UNIVERSITÀ DEGLI STUDI DI MILANO

Dipartimento di Scienze dell'Informazione

Introduzione alla logica proposizionale e alla procedura DPLL

Silvio Ghilardi

Enrica Nicolini, Daniele Zucchelli

Osservazione. Questi appunti sono destinati ad un uso didattico in corsi di base per studenti delle lauree triennali di classe Informatica. Per questo motivo tutte le dimostrazioni (peraltro piuttosto semplici) sono state omesse e per importanti euristiche avanzate si è semplicemente fatto rimando alla letteratura. Il materiale del Capitolo 1 riprende il contenuto dell'analogo capitolo della dispensa in formato elettronico *Corso Propedeutico di Logica* (a cura di M. Franchella, S. Ghilardi, L. Sacchetti) utilizzata fino all'anno accademico 2005-06. Questa versione della dispensa porta la data del **7 marzo 2007**.

Indice

Capitolo 1

La Logica Proposizionale

1.1 Linguaggi Proposizionali

Con il termine **enunciato** intendiamo una una qualsiasi proposizione per la quale sia sensato chiedersi se sia vera o falsa. Ad esempio, sono enunciati proposizioni come "Paolo corre", "Laura ha i capelli rossi", "Piove e c'è vento", "Se c'è il sole, allora vado al mare". Non sono enunciati invece proposizioni come "C'è il sole, vero?", "Paolo, torna a casa presto!", il cui contenuto non consiste in una mera affermazione.

Informalmente, diciamo poi che un enunciato è un **enunciato semplice** o **enunciato atomico** se non contiene nessun altro enunciato come sua parte propria. Diciamo invece che un enunciato è un **enunciato composto** se contiene altri enunciati, cioè se è possibile scomporlo in enunciati più semplici.

Ad esempio, sono enunciati atomici le proposizioni "Paolo corre" e "Laura ha i capelli rossi"; sono invece enunciati composti gli enunciati "Piove e c'è vento", "Se c'è il sole vado al mare" e "Carla ha gli occhi neri o Carla ha i capelli neri" (per vedere che "Piove e c'è vento" è composto è sufficiente osservare che "Piove e c'è vento" contiene come sue parti proprie i due enunciati "Piove" e "C'è vento".)

Le parole "e", "o", "se...allora", "non", "ma" si dicono **connettivi**. I connettivi permettono quindi di ottenere enunciati da altri enunciati, cioè permettono di ottenere enunciati composti mediante enunciati più semplici.

Consideriamo l'enunciato composto "Piove e c'è vento". Se associamo all'enunciato

atomico "piove" la lettera p, all'enunciato atomico "c'è vento" la lettera q e indichiamo con il carattere $\wedge$ il connettivo "e", possiamo associare all'enunciato "Piove e c'è vento" una espressione formale del tipo $p \wedge q$. L'espressione $p \wedge q$ è un esempio di *formula*, in generale le formule saranno particolari espressioni ottenute da un dato insieme di simboli, chiamati lettere proposizionali, mediante un'opportuna applicazione dei connettivi. Le formule costituiscono la rappresentazione (certamente semplificata ed impoverita) in un linguaggio artificiale del contenuto concettuale degli enunciati del linguaggio naturale. Definiamo ora in maniera precisa il concetto di formula.

Un **linguaggio proposizionale** $\mathcal{L}$ è semplicemente un insieme, i cui elementi si dicono **lettere proposizionali** e vengono indicati con i caratteri $p, q, r, \ldots$ eventualmente muniti di indici o apici. Su $\mathcal{L}$ non facciamo restrizioni di sorta, anche se spesso nei testi si assume che $\mathcal{L}$ sia numerabile.

Utilizzando gli elementi di $\mathcal{L}$, le parentesi $(,)$ e i caratteri relativi ai connettivi proposizionali $\wedge$ (e), $\vee$ (o), $\rightarrow$ (se...allora), $\neg$ (non), siamo in grado di scrivere tutte le possibili stringhe di simboli come ad esempio

$$(*) \qquad p \vee \neg qqq, \quad \neg\neg)p\wedge, \quad (\neg(p \vee q)), \ \ldots$$

Non tutte queste stringhe di simboli si considerano accettabili da un punto di vista grammaticale, ossia non tutte verranno considerate $\mathcal{L}$-formule. Sono $\mathcal{L}$- formule o, più brevemente, **formule** solo quelle stringhe di simboli che sono ottenute applicando un numero finito di volte le seguenti regole di formazione:

(i) ogni $p \in \mathcal{L}$ è una formula (detta **formula atomica**);

(ii) se A_1 e A_2 sono formule, tali sono $(A_1 \wedge A_2)$, $(A_1 \vee A_2)$, $(A_1 \rightarrow A_2)$, $(\neg A_1)$.

In termini del tutto equivalenti, si può dire così: una stringa di simboli è una $\mathcal{L}$-formula qualora compaia all'ultimo posto di una $\mathcal{L}$-*costruzione*, dove una $\mathcal{L}$-costruzione è una lista di stringhe di simboli ciascun elemento della quale o è una stringa fatta di un solo carattere (per di più appartenente a $\mathcal{L}$) o è ottenuto da elementi della lista che lo precedono secondo quanto prescritto in (ii). Ad esempio, la $\mathcal{L}$-costruzione

$$p, \quad q, \quad (p \vee q), \quad (\neg(p \vee q))$$

testimonia che $(\neg(p \vee q))$ è una formula. Invece la prima e la seconda stringa menzionate in $(*)$ non sono formule perchè non esiste una $\mathcal{L}$-costruzione che termini in esse (provate a costruirne una e vedrete che è impossibile!).

Le formule vengono indicate con le lettere $A, B, \ldots$ (dette *metavariabili*) a loro volta munite di indici o apici. L'insieme di tutte le formule viene indicato con $F_{\mathcal{L}}$.

Stipuliamo le seguenti convenzioni di notazione.

- L'introduzione di molte parentesi nella definizione di formula serve per avere *unicità di lettura*, ossia per poter ricostruire in modo univoco il procedimento di costruzione delle formule stesse; per non appesantire la trattazione non ci addentriamo in ulteriori dettagli, ci basta segnalare che solo l'unicità di lettura dà modo di procedere in modo non ambiguo in molte circostanze. In particolare, l'unicità di lettura consente, data una formula non atomica, di stabilire univocamente qual è il suo **connettivo principale**, ossia qual è l'ultimo connettivo introdotto nella costruzione della formula stessa. Ad esempio, il connettivo principale di $((p \wedge q) \rightarrow (q \vee p))$ è $\rightarrow$.

- L'uso delle parentesi non verrà però rispettato alla lettera come sarebbe previsto nella definizione di formula, ad esempio le parentesi esterne saranno di regola omesse. Per alleggerire la scrittura, stipuliamo anche che $\neg$ lega più strettamente di $\wedge, \vee$, che a loro volta legano più strettamente di $\rightarrow$. Così ad esempio $\neg p \rightarrow q \vee r$ sta per $((\neg p) \rightarrow (q \vee r))$. Invece $p \wedge q \vee r$ è ambiguo e dovremo sempre mettere le parentesi per leggerlo come $(p \wedge q) \vee r$ o come $p \wedge (q \vee r)$.

- Abbreviazioni come $p \wedge q \wedge r$ saranno usate, ponendo convenzionalmente che $p_1 \wedge \cdots \wedge p_n$ stia per $(p_1 \wedge (p_2 \wedge (\cdots (p_{n-1} \wedge p_n) \cdots)))$. La stessa convenzione vale per $\vee$, ma nessuna convenzione di tal tipo vale per $\rightarrow$, per cui scritture come $p \rightarrow q \rightarrow r$ non vengono ammesse.

- $A \leftrightarrow B$ sta per $(A \rightarrow B) \wedge (B \rightarrow A)$: il connettivo $\leftrightarrow$ si chiama 'bi-implicazione' o 'equivalenza materiale'. Si potrebbero in realtà eliminare altri connettivi dalla lista dei connettivi primitivi e introdurli come abbreviazioni, la scelta che abbiamo fatto non è minimale in questo senso.

1.2 Funzioni di Verità

Una formula denota un'asserzione che, in una data situazione specifica, risulta essere vera o falsa (ma non contemporaneamente vera e falsa). In effetti, una volta noto il valore di verità dei suoi costituenti, si può determinare meccanicamente in modo agevole il valore di verità di tutta una formula: questo perchè i nostri connettivi saranno analizzati in un'ottica **vero-funzionale**, ossia come funzioni che hanno sia in ingresso che in uscita dei valori di verità. Definire il significato di un connettivo significherà quindi semplicemente specificare sotto quali condizioni è vero o falso un enunciato che contiene il connettivo in esame come connettivo principale.

La semantica che proponiamo è **bivalente**, cioè i valori di verità sono due, il vero e il falso, li indichiamo con T, F ('True', 'False') o con $1, 0$. Analizziamo ora il significato dei nostri connettivi.

• L'enunciato "Piove e c'è vento" è vero se e solo se sono veri entrambi gli enunciati componenti "Piove" e "c'è vento". Ricordiamo che il connettivo "e" si chiama **congiunzione** e si indica con il simbolo $\wedge$. Dunque se indichiamo con p l'enunciato "Piove" e con q l'enunciato "c'è vento", possiamo formalizzare l'ununciato "Piove e c'è vento" mediante la formula $p \wedge q$. Per quanto detto la formula $p \wedge q$ è vera se e solo se sia p che q sono vere.

In generale, se A e B indicano enunciati qualsiasi, la formula $A \wedge B$ è vera se e solo se sia A che B sono vere. Possiamo riassumere quanto detto mediante la seguente tabella, detta **tavola di verità per il connettivo** $\wedge$:

A	B	$A \wedge B$
T	T	T
F	T	F
T	F	F
F	F	F

• L'enunciato "Carla ha gli occhi neri o Carla ha i capelli neri" invece è vero quando almeno uno degli enunciati "Carla ha gli occhi neri " e "Carla ha i capelli neri" è vero. Ricordiamo che il connettivo "o" si chiama **disgiunzione** e si indica con il simbolo $\vee$. Nel linguaggio naturale esistono almeno due usi diversi della disgiunzione, quello inclusivo

(corrispondente al latino 'vel') e quello esclusivo (corrispondente al latino 'aut'). I due usi differiscono per la valutazione del caso in cui entrambi i disgiunti siano veri: nell'uso inclusivo la disgiunzione corrispondente viene considerata vera, nell'uso esclusivo viene considerata falsa. Scegliamo l'uso inclusivo della disgiunzione che prevale nella letteratura (tale scelta non pregiudica comunque l'espressività del linguaggio in quanto la disgiunzione esclusiva risulta ottenibile dalla combinazione $(A \vee B) \wedge \neg(A \wedge B)$).

In generale, se A e B indicano formule qualsiasi, la formula $A \vee B$ è vera se e solo se o A o B è vera. Possiamo riassumere quanto detto mediante la **tavola di verità per** $\vee$:

A	B	$A \vee B$
T	T	T
F	T	T
T	F	T
F	F	F

• L'enunciato "Non piove" è vero quando è falso l'enunciato "piove". Ricordiamo che il connettivo "non" si chiama **negazione** e si indica con il simbolo $\neg$.

In generale, se A indica una formula qualsiasi, la formula $\neg A$ è vera se e solo se A è falsa. La **tavola di verità per** $\neg$ è quindi:

A	$\neg A$
T	F
F	T

Questo uso della negazione viene dalla tradizione aristotelica e caratterizza la logica classica (che forma l'oggetto delle presenti note). Segnaliamo che nel corso della storia ed in particolare nel XX secolo sono state proposte differenti ed interessanti analisi della negazione, sia da un punto di vista sintattico che semantico.

