Forme normali congiuntive e disgiuntive

Clausole, Forma normale congiuntiva

◇ *letterali*: simboli proposizionali (*atomi*) o simboli proposizionali negati

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

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

◇ Una *formula* è in *forma normale congiuntiva* (CNF) (o in *forma clausale*) se è in forma normale negativa e 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 è del tipo:

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

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

Clausole

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

$\diamondsuit$ 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 $\vee$.

$\diamondsuit$ 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 $C' = \{L, L_1, L_2, \ldots, L_p\}$.

$\diamondsuit$ Se $n = 1$, cioè se la clausola ha un solo letterale positivo e quindi la forma:

$$A_1 \vee \neg B_1 \vee \ldots, \vee \neg B_m$$

si parla di *clausola definita*.

Trasformazione in clausole: $Convert1$

<u>Teorema</u> Ogni formula è equivalente ad una formula in forma normale disgiuntiva (DNF) ed è equivalente ad una formula in forma normale congiuntiva (CNF).

Si può 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 $Convert1$

Il problema dellalgoritmo $Convert1$ è che la dimensione della formula CNF ottenuta potrebbe essere molto maggiore di quella originaria. Esempio:

$$((A_1 \land A_2) \lor (B_1 \land B_2)) \lor (C_1 \land C_2))$$
$$(((A_1 \land A_2) \lor B_1) \land ((A_1 \land A_2) \lor B_2) \lor (C_1 \land C_2))$$
$$((A_1 \lor B_1) \land (A_2 \lor B_1) \land ((A_1 \lor B_2) \land (A_2 \lor B_2) \lor (C_1 \land C_2))$$
$$((A_1 \lor B_1) \land (A_2 \lor B_1) \land ((A_1 \lor B_2) \land (A_2 \lor B_2) \lor C_1) \land ((A_1 \lor B_1) \land$$
$$(A_2 \lor B_1) \land ((A_1 \lor B_2) \land (A_2 \lor B_2) \lor C_2))$$
$$\ldots$$
$$\ldots$$
$$(A_1 \lor B_1 \lor C_1) \land (A_2 \lor B_1 \lor C_1) \land (A_1 \lor B_2 \lor C_1) \land (A_2 \lor B_2 \lor C_1) \land$$
$$(A_1 \lor B_1 \lor C_2) \land (A_2 \lor B_1 \lor C_2) \land (A_1 \lor B_2 \lor C_2) \land (A_2 \lor B_2 \lor C_2)$$

Trasformazione in clausole: $Convert2$

Per ovviare a questi problemi possiamo usare una diversa trasformazione, dove i passi 1 e 2 sono uguali al precedente producendo una formula F, mentre il passo 3 produce una formula CNF F_1 e consiste in:

- Se A è una lettera proposizionale, $Convert2(A) = A$ e $Convert2(\neg A) = \neg A$.
- Per ogni sottoformula $\gamma = \alpha \wedge \beta$ di F, crea tre nuove lettere $A_\gamma, A_\alpha, A_\beta$, aggiