che il modello $\mathcal{M}$ è appropriato per la formula φ se e solo se il tipo della formula è contenuto nel tipo del modello. Un modello appropriato $\mathcal{M}$ soddisfa una formula φ nel contesto γ se e solo se:

- Se φ è una costante c (o $\top/\bot$) allora $\mathcal{M}, \gamma \models c$ se e solo se $c^I = 1$
- Se φ è della forma $\neg\psi$ (dove ψ è una formula) allora $\mathcal{M}, \gamma \models \varphi$ se e solo se $\mathcal{M}, \gamma \not\models \psi$
- Se φ è della forma $(\psi \wedge \chi)$ (con ψ e χ formule) allora $\mathcal{M}, \gamma \models \varphi$ se e solo se $\mathcal{M}, \gamma \models \psi$ e $\mathcal{M}, \gamma \models \chi$
- Se φ è della forma $(\psi \vee \chi)$ (con ψ e χ formule) allora $\mathcal{M}, \gamma \models \varphi$ se e solo se $\mathcal{M}, \gamma \models \psi$ o $\mathcal{M}, \gamma \models \chi$
- Se φ è della forma $\forall x(\psi)$ (dove ψ è una formula) allora $\mathcal{M}, \gamma \models \varphi$ se e solo se per ogni elemento $m \in D$ vale $\mathcal{M}, \gamma[x/m] \models \psi$

- Se φ è della forma $\exists x(\psi)$ (dove ψ è una formula) allora $\mathcal{M}, \gamma \models \varphi$ se e solo se esiste un elemento $m \in D$ tale che $\mathcal{M}, \gamma[x/m] \models \psi$
- Infine se φ è un letterale $p(\tau_1, ..., \tau_n)$ allora $\mathcal{M}, \gamma \models \varphi$ se e solo se $p^I({\tau_1}^I_\gamma, ..., {\tau_n}^I_\gamma)$

Data una formula φ si dice che un modello $\mathcal{M}$ soddisfa φ o anche che $\mathcal{M}$ è un modello di φ se e solo se $\mathcal{M}$ è appropriato per φ e $\mathcal{M}, \gamma \models \varphi$ per ogni contesto γ, in notazione $\mathcal{M} \models \varphi$. Una formula è detta *soddisfacibile* se esiste un modello che la soddisfa. Una formula è detta *valida* se ogni modello la soddisfa. La relazione di conseguenza logica, equivalenza, equisoddisfacibilità e soundness sono definite in modo analogo alla logica proposizionale.

Così come è stata posta questa semantica nessuna formula con variabili libere può avere un modello. Ad esempio se si considera una formula φ con una variabile libera x e si considera un modello $\mathcal{M}$ appropriato per φ, è sufficiente prendere un contesto vuoto $\gamma = \{\}$ per ottenere $\mathcal{M}, \gamma \not\models \varphi$. Per aggirare questa limitazione si estende la definizione di soddisfacibilità per formule con variabili libere. Dato un modello $\mathcal{M}$ appropriato ad una formula φ si dice che $\mathcal{M}$ soddisfa φ se e solo se

- Se la formula non contiene variabili libere allora la definizione di soddisfacibilità rimane invariata.
- Se φ contiene variabili libere allora $\mathcal{M} \models \varphi$ se e solo se per ogni contesto γ che contiene tutte le variabili libere di φ vale $\mathcal{M}, \gamma \models \varphi$.

Con questa modifica non vi è differenza nel significato tra enunciati del tipo $\forall x(\varphi)$ e formule φ, quindi da questo momento in poi ogni variabile libera presente in una formula verrà considerata come vincolata da un quantificatore universale posto all'inizio della formula.

Un'altra osservazione interessante è che se nella formula sono presenti esclusivamente costanti e simboli logici allora il modello si comporta esattamente come un'assegnazione proposizionale. È possibile estendere questa considerazione anche alle formule ground, ovvero formule senza variabili. Infatti se ogni predicato ground viene sostituito da un nuovo simbolo di costante allora la formula proposizionale risultante si comporterà, in termini di soddisfacibilità, esattamente come la formula originale. Ad esempio la formula ground $(p_1(c_1) \vee p_2(c_2, f_1(c_1))) \wedge \neg p_1(c_1) \wedge c_2$ è esattamente equivalente alla formula proposizionale $(c_3 \vee c_4) \wedge \neg c_3 \wedge c_2$, nel senso che per ogni assegnamento proposizionale per la seconda formula esiste un modello per la prima e viceversa.

1.2.4 Skolemizzazione e Forme Normali

In questo paragrafo verrà descritta una procedura fondamentale per la dimostrazione automatica di teoremi, la *Skolemizzazione*. Varrà inoltre introdotta una

nuova forma normale chiamata *PNF* e verranno estese le forme normali descritte nel paragrafo della logica proposizionale per adattarle alla logica del primo ordine.