• Passiamo infine al connettivo "se ... allora". Questo connettivo si chiama **implicazione** o **condizionale** e si indica con $\rightarrow$. Consideriamo l'enunciato "Se a è un numero pari, allora $a + 1$ è un numero dispari". Certamente non vogliamo considerare come controesempi a tale asserzione casi di numeri a che non siano pari. D'altra parte, gli unici controesempi possibili sarebbero quelli dati da numeri a che fossero pari e per cui $a+1$ non fosse dispari. Siccome non è possibile trovare un tale numero, siamo portati a concludere

che l'enunciato in questione è vero. Risulta quindi immediato stipulare che, se A e B indicano enunciati qualsiasi, la formula $A \to B$ è falsa solamente quando A è vera e B è falsa. Possiamo riassumere quanto detto mediante la **tavola di verità per** $\to$:

A	B	$A \to B$
T	T	T
F	T	T
T	F	F
F	F	T

Certamente questo è un uso dell'implicazione che è giustificato in alcuni ambiti matematici (come quello sopra illustrato), ma che tuttavia è ben lungi dal render conto della ricchezza dell'implicazione nel linguaggio naturale, dove sono presenti molteplici significati ad esempio di natura causale o controfattuale. Per l'analisi di tali significati occorrono strumenti logici più complessi, che esulano dal presente corso introduttivo.

Nella formula $A \to B$, la sottoformula A si dice *antecedente* o *premessa* dell'implicazione, mentre la sottoformula B si dice *conseguente* o *conclusione* dell'implicazione.

Possiamo riassumere in maniera sintetica quanto detto sopra come segue. Sia

$$\mathbf{2} := \{1, 0\}$$

l'insieme dei valori di verità. Chiamiamo interpretazione (o **assegnamento**) una qualunque funzione

$$V : \mathcal{L} \longrightarrow \mathbf{2}.$$

V si estende a $F_{\mathcal{L}}$ per induzione, utilizzando le tavole di verità sopra descritte. L'induzione è sul numero di connettivi della formula di cui si vuole calcolare il valore di verità sotto l'assegnamento V; formalmente, si pone:

- $V(A \wedge B) := min(V(A), V(B))$;
- $V(A \vee B) := max(V(A), V(B))$;
- $V(\neg A) := 1 - V(A)$;
- $V(A \to B) := max(1 - V(A), V(B))$.

Per effetto delle definizioni date, è facile vedere che $V(A \leftrightarrow B) = 1$ se e solo se $V(A) = V(B)$.

Una formula A è una **verità logica** (o **tautologia**) qualora si abbia $V(A) = 1$ *per ogni* possibile V. Dunque una tautologia è una proposizione che è sempre vera (cioè tale che il suo valore di verità è sempre T). A è una **contraddizione** (o è **refutabile**) qualora valga $V(A) = 0$ per ogni V. Dunque una contraddizione è una formula che risulta sempre falsa. Infine A è **soddisfacibile** se e solo se[1] $V(A) = 1$ *per almeno un* V.

Per valutare se una data formula A sia o meno una tautologia, una contraddizione, ecc. si può fare una tabella (detta tavola di verità) che calcoli $V(A)$ per tutti i possibili V. Ovviamente, è irrilevante il valore di V sulle lettere proposizionali che non compaiono in A, perciò se A contiene n lettere proposizionali saranno prese in considerazione solo 2^n interpretazioni possibili. Questo metodo è oneroso, non sono noti (e probabilmente non esistono neppure) metodi veloci che operino nella totalità dei casi; spesso però nella pratica è inutile far passare tutte le interpretazioni possibili e ci sono vari algoritmi meno brutali dell'esame di tutta la tavola (ne analizzeremo uno nel prossimo capitolo).

Facciamo un esempio di esame completo della tavola di verità di una formula; consideriamo l'enunciato

"Se piove o c'è vento allora Marco non esce di casa".

Gli enunciati atomici componenti di tale enunciato sono "Piove", "C'è vento" e "Marco esce di casa". Indichiamo con p l'enunciato atomico "Piove", con q l'enunciato atomico "C'è vento", con r l'enunciato atomico "Marco esce di casa". Possiamo quindi formalizzare l'enunciato "Se piove o c'è vento allora Marco non esce di casa" mediante la formula

$$(p \vee q) \rightarrow \neg r.$$

Costruiamo la tavola di verità di questa formula applicando successivamente le tavole di

[1] D'ora in poi abbreviamo 'se e solo se' con 'sse'.

verità dei connettivi:

p	q	r	$p \vee q$	$\neg r$	$(p \vee q) \rightarrow \neg r$
T	T	T	T	F	F
F	T	T	T	F	F
T	F	T	T	F	F
F	F	T	F	F	T
T	T	F	T	T	T
F	T	F	T	T	T
T	F	F	T	T	T
F	F	F	F	T	T

Osserviamo che ad ogni assegnamento di verità T o F alle lettere proposizionali corrisponde un valore di verità della formula nel suo complesso, e questo valore di verità si ottiene applicando via via le tavole di verità per i connettivi a tutte le sottoformule. Inoltre, come già osservato, se in una formula vi sono n lettere proposizionali diverse allora si hanno 2^n possibili assegnamenti di verità a tali lettere proposizionali e quindi 2^n righe nella tavola di verità. In questo modo vengono presi in esame tutti i casi possibili. Quindi, ad esempio, se compaiono 3 lettere proposizionali (come in $(p \vee q) \rightarrow \neg r$) vi saranno 8 righe nella tavola di verità, se compaiono 2 lettere proposizionali (come in $(p \vee q)$) vi saranno 4 righe nella tavola di verità, e via dicendo.

Dallo studio della tavola di verità per $(p \vee q) \rightarrow \neg r$ otteniamo che la formula $(p \vee q) \rightarrow \neg r$ è soddisfacibile (in quanto esiste un assegnamento che la rende vera), dunque, in particolare, $(p \vee q) \rightarrow \neg r$ non è una contraddizione. Siccome però esiste un assegnamento che rende falsa $(p \vee q) \rightarrow \neg r$, si ha che $(p \vee q) \rightarrow \neg r$ non è una tautologia.

Invece, si può verificare che $p \wedge q \rightarrow q$ e $p \rightarrow (\neg p \rightarrow q)$ sono tautologie, mentre formule come $p \wedge \neg p$ e $p \wedge \neg(p \vee q)$ sono contraddizioni.

