Fondamenti di Informatica 2 - DPLL

Prof. Marco Schaerf

Clausole

♢ I *letterali* sono simboli proposizionali (*atomi*) o simboli proposizionali negati *atomi negati*.

♢ Una *formula* è detta in *forma normale negativa* se il segno di negazione compare solo davanti agli atomi.

♢ Una *clausola* è una disgiunzione di letterali $L_1 \vee L_2 \vee \cdots \vee L_n$.

♢ Una *formula* è detta in *forma normale congiuntiva* (CNF) o in *forma clausale* oppure si dice che essa è un *insieme di clausole* se è in forma normale negativa e inoltre ha la forma $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ (oppure equivalentemente $\{C_1, C_2, \cdots, C_n\}$) dove le C_i sono clausole.

♢ Poiché la disgiunzione è commutativa, una clausola generica può essere scritta come:

$$A_1 \vee A_2 \vee \cdots \vee A_n \vee \neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_m$$

dove gli A_i sono gli n atomi e B_j sono gli m atomi negati.

Clausole

♢ Se $n = m = 0$ si ha la *clausola vuota* e si scrive $\{\}$.

♢ Una clausola verrà nel seguito anche indicata come l'insieme dei suoi letterali e cioè $\{L_1, L_2, \ldots, L_p\}$, omettendo il simbolo di disgiunzione $\vee$.

♢ Se L è un letterale e $C = \{L_1, L_2, \ldots, L_p\}$ una clausola, talvolta scriveremo $L \cup C$ per indicare la clausola $C' = \{L\} \cup C$,
cioè $C' = \{L, L_1, L_2, \ldots, L_p\}$.

Trasformazione in clausole: *Converti*

Si può facilmente mettere una formula qualunque in forma CNF attraverso l'applicazione ripetuta dei seguenti 3 passi:

- Eliminando i connettivi $\rightarrow$ e $\leftrightarrow$:

$$(\alpha \leftrightarrow \beta) \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$
$$(\alpha \rightarrow \beta) \equiv (\neg\alpha \vee \beta)$$

- Mettendo in forma normale per la negazione (NNF):

$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \equiv \alpha$$

- Distribuendo la disgiunzione $\vee$ sulla congiunzione $\wedge$:

$$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

A questo punto dovremo solo raggruppare tutti i letterali in disgiunzione in clausole per ottenere una formula in CNF.

Problemi di $Converti$

Data una formula α, si può facilmente dimostrare che $\alpha \equiv Converti(\alpha)$

Il problema dell'algoritmo $Converti$ è che la dimensione della formula CNF ottenuta potrebbe essere molto maggiore di quella originaria. Esempio:

$((A_1 \wedge A_2) \vee (B_1 \wedge B_2)) \vee (C_1 \wedge C_2))$
$(((A_1 \wedge A_2) \vee B_1) \wedge ((A_1 \wedge A_2) \vee B_2) \vee (C_1 \wedge C_2))$
$((A_1 \vee B_1) \wedge (A_2 \vee B_1) \wedge ((A_1 \vee B_2) \wedge (A_2 \vee B_2) \vee (C_1 \wedge C_2))$
$((A_1 \vee B_1) \wedge (A_2 \vee B_1) \wedge ((A_1 \vee B_2) \wedge (A_2 \vee B_2) \vee C_1) \wedge ((A_1 \vee B_1) \wedge$
$(A_2 \vee B_1) \wedge ((A_1 \vee B_2) \wedge (A_2 \vee B_2) \vee C_2))$
$\ldots$
$\ldots$
$(A_1 \vee B_1 \vee C_1) \wedge (A_2 \vee B_1 \vee C_1) \wedge (A_1 \vee B_2 \vee C_1) \wedge (A_2 \vee B_2 \vee C_1) \wedge$
$(A_1 \vee B_1 \vee C_2) \wedge (A_2 \vee B_1 \vee C_2) \wedge (A_1 \vee B_2 \vee C_2) \wedge (A_2 \vee B_2 \vee C_2)$

Algoritmo DPLL: Idea

Si applica ad un insieme di clausole ed è basato sulla tecnica del backtracking, infatti prova ricorsivamente tutte le assegnazioni possibili per vedere se una soddisfa la formula. Usa 2 controlli per ridurre (significativamente) il numero di assegnazioni da provare:

- **Unit propagation**. Se una clausola è **unitaria** (contiene un solo letterale) essa può essere soddisfatta solo assegnandogli il valore corretto (1 se il letterale è positivo, 0 se è negativo).
- **Pure literal elimination**. Se un letterale appare sempre o positivo o negativo viene chiamato **puro**. I letterali puri possono essere resi tutti veri, senza bisogno di fare scelte.