La definizione per le forme ENNF e NNF per la logica del primo ordine è pressoché identica a quella della logica proposizionale. Come per la logica proposizionale, la trasformazione di una formula in ENNF/NNF preserva la relazione di conseguenza logica ed è sempre possibile trasformare una formula in una equivalente in ENNF/NNF. Il calcolo per la normalizzazione viene effettuato allo stesso modo ma con l'aggiunta di due regole per la negazione dei quantificatori:

$$\neg\forall x(\varphi) \to \exists x(\neg\varphi)$$
$$\neg\exists x(\varphi) \to \forall x(\neg\varphi)$$

Chiameremo *prefisso di quantificatori* una lista di quantificatori (es. $\forall x_1 \forall x_2 \exists x_3$). Un prefisso viene detto *universale* se è composto esclusivamente da quantificatori universali e viene detto *esistenziali* se è composto esclusivamente da quantificatori esistenziali. Una formula è in formato PNF (Prenex Normal Form) se tutti e soli i quantificatori si trovano all'inizio della formula. La classe di formule PNF è generata dalla seguente grammatica:

$$P_0 := \rho(P)$$
$$P := \top \mid \bot \mid A \mid \neg P \mid (P \wedge P) \mid (P \vee P)$$

Dove ρ è un prefisso di quantificatori e A è un atomo. La parte della formula generata dalla seconda regola viene spesso chiamata *matrice*. È sempre possibile normalizzare una formula in PNF, è un processo sound, ma non è sempre possibile mantenere la relazione di conseguenza logica.

La skolemizzazione è una procedura che permette di eliminare i quantificatori esistenziali da una formula. Sia ρ un prefisso di quantificatori qualunque, la funzione $sk : F' \to F'$ può essere descritta in questo modo:

- Se la formula è del tipo $\exists x(\rho\phi)$ allora sk rimuove il primo quantificatore esistenziale e sostituisce la variabile x all'interno della formula con una nuova costante c_{n+1}, dove n è il massimo indice di costante presente nella formula.
- Se la formula è del tipo $\forall x_k ... x_{k+m-1} \exists x_{k+m}(\rho\phi)$ allora sk rimuove il primo quantificatore esistenziale e sostituisce la variabile x_{k+m} all'interno della formula con una nuova funzione $f_{n+1}(x_k, ..., x_{k+m-1})$ $m-1$-aria, dove n è il massimo indice di funzione presente nella formula.

Applicando la funzione sk tante volte quanto il numero di quantificatori esistenziali presenti nella formula si ottiene una formula senza quantificatori esistenziali. Anche la skolemizzazione è un processo sound, ma non è detto che preservi la conseguenza logica.

Combinando le tecniche apprese finora è possibile definire una procedura di normalizzazione che permette di trasformare una formula del primo ordine in formato CNF. Le formule CNF per il primo ordine sono definite come segue:

$$\zeta_0 := \rho(\zeta_1)$$
$$\zeta_1 := \xi \mid (\xi \wedge \zeta_1)$$
$$\xi := \top \mid \bot \mid L \mid (\xi \vee \xi)$$

Dove ρ è un prefisso di quantificatori universale e L un letterale. Per ottenere una formula CNF è sufficiente:

1. Normalizzare la formula in NNF
2. Normalizzare la formula in PNF
3. Skolemizzare la formula
4. Applicare lo stesso algoritmo di clausificazione descritto per la logica proposizionale sulla matrice della formula

Visto le tecniche applicate il processo di clausificazione per la logica del primo ordine è sound ma non preserva la conseguenza logica. Dato che in una formula CNF tutte le variabili sono universalmente quantificate per brevità è possibile omettere il prefisso di quantificatori. Ad esempio, la formula CNF $\forall x_1 x_2 x_3 x_4(\neg p_1(x_1) \vee p_2(x_2) \vee p_3(x_3)) \wedge (\neg p_4(x_4))$ può essere scritta come $(\neg p_1(x_1) \vee p_2(x_2) \vee p_3(x_3)) \wedge (\neg p_4(x_4))$ e quindi rappresentata in forma insiemistica come $\{\{\neg p_1(x_1), p_2(x_2), p_3(x_3)\}, \{\neg p_4(x_4)\}\}$. Un'osservazione interessante è che visto che tutte le variabili sono universalmente quantificate, è possibile rinominare le variabili di clausole diverse senza cambiare il significato della formula. È quindi possibile normalizzare le clausole in modo tale che ogni coppia di clausole contenga variabili diverse. Un caso d'uso tipico è quello di voler unificare due letterali di due clausole diverse. Con l'assunzione che le variabili siano tutte diverse è possibile applicare l'algoritmo di unificazione senza occurrence-check visto nel capitolo sull'Unificazione.

1.3 Soddisfacibilità e Validità

1.4 Resolution

1.5 Il formato TPTP

In questa sezione verrà descritto il formato TPTP (Thousands of Problems for Theorem Provers) per la rappresentazione di problemi di logica del primo ordine. TPTP è una nota libreria di problemi utilizzata per testare e valutare diversi ATP systems (Automatic Theorem Prover). TPTP fornisce diversi formati per la rappresentazione dei problemi, in questa sezione ci si soffermerà sui formati *CNF* (Clausal Normal Form) e *FOF* (First Order Formula).