Capitolo 2

La procedura DPLL

2.1 Generalità

La procedura di Davis-Putnam-Logemann-Loveland (DPLL) rappresenta probabilmente il metodo più efficace per affrontare il problema (computazionalmente intrattabile) della soddisfacibilità di una formula della logica proposizionale classica. Tale procedura fu introdotta in due articoli (a firma di Davis-Putnam e di Davis-Logemann-Loveland, rispettivamente) apparsi nei primi anni 60; il successo della procedura è principalmente dovuto alla impressionante opera di implementazione e di ingegnerizzazione che è stata condotta via via negli anni e che continua ancor oggi.

Descritta in modo schematico e sommario, la procedura tenta di costruire (mediante un meccanismo di backtracking) un assegnamento V che soddisfi una data formula A; la formula A viene usualmente preprocessata in modo da ridurla ad un insieme di clausole. Ad ogni nodo di scelta, la procedura assegna un valore di verità ad una lettera proposizionale; prima, però, esegue tutte le operazioni deterministiche utili a *propagare* le informazioni già acquisite.

La DPLL si differenzia significativamente dalle procedure classiche centrate su tableaux o sequenti, in quanto basa le sue scelte sull'assegnamento di valori di verità a lettere proposizionali, anziché a sottoformule. Si differenzia anche dalle pure tavole di verità per l'utilizzo accorto e massimizzato del meccanismo di propagazione.

La procedura in quanto tale è molto flessibile, ma solo se completata con opportune *euristiche* risulta davvero efficace nelle applicazioni. Tali euristiche consistono ad esempio in adeguate selezioni della lettera proposizionale su cui operare la scelta non deterministica

di un assegnamento, ma soprattutto in meccanismi di *backtracking non cronologico* e in appropriate *strategie di apprendimento* dai tentativi di ricerca falliti.

Esporremo uno *schema-base* della procedura DPLL, che prescinde dalle euristiche sopra accennate. Si osservi innanzitutto che la DPLL in quanto tale (a meno di farne estensioni ad hoc) non opera su tipi di dati che siano formule, ma solo su insiemi di clausole. Il suo utilizzo necessita quindi di una fase di *preprocessamento*.

Chiamiamo **atomi** le formule atomiche, **letterali** le formule atomiche e le atomiche negate, **clausole** le disgiunzioni di letterali. Con $\Box$ viene indicata la **clausola vuota** (la clausola vuota è interpretata come una formula insoddisfacibile).

Esempio 1. p, q, r sono atomi, p, $\neg r$, q sono letterali, $p \vee \neg r$, $p \vee q$, $\neg p \vee \neg q$ sono clausole.

Due formule A, B sono **logicamente equivalenti** se e solo se $A \leftrightarrow B$ è una tautologia (ossia se e solo se vale $V(A) = V(B)$ per ogni assegnamento V).

Due formule sono **equisoddisfacibili** se e solo se (A è soddisfacibile se e solo se lo è B).

Esempio 2. $\neg(p \wedge q)$ e $\neg p \vee \neg q$ sono logicamente equivalenti; $\neg(p \wedge q)$ e $\neg p \wedge \neg q$ sono equisoddisfacibili ma non logicamente equivalenti; $\neg p \wedge p$ e $p \rightarrow q$ non sono nemmeno equisoddisfacibili (perchè la prime di esse non è soddisfacibile mentre la seconda sì).

Una formula è in **forma normale negativa** (**fnn**) se e solo se non contiene implicazioni e contiene negazioni solo di fronte a sottoformule atomiche. Una formula è in **forma normale congiuntiva** (**fnc**) se e solo se è una congiunzione di clausole.

Esempio 3. $(\neg p \vee r) \wedge (\neg q \vee \neg r)$ è in forma normale congiuntiva; $(\neg p \wedge q) \vee (\neg p)$ è in forma normale negativa, ma non in forma normale congiuntiva.

La procedura che proporremo per scoprire se una data formula A è o meno soddisfacibile consta di tre fasi:

1. A viene trasformata in A' che è in forma normale negativa ed è logicamente equivalente ad A;

2. A' viene trasformata in A'' che è in forma normale congiuntiva ed è equisoddisfacibile con A;

3. sull'insieme $\mathcal{C}$ di clausole di cui A'' è congiunzione agisce la DPLL: se la DPLL trova un assegnamento per $\mathcal{C}$, la formula originaria A risulterà soddisfacibile, altrimenti no.

Osservazione importante. Se si usa la procedura per scoprire se una data formula B è una *tautologia*, occorre una fase zero, in cui **si nega** B e si pone $A := \neg B$.

2.2 Forme Normali Negative

Per trasformare una formula A in una formula A' in fnn *logicamente equivalente* ad A, è sufficiente eseguire (**in ordine qualunque, ma in modo esaustivo**) le seguenti trasformazioni:

$$\begin{aligned} C \to D &\rightsquigarrow \neg C \vee D \\ \neg\neg C &\rightsquigarrow C \\ \neg(C \vee D) &\rightsquigarrow \neg C \wedge \neg D \\ \neg(C \wedge D) &\rightsquigarrow \neg C \vee \neg D. \end{aligned}$$

Le trasformazioni vanno viste come regole di riscrittura: ossia ogniqualvolta la formula corrente A contenga una sottoformula del tipo indicato a sinistra, la si rimpiazzi con la corrispondente formula del tipo indicato a destra.

Negli esercizi risulta sempre utile sostituire $\neg(C \to D)$ direttamente con $C \wedge \neg D$ per abbreviare i passaggi

$$\neg(C \to D) \rightsquigarrow \neg(\neg C \vee D) \rightsquigarrow \neg\neg C \wedge \neg D \rightsquigarrow C \wedge \neg D.$$

Esempio 4. Trasformiamo in fnn la formula $\neg(((p \vee q) \to r) \wedge \neg(q \vee r))$:

$$\begin{aligned} & \neg(((p \vee q) \to r) \wedge \neg(q \vee r)) && \rightsquigarrow \\ \rightsquigarrow\ & \neg((p \vee q) \to r) \vee \neg\neg(q \vee r) && \rightsquigarrow \\ \rightsquigarrow\ & ((p \vee q) \wedge \neg r) \vee q \vee r \end{aligned}$$

Esempio 5. Il medesimo risultato dell'esempio 4 si sarebbe potuto ottenere applicando le

regole in un diverso ordine:

$$\begin{aligned}
& \neg(((p \vee q) \rightarrow r) \wedge \neg(q \vee r)) && \rightsquigarrow \\
\rightsquigarrow\ & \neg(((p \vee q) \rightarrow r) \wedge (\neg q \wedge \neg r)) && \rightsquigarrow \\
\rightsquigarrow\ & \neg((p \vee q) \rightarrow r) \vee \neg(\neg q \wedge \neg r)) && \rightsquigarrow \\
\rightsquigarrow\ & ((p \vee q) \wedge \neg r) \vee \neg\neg q \vee \neg\neg r && \rightsquigarrow \\
\rightsquigarrow\ & ((p \vee q) \wedge \neg r) \vee p \vee r
\end{aligned}$$

2.3 Forme Normali Congiuntive

Per trasformare una data A' in fnc esistono due metodi: il *metodo strutturale* ed il *metodo non strutturale*. Il metodo non strutturale produce una A'' logicamente equivalente ad A', ma A'', così ottenuta, risulta essere *esponenzialmente lunga* rispetto ad A'. Il metodo strutturale produce una A'' che è solo equisoddisfacibile con A', ma che è dello *stesso ordine di grandezza* di A'. Il metodo strutturale risulta essere quindi di gran lunga preferibile, almeno su esempi di considerevole grandezza. Cominciamo ad esporre nel seguente paragrafo il metodo non strutturale, che può essere utile negli esercizi (che saranno, per forza di cose, di dimensione ridotta).

Il metodo non strutturale

Il metodo non strutturale è basato sulle regole di riscrittura:

$$\begin{aligned}
C \vee (D_1 \wedge D_2) \quad & \rightsquigarrow \quad (C \vee D_1) \wedge (C \vee D_2) \\
(D_1 \wedge D_2) \vee C \quad & \rightsquigarrow \quad (D_1 \vee C) \wedge (D_2 \vee C).
\end{aligned}$$

Nei casi pratici, è spesso utile svolgere in un sol colpo diverse applicazioni delle regole distributive nel modo seguente. Supponiamo che la formula corrente A' abbia una sottoformula A_0 che sia una disgiunzione del tipo

$$\begin{aligned}
& (C_{11} \wedge \cdots \wedge C_{1k_1}) \vee \\
\vee\ & (C_{21} \wedge \cdots \wedge C_{2k_2}) \vee \\
& \cdots \\
\vee\ & (C_{n1} \wedge \cdots \wedge C_{nk_n}).
\end{aligned}$$

A_0 viene sostituita da una congiunzione i cui congiunti sono le disgiunzioni ottenute prelevando una C_{1i} da $\{C_{11}, \ldots, C_{1k_1}\}$, una C_{2i} da $\{C_{21}, \ldots, C_{2k_2}\}$, eccetera, in tutti i modi possibili.

Esempio 6. La formula $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3)$ diventa

$$(p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee q_3) \wedge (p_1 \vee q_2 \vee p_3) \ \wedge \ (p_1 \vee q_2 \vee q_3) \ \wedge$$
$$\wedge (q_1 \vee p_2 \vee p_3) \ \wedge (q_1 \vee p_2 \vee q_3) \wedge (q_1 \vee q_2 \vee p_3) \ \wedge \ (q_1 \vee q_2 \vee q_3).$$

Questo esempio dovrebbe facilmente convincere dell'andamento esponenziale dell'applicazione delle leggi distributive (in generale, operando su $\bigvee_{i=1}^{n}(p_i \wedge q_i)$ si ottengono 2^n congiunti).

Il metodo strutturale

Vediamo ora il metodo strutturale.

Osservazione. Il metodo descritto in questo paragrafo dà risultati corretti solo se applicato a *formule che siano già in fnn.*

Una formula A' che sia già in fnn ma non in fnc deve per forza contenere una sottoformula del tipo $C \vee (D_1 \wedge D_2)$ o del tipo $(D_1 \wedge D_2) \vee C$. Ci occupiamo del primo caso: il secondo è analogo.

Invece di applicare le leggi distributive a $C \vee (D_1 \wedge D_2)$, consideriamo una **nuova** lettera proposizionale a e aggiorniamo A' con

$$(A'[a/D_1 \wedge D_2]) \wedge (\neg a \vee D_1) \wedge (\neg a \vee D_2)$$

dove $(A'[a/D_1 \wedge D_2]$ è ottenuta da A' sostituendo (una o più) occorrenze di sottoformule del tipo $D_1 \wedge D_2$ con a. Si noti che la lunghezza della nuova A' è stata incrementata al più di una *quantità costante*, per cui il fenomeno di esplosione in spazio non può ripetersi.

Esempio 7. Se applichiamo trasformazioni strutturali alla formula dell'esempio 6, otteniamo come risultato finale

$$(a_1 \vee a_2 \vee a_3) \wedge (\neg a_1 \vee p_1) \wedge (\neg a_1 \vee q_1) \wedge (\neg a_2 \vee p_2) \wedge (\neg a_2 \vee q_2) \wedge (\neg a_3 \vee p_3) \wedge (\neg a_3 \vee q_3)$$

ossia 7 clausole. Nel caso generale di $\bigvee_{i=1}^{n}(p_i \wedge q_i)$ avremo $2n + 1$ clausole.

2.4 L'algoritmo DPLL

La formula A'' in fnc che abbiamo ottenuto dalle trasformazioni delle fasi precedenti sarà del tipo $C_1 \wedge \cdots \wedge C_k$, dove le $C_1, \ldots, C_k$ sono clausole. Identifichiamo A'' con l'insieme di clausole

$$\mathcal{C}_0 = \{C_1, \ldots, C_k\}.$$

La procedura DPLL[1] manipola coppie $(V, \mathcal{C})$ ove $\mathcal{C}$ è un insieme di clausole e V è un *assegnamento parziale* (cioè V assegna un valore di verità solo ad *alcune* delle lettere proposizionali della formula A originaria). Scriviamo la coppia $V, \mathcal{C}$ con

$$V \vdash \mathcal{C}.$$

La procedura si inizializza a $\emptyset \vdash \mathcal{C}_0$ e sviluppa un albero etichettato con coppie $V \vdash \mathcal{C}$; al termine dell'esecuzione della procedura, le foglie dell'albero finale avranno etichetta $V \vdash \mathcal{C}$, *dove* $\mathcal{C}$ *è vuoto oppure contiene la clausola vuota* $\square$.