Algoritmo DPLL: Funzioni di appoggio

- **unit-propagate(letterale l, formula F)** Per ogni clausola $c \in F$, se l compare in c con lo stesso segno allora cancella la clausola c, se l compare in c con il segno opposto allora cancella l da c. Restituisce la formula semplificata.

- **pure-literal-assign(letterale l, formula F)** Cancella tutte le clausole che contengono il letterale l. Restituisce la formula semplificata.

- **choose-literal(formula F)** Seleziona (casualmente) un letterale che compare in F. Restituisce il letterale scelto.

Algoritmo DPLL: Codice

```
function DPLL(F)
   if F is empty
       then return true;
   if F contains an empty clause
       then return false;
   for every unit clause l in F
      F = unit-propagate(l, F);
   for every literal l that occurs pure in F
      F = pure-literal-assign(l, F);
   l := choose-literal(F);
   return DPLL(F Union {l}) OR DPLL(F Union {NOT l});
```

NOTA

Il codice non è ottimizzato; infatti se `F Union l ha successo (ritorna il valore true) non è necessario attivare DPLL(Union NOT l)`

Algoritmo DPLL: esempio

Sia data $F = (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R)$

Alla prima iterazione non possiamo semplificare la formula e quindi effettuiamo operazione di choose-literal

supponi di scegliere $l = \neg P$ ottenendo

$$F = (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R) \wedge (\neg P)$$

applicando **pure-literal-assign** e **unit-propagate** soddisfaciamo la terza e quarta clausola e dobbiamo far sparire P dalla prima e seconda clausola ottenendo

$$F' = (Q) \wedge (\neg Q)$$

ora applicando **pure-literal-assign** su Q e su $\neg Q$ otteniamo in entrambi i casi una clausola vuota

$\Rightarrow$ clausola non soddisfacibile

$\Rightarrow$ dobbiamo fare backtracking e provare assegnazione $l = P$

Algoritmo DPLL: esempio (continua)

Sia data $F = (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R)$

a questo punto si sceglie $l = P$ ottenendo

$$F \wedge (P) = (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R) \wedge (P)$$

applicando **pure-literal-assign** e **unit-propagate** soddisfaciamo la prima, la seconda e la quinta clausola e dobbiamo far sparire $\neg P$ dalla terza e quarta clausola ottenendo

$$F'' = (Q) \wedge (\neg R)$$

ora applicando **pure-literal-assign** su Q otteniamo $F''' = (\neg R)$ e poi su $\neg R$ otteniamo la formula vuota (senza clausole) e quindi verifichiamo che la formula è soddisfacibile.

Algoritmo DPLL: funzione choose-literal

- L'algoritmo DPLL dipende dalla scelta del letterale su cui si esegua l'attivazione ricorsiva.
- Pertanto il codice fornito prima non è esattamente un algoritmo, ma piuttosto una famiglia di algoritmi, uno per ogni possibile criterio di scelta del letterale su cui si esegue la funzione choose-literal.
- L'efficenza del codice dipende molto da questa scelta: esistono istanze per cui il tempo di esecuzione è lineare o esponenziale a seconda della scelta della funzione choose-literal.
- Data l'importanza del problema sia teorica che pratica per le numerose applicazioni ci sono molti studi per la ricerca di metodi efficenti di scelta. Queste funzioni sono dette euristiche.

Algoritmo DPLL: Conclusioni

- Il codice precedente fornisce come risposta solo se la formula è soddisfacibile (vedi prima istruzione if). Il codice può essere facilmente modificato per memorizzare l'assegnazione parziale calcolata e quindi fornire - nel caso la formula sia vera - un'assegnazione di valori che soddisfa la formula.
- DPLL risolve esattamente SAT. Dato che SAT è NP-Completo se assumiamo vera la congettura che $NP \neq P$ allora non può esistere un algoritmo che risolve tutte le istanze del problema in tempo polinomiale. Questo non implica però che possano esistere algoritmi per SAT che hanno tempi di esecuzione accettabili anche per decidere la soddisfacibilitaà di una fomula di significative dimensioni.

Esercizi

1. Verificare che $\neg A \rightarrow (A \rightarrow B)$ è una tautologia.
2. Mostrare che dall'insieme di formule $\Gamma = \{(P \wedge Q) \rightarrow R, P, P \rightarrow Q\}$ si deriva $Q \wedge R$.

Riformulate i problemi come problemi di soddisfacibilità , poi usate l'algoritmo *Converti* per mettere in forma CNF ed infine DPLL per verificare la soddisfacibilità .