La traduzione dei predicati, termini e variabili è la stessa sia per il formato CNF che per il formato FOF. Ogni variabile è rappresentata da una stringa alfanumerica Maiuscola. Simboli di funzione, predicato e costanti sono tutti rappresentati senza distinzione da stringhe alfanumeriche minuscole. Le regole della grammatica della generazione dei termini è esattamente la stessa vista nel paragrafo 1.2.1. Per esempio il predicato $p_1(f_1(x_1), x_2, c_1)$ può essere rappresentato come `p1(f1(X1), X2, c1)` o anche `pred(fun(VAR_A), VAR_B, costante)`.

Per la traduzione dei simboli logici si utilizza la seguente mappatura:

Simbolo	Traduzione
$\top$	`$true`
$\bot$	`$false`
$\neg$	`~`
$\wedge$	`&`
$\vee$	`|`
$\Rightarrow$	`=>`
$\Leftrightarrow$	`<=>`
$\oplus$	`<~>`
$\forall$	`!`
$\exists$	`?`

Tabella 1.1: Traduzione dei simboli logici

Anche le regole per la generazione delle formule sono le stesse viste nella sezione 1.2.1, con l'unica differenza che i simboli logici vengono tradotti secondo la tabella 1.1. Le parentesi '(' e ')' possono essere omesse e in tal caso si segue il seguente ordine di valutazione dei simboli: `!`, `?`, `~`, `&`, `|`, `<=>`, `=>`, `<~>`. Dopo un quantificatore (`!`/`?`) è necessaria la lista delle variabili quantificate racchiuse tra parentesi quadre '[' e ']' seguite da ':' e la formula quantificata. Per esempio la formula $\forall x_1 x_2 \exists x_3 (p_1(x_1) \vee p_2(x_2) \vee p_3(x_3))$ viene rappresentata come `![X1, X2] : (?[X3] : (p1(X1) | p2(X2) | p3(X3)))`. Se non presenti quantificatori nella formula le variabili libere vengono considerate quantificate universalmente.

Il formato FOF prevede una lista di assiomi seguiti da una congettura. Il formato cambia a seconda della domanda che si vuole porre all'ATP. Data una lista di assiomi $A_1, ..., A_n$ e una congettura C:

- Se si da in input la lista di assiomi $A_1, ..., A_n$ l'ATP cerca di determinare la Soddisfacibilità della formula $A_1 \wedge ... \wedge A_n$.
- Se vengono dati sia gli assiomi che la congettura l'ATP cerca di determinare se $A_1 \wedge ... \wedge A_n \Rightarrow C$ è valida.
- Se invece viene data solo la congettura l'ATP cerca di determinare se C è valida.

Il formato per inserire un assioma o la congettura è il seguente:

```
fof(<nome>, <tipo>, <formula>).
```

'Nome' è una stringa alfanumerica che identifica la formula, 'tipo' può essere `axiom` per gli assiomi e `conjecture` per la congettura. 'Formula' è una formula del primo ordine in formato FOF. Ad esempio con il file di input:

```
fof(ax1, axiom, p(X)).
fof(ax2, axiom, p(X) => q(X)).
fof(conj, conjecture, q(X)).
```

L'ATP cercherà di determinare se la formula $(p(X) \wedge (p(X) \Rightarrow q(X))) \Rightarrow q(X)$ è valida. Per le formula CNF invece il formato è il seguente:

```
cnf(<nome>, axiom, <clausola>).
```

Dove 'nome' è definito come per il formato FOF e 'clausola' è una clausola del primo ordine. L'unico tipo consentito è `axiom` e non è possibile inserire una congettura. In questo formato ogni clausola deve essere scritta in un'annotazione separata. Ad esempio lo stesso problema dell'esempio precedente può essere posto all'ATP in questo modo:

```
cnf(ax1, axiom, p(X)).
cnf(ax2, axiom, ~p(X) | q(X)).
cnf(conj, axiom, ~q(X)).
```

Capitolo 2

Algoritmo di decisione di Frammenti Binding

2.1 Classificazione

2.2 Algoritmo Astratto

Capitolo 3

Il Theorem prover Vampire

3.1 I Termini

3.2 Formule e Clausole

3.3 Unificazione e Substitution Trees

3.4 Preprocessing

3.5 Saturazione e Refutazione

3.6 Il SAT-Solver

3.7 Misurazione dei Tempi

3.8 Opzioni

Capitolo 4

Implementazione di procedure di decisione per frammenti Binding in Vampire

L'algoritmo di decisione, la classificazione, Il preprocessing

4.1 Algoritmo di Classificazione

4.2 Preprocessing

4.2.1 Boolean Top Formula

4.2.2 Forall-And

4.2.3 SAT-Clausification

4.3 Procedura di Decisione

4.3.1 Implicants Sorting

4.3.2 Maximal Unifiable Subsets

4.3.3 Algoritmo Finale

Capitolo 5

Analisi Sperimentale

5.1 La libreria TPTP

5.2 Analisi dei risultati

5.3 Ottimizzazioni

5.4 Conclusioni e Possibili Sviluppi futuri