L'insieme di clausole originario $\mathcal{C}_0$ è soddisfacibile se e solo se l'albero generato da una (qualsiasi) esecuzione della procedura *contiene una foglia etichettata con* $V \vdash \emptyset$ (nel qual caso V - o una sua qualsiasi estensione totale - è un assegnamento che soddisfa $\mathcal{C}_0$).

L'albero generato dalla procedura deve essere esplorato (preferibilmente in profondità) secondo le regole sottoindicate (Sussunzione, Risoluzione Unitaria, Asserzione, Letterale Puro, Spezzamento), *dando precedenza* alle regole che non introducono sdoppiamenti (altrimenti detto, la regola di spezzamento si applica solo se le altre non sono applicabili).

Le regole vanno applicate in modo esaustivo; per applicare una regola, si sceglie una foglia dell'albero corrente la cui etichetta sia del tipo indicato dalla premessa della regola stessa e si generano uno (o due nel caso della regola di spezzamento) nodi figli etichettandoli come previsto dal conseguente della regola.

Osservazione. Si noti che la regola di sussunzione *cancella clausole*, mentre la regola di risoluzione unitaria *cancella letterali*.

[1] Ci ispiriamo liberamente alla presentazione di Cesare Tinelli, *A DPLL-based Calculus for Ground Satisfiability Modulo Theories* (JELIA 2002).

Sussunzione

$$\frac{V \vdash \mathcal{C} \cup \{p \vee C\}}{V \vdash \mathcal{C}} \quad \text{se} \quad V(p) = 1 \tag{2.1}$$

$$\frac{V \vdash \mathcal{C} \cup \{\neg p \vee C\}}{V \vdash \mathcal{C}} \quad \text{se} \quad V(p) = 0 \tag{2.2}$$

Risoluzione unitaria

$$\frac{V \vdash \mathcal{C} \cup \{p \vee C\}}{V \vdash \mathcal{C} \cup \{C\}} \quad \text{se} \quad V(p) = 0 \tag{2.3}$$

$$\frac{V \vdash \mathcal{C} \cup \{\neg p \vee C\}}{V \vdash \mathcal{C} \cup \{C\}} \quad \text{se} \quad V(p) = 1 \tag{2.4}$$

Asserzione

$$\frac{V \vdash \mathcal{C} \cup \{p\}}{V \cup \{V(p) = 1\} \vdash \mathcal{C}} \tag{2.5}$$

$$\frac{V \vdash \mathcal{C} \cup \{\neg p\}}{V \cup \{V(p) = 0\} \vdash \mathcal{C}} \tag{2.6}$$

Letterale puro

$$\frac{V \vdash \mathcal{C}}{V \cup \{V(p) = 1\} \vdash \mathcal{C}} \quad \text{se } p \text{ occorre in } \mathcal{C} \text{ e } \neg p \text{ non occorre in } \mathcal{C} \tag{2.7}$$

$$\frac{V \vdash \mathcal{C}}{V \cup \{V(p) = 0\} \vdash \mathcal{C}} \quad \text{se } \neg p \text{ occorre in } \mathcal{C} \text{ e } p \text{ non occorre in } \mathcal{C} \tag{2.8}$$

Spezzamento

$$\frac{V \vdash \mathcal{C}}{V \cup \{V(p) = 1\} \vdash \mathcal{C} \;\Big\|\; V \cup \{V(p) = 0\} \vdash \mathcal{C}} \tag{2.9}$$

2.5 Esempi

Esempio 8. $\mathcal{C}_0 = \{p_1 \vee p_2, p_1 \vee \neg p_2, \neg p_1 \vee p_2, \neg p_1 \vee \neg p_2\}$

(2.9)		$\emptyset \vdash \mathcal{C}_0$		
(2.1)	$p_1 = 1 \vdash \mathcal{C}_0$		$p_1 = 0 \vdash \mathcal{C}_0$	(2.2)
(2.4)	$p_1 = 1 \vdash \{\neg p_1 \vee p_2, \neg p_1 \vee \neg p_2\}$		$p_1 = 0 \vdash \{p_1 \vee p_2, p_1 \vee \neg p_2\}$	(2.3)
(2.5)	$p_1 = 1 \vdash \{p_2, \neg p_2\}$		$p_1 = 0 \vdash \{p_2, \neg p_2\}$	(2.5)
(2.4)	$p_1 = 1, p_2 = 1 \vdash \{\neg p_2\}$		$p_1 = 0, p_2 = 1 \vdash \{\neg p_2\}$	(2.4)
	$p_1 = 1, p_2 = 1 \vdash \square$		$p_1 = 0, p_2 = 1 \vdash \square$	
	$\times$		$\times$	

$\mathcal{C}_0$ risulta dunque insoddisfacibile.[2]

Esempio 9. $\mathcal{C}_0 = \{\neg p \vee q \vee r, p \vee \neg q \vee \neg r, \neg q \vee r, q \vee \neg r, \neg p \vee \neg q \vee \neg r\}$

$\emptyset \vdash \mathcal{C}_0$			
$p = 1 \vdash \mathcal{C}_0$		$p = 0 \vdash \mathcal{C}_0$	
$p = 1 \vdash \{q \vee r, \neg q \vee r, q \vee \neg r, \neg q \vee \neg r\}$		$p = 0 \vdash \{\neg q \vee \neg r, \neg q \vee r, q \vee \neg r\}$	
$p{=}1, q{=}1 \vdash \ldots$	$p{=}1, q{=}0 \vdash \ldots$	$p{=}0, q{=}1 \vdash \ldots$	$p{=}0, q{=}0 \vdash \ldots$
$p{=}1, q{=}1 \vdash \{r, \neg r\}$	$p{=}1, q{=}0 \vdash \{r, \neg r\}$	$p{=}0, q{=}1 \vdash \{r, \neg r\}$	$p{=}0, q{=}0 \vdash \{\neg r\}$
$p{=}1, q{=}1, r{=}1 \vdash \square$	$p{=}1, q{=}0, r{=}1 \vdash \square$	$p{=}0, q{=}1, r{=}1 \vdash \square$	$\boxed{p{=}0, q{=}0, r{=}0 \vdash \emptyset}$
$\times$	$\times$	$\times$	

$\mathcal{C}_0$ risulta soddisfacibile:[3] l'assegnamento $V(p) = 0, V(q) = 0$ e $V(r) = 0$ ritornato dalla procedura soddisfa $\mathcal{C}_0$.

Esempio 10. Nell'esempio 9, spezzando subito sull'atomo q anziché su p, si sarebbe potuta ottenere una diversa esecuzione:

[2] Si noti che, per facilitare il lettore nella comprensione di questo primo esempio, abbiamo annotato sui nodi interni dell'albero il corrispondente numero della regola applicata ad essi (applicazioni iterate della stessa regola sono state però condensate in un passo solo).

[3] Si noti che, per motivi di sintesi grafica, nel generare l'albero relativo a questo esempio abbiamo applicato in un sol colpo le Regole di Sussunzione e Risoluzione Unitaria. Così, ad esempio, in presenza di un assegnamento parziale contenente $V(p) = 1$, nel passare al nodo successore abbiamo sia cancellato tutte le clausole contenenti p sia rimosso $\neg p$ nelle clausole rimanenti.

$$\emptyset \quad \vdash \quad \mathcal{C}_0$$

$q = 1 \quad \vdash \quad \mathcal{C}_0$	$q = 0 \quad \vdash \quad \mathcal{C}_0$
$q = 1 \quad \vdash \quad \{p \vee \neg r, r, \neg p \vee \neg r\}$	$q = 0 \quad \vdash \quad \{\neg p \vee r, \neg r\}$
$q = 1, r = 1 \quad \vdash \quad \{p, \neg p\}$	$q = 0, r = 0 \quad \vdash \quad \{\neg p\}$
$p = 1, q = 1, r = 1 \quad \vdash \quad \square$	$\boxed{p = 0, q = 0, r = 0 \vdash \emptyset}$
$\times$	

$\mathcal{C}_0$ risulta ovviamente soddisfacibile. Si osservi come la scelta di spezzare sull'atomo q permetta di ottenere il risultato più rapidamente e richieda un numero minore di passi di backtracking rispetto a quanto visto nell'esercizio precedente. Una delle linee di sviluppo dei moderni SAT solver è incentrata proprio nel trovare efficaci euristiche per la scelta del letterale su cui effettuare lo spezzamento, così da ottenere esecuzioni migliori.

Esempio 11.

Si desidera verificare se la formula

$$B \equiv (((r \to r) \to q) \to ((r \to r) \wedge \neg p \wedge q)) \vee (p \wedge q)$$

è una tautologia. B è una tautologia se e solo se $\neg B$ è insoddisfacibile, dunque procederemo ponendo $A := \neg B$ e verificandone la soddisfacibilità. Trasformiamo la formula A in una formula A' in forma normale negativa e logicamente equivalente ad A:

$$\begin{aligned}
A \equiv \quad & \neg((((r \to r) \to q) \to ((r \to r) \wedge \neg p \wedge q)) \vee (p \wedge q)) \rightsquigarrow \\
\rightsquigarrow \quad & \neg(((r \to r) \to q) \to ((r \to r) \wedge \neg p \wedge q)) \wedge \neg(p \wedge q) \rightsquigarrow \\
\rightsquigarrow \quad & ((r \to r) \to q) \wedge \neg((r \to r) \wedge \neg p \wedge q)) \wedge (\neg p \vee \neg q) \rightsquigarrow \\
\rightsquigarrow \quad & (\neg(r \to r) \vee q) \wedge (\neg(r \to r) \vee p \vee \neg q) \wedge (\neg p \vee \neg q) \rightsquigarrow \\
\rightsquigarrow \quad & ((r \wedge \neg r) \vee q) \wedge ((r \wedge \neg r) \vee p \vee \neg q) \wedge (\neg p \vee \neg q) \quad \equiv A'.
\end{aligned}$$

Trasformiamo ora la formula A' in una formula A'' in forma normale congiuntiva. In questo caso utilizziamo il metodo non strutturale che mantiene l'equivalenza logica:

$$\begin{aligned}
A' \equiv \quad & ((r \wedge \neg r) \vee q) \wedge ((r \wedge \neg r) \vee p \vee \neg q) \wedge (\neg p \vee \neg q) \rightsquigarrow \\
\rightsquigarrow \quad & (r \vee q) \wedge (\neg r \vee q) \wedge (r \vee p \vee \neg q) \wedge (\neg r \vee p \vee \neg q) \wedge (\neg p \vee \neg q) \equiv A''.
\end{aligned}$$

Identifichiamo ora la formula A'' con l'insieme di clausole

$$\mathcal{C}_0 = \{p \vee \neg q \vee r, p \vee \neg q \vee \neg r, \neg p \vee \neg q, q \vee r, q \vee \neg r\}$$

ed applichiamo l'algoritmo DPLL per verificare la soddisfacibilità dell'insieme di clausole.

$\emptyset \quad \vdash \quad \mathcal{C}_0$

$p = 1 \vdash \mathcal{C}_0$	$p = 0 \vdash \mathcal{C}_0$	
$p = 1 \vdash \{\neg q, q \vee r, q \vee \neg r\}$	$p = 0 \vdash \{\neg q \vee r, \neg q \vee \neg r, q \vee r, q \vee \neg r\}$	
$p = 1, q = 0 \vdash \{r, \neg r\}$	$p = 0, q = 1 \vdash \ldots$	$p = 0, q = 0 \vdash \ldots$
$p = 1, q = 0, r = 1 \vdash \{\neg r\}$	$p = 0, q = 1 \vdash \{r, \neg r\}$	$p = 0, q = 0 \vdash \{r, \neg r\}$
$p = 1, q = 0, r = 1 \vdash \square$	$p = 0, q = 1, r = 1 \vdash \{\neg r\}$	$p = 0, q = 0, r = 1 \vdash \{\neg r\}$
$\times$	$p = 0, q = 1, r = 1 \vdash \square$	$p = 0, q = 0, r = 1 \vdash \square$
	$\times$	$\times$

A'', logicamente equivalente a A, risulta insoddisfacibile: da ciò possiamo dedurre che $B \equiv \neg A$ è una tautologia.

2.6 zChaff

Uno dei più sofisticati SAT-solver attualmente disponibili è zChaff, sviluppato alla Princeton University:[4] esso rappresenta una efficiente implementazione dell'algoritmo Chaff. I codici sorgenti di zChaff, interamente scritto in linguaggio C++, sono scaricabili all'indirizzo http://www.princeton.edu/~chaff/zchaff.html.

Sotto Linux, i sorgenti possono essere estratti dal'archivio digitando il comando:

```
unzip zchaff*.zip
```

o utilizzando una qualunque utilità atta a scompattare il file. Per compilare il codice, a condizione di aver installato gli strumenti per lo sviluppo quali *gcc* e *make*, è sufficiente digitare il comando

```
make
```

dopo essersi portati nella directory nella quale sono stati scompattati i sorgenti. Per ulteriori informazioni, è possibile seguire le istruzioni contenute nel file `README` incluso nel pacchetto.

Una volta compilato, il programma può essere lanciato con il comando:

[4] Una più che valida alternativa a zChaff è costituita da MINISAT (si veda la pagina web http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/). Si noti che, siccome la sintassi dei SAT-solvers è piuttosto standardizzata, le informazioni di questo paragrafo sono sostanzialmente indipendenti dal sistema su cui si intende lavorare.

```
./zchaff CNF_FILE [LimiteDiTempo]
```

ove `CNF_FILE` è il nome del file contenente una rappresentazione della formula in forma normale congiuntiva di cui si è interessati a verificare la (in)soddifacibilità. `LimiteDiTempo` è invece un parametro opzionale che rappresenta il numero di secondi che devono trascorrere prima che venga terminata l'esecuzione del programma qualora la computazione non fosse ancora conclusa.

I file contenenti la formula devono rispettare una ben precisa sintassi. In primo luogo, ogni linea che inizia con la lettera `c` viene considerata un commento. Inoltre, occorre preporre all'elenco delle clausole un'intestazione rappresentata dalla seguente linea:

```
p cnf numero_di_lettere numero_di_clausole
```

ove `numero_di_lettere` indica il numero di lettere proposizionali che occorrono nella formula e `numero_di_clausole` indica il numero delle clausole presenti nella formula.

Gli atomi sono espressi con numeri da `1` a `numero_di_lettere`; la negazione di un atomo è rappresentata da un numero negativo. Una clausola è invece rappresentata da una riga di letterali separati tra loro da uno spazio e terminante con uno `0`.

L'esempio 8, ($\mathcal{C}_0 = \{p_1 \vee p_2, p_1 \vee \neg p_2, \neg p_1 \vee p_2, \neg p_1 \vee \neg p_2\}$), potrà dunque essere descritto dal seguente file:

```
c *** un semplice esempio insoddisfacibile ***
p cnf 2 4
1 2 0
1 -2 0
-1 2 0
-1 -2 0
```

Esempio 8

ove la cifra `1` corrisponde alla lettera proposizionale p_1 e la cifra `2` corrisponde alla lettera proposizionale p_2.

zChaff indicherà l'insoddisfacibilità della formula testata attraverso il seguente output:

```
Instance unsatisfiable
[...]
RESULT: UNSAT
```

zChaff: formula insoddisfacibile

mentre, se la formula è soddisfacibile, si potrà ottenere un risultato simile al seguente:

```
Instance Satisfiable
-1 -2 -3 Random Seed Used 0
[...]
RESULT: SAT
```

zChaff: formula soddisfacibile

Si noti che la riga `Random Seed Used` fornisce un assegnamento che verifica la formula. L'assegnamento si legge come segue: ogni numero positivo (negativo) equivale ad un assegnamento uguale a 1 (rispettivamente 0) sulla corrispondente lettera proposizionale. Nel caso dell'esempio, la formula è verificata da un assegnamento che associa 0 a ognuna delle tre lettere proposizionali che occorrono nella formula.

L'output di zChaff fornisce molte altre informazioni riguardo l'esecuzione: la maggior parte di esse sono relative alle ottimizzazioni dell'algoritmo DPLL implementate dagli sviluppatori. Per ulteriori informazioni a tal proposito, si consulti la sezione *Approfondimenti*.

2.7 Approfondimenti

Per ulteriori approfondimenti sull'argomento rimandiamo alla consultazione del seguente materiale.

- Procedura DPLL: per una esposizione rigorosa e attuale, incentrata anche sul problema SMT dell'integrazione con teorie specifiche, si veda R. Nieuwenhuis, A. Oliveras, C. Tinelli, *Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)*, Journal of the ACM, in corso di pubblicazione. L'articolo è scaricabile all'indirizzo: ftp://ftp.cs.uiowa.edu/pub/tinelli/papers/NieOT-JACM-06.pdf;

- Il problema SAT - algoritmi e applicazioni: si veda Roberto Sebastiani, *SAT Beyond Propositional Satisfiability*. Le slide sono scaricabili all'indirizzo: http://www.dit.unitn.it/~rseba/DIDATTICA/Tutorials/Slides_tutorial_cade04.pdf;

- Materiale su zChaff. Si consulti la pagina Web del progetto all'indirizzo: http://www.princeton.edu/~chaff;
- Uso di zChaff e panoramica sul codice sorgente: si veda Yinlei Yu, *How to Use/Hack zChaff SAT Solver*. Le slide sono scaricabili in formato PowerPoint all'indirizzo: http://www.princeton.edu/~jdonald/research/zchaff/How to Use Chaff.ppt oppure, in formato pdf, all'indirizzo: http://staff.science.uva.nl/~bcate/AR2005/UsingChaff.pdf;
- Algoritmo DPLL - origini storiche e tecniche avanzate: si veda Sharad Malik, *The Quest for Efficient Boolean Satisfiability Solvers*. Le slide sono scaricabili all'indirizzo: http://www.princeton.edu/~sharad/CMUSATSeminar.pdf;
- Librerie di benchmark/esempi: http://www.satlib.org/. In particolare, per ciò che concerne la verifica formale di microprocessori, le suite di Miroslav N. Velev si trovano all'indirizzo: http://www.satlib.org/I-Velev03/index.htm.