Esercizi: Svolgimento 1

$\neg A \to (A \to B)$ è una tautologia se e solo se $\neg(\neg A \to (A \to B))$ non è soddisfacibile. Usando $Converti$ otteniamo:

$\neg(\neg\neg A \vee (\neg A \vee B))$	elimino le implicazioni
$\neg(A \vee (\neg A \vee B))$	elimino doppia negazione
$(\neg A) \wedge \neg(\neg A \vee B))$	spingo dentro la negazione
$\neg A \wedge A \wedge \neg B$	spingo dentro la negazione
$F = \{A, \neg A, B\}$	metto in forma a clausole

Applicando la DPLL ottengo:

$F =$ unit-propagate(A, F)	seleziono la clausola unitaria$A$
$F = \{\}$	applicando unit-propagate la clausola$\neg A$ diventa vuota

Quindi la formula $\neg A \to (A \to B)$ è una tautologia.

Esercizi: Svolgimento 2

Da $\Gamma = \{(P \wedge Q) \rightarrow R, P, P \rightarrow Q\}$ si deriva $Q \wedge R$ se e solo se $\Gamma \cup \{\neg(Q \wedge R)\}$ non è soddisfacibile. Usando $Converti$ otteniamo:

$\neg(P \wedge Q) \vee R, P, \neg P \vee Q, \neg(Q \wedge R)$	elimino le implicazioni
$(\neg P \vee \neg Q) \vee R, P, \neg P \vee Q, \neg Q \vee \neg R$	spingo dentro la negazione
$F = \{\neg P \vee \neg Q \vee R, P, \neg P \vee Q, \neg Q \vee \neg R\}$	metto in forma a clausole

Applicando la DPLL ottengo:

$F =$ unit-propagate(P, F)	seleziono la clausola unitaria P
$F = \{\neg Q \vee R, Q, \neg Q \vee \neg R\}$	applico unit-propagate(P, F)
$F =$ unit-propagate(Q, F)	seleziono la clausola unitaria Q
$F = \{R, \neg R\}$	applico unit-propagate(Q, F)
$F =$ unit-propagate(R, F)	seleziono la clausola unitaria R
$F = \{\}$	applico unit-propagate(R, F)
	ottenendo la clausola vuota

Quindi dalla formula $\Gamma\{(P \wedge Q) \rightarrow R, P, P \rightarrow Q\}$ si deriva $Q \wedge R$

Esercizio 3

Una formula costituita dalla congiunzione di n letterali distinti quanti modelli ha?

$$A_1 \wedge \ldots \wedge A_n$$

Risposta 1

Una formula costituita dalla disgiunzione di n letterali distinti quante assegnazioni di verità la rende vera?

$$A_1 \vee \ldots \vee A_n$$

Risposta: $2^n - 1$

Esercizio 4

Data una formula DNF trovare un'assegnazione di valori di verità che la soddisfa è polinomiale?
Risposta: SI, è sufficente considerare ciacuna clausola e cercare per essa un'assegnazione di valori che la soddisfa.

Esempio: Data $(A_1 \wedge A_2) \vee (\neg \mathcal{A}_2 \wedge A_4) \vee (A_4 \wedge \neg A_4)$ si può vedere che la prima e la seconda clausola sono facilmente sodddisfacibili, mentre la terza no. Dato che abbiamo una clausola che può essere soddisfatta la formula è soddisfacibile

Esercizio 4, cont.

1. Il problema della soddisfacibilità di formule booleane è NP-completo; pertanto congetturiamo che non esistono algoritmi polinomiali nella lungheza dell'input che decidono se la formula è soddisfacibile o meno
2. Abbiamo visto che il problema della soddisfacibilità di formule in formato DNF è polinomiale nella lunghezza dell'input.

Domanda: le due affermazioni precedenti sono in contraddizione fra loro? In altre parole per quale ragione non posso affermare che 2. precedente non implica che anche il problema generale di decidere se una data formula sia soddisfacibile è polinomiale? In particolare posso usare il seguente algoritmo

1. Data una formula F trasformala in una formula F' in forma DNF equivalente
2. Applica una algoritmo polinomiale per decidere se F' è soddisfacibile

Esercizio 4, cont.

Risposta: NO, in particolare l'algoritmo proposto non è polinomiale e può avere tempo di esecuzione esponenziale.

Infatti è vero che, data una formula F, la possiamo trasformare in una formula F' equivalente che è vera se e solo se F è vera.

Ma il problema è che questa trasformazione da F a F' potrebbe non essere realizzabile in tempo polinomiale.
In particolare possiamo avere che la lunghezza della formula F' sia esponenziale nella lughezza di F (ad esempio potrebbe accadere che F abbia lunghezza n e F' ha una numero di clausole esponenziale in n).
In questi casi l'algoritmo che trasforma F in F' ha necessariamente tempo di esecuzione esponenziale nella dimensione di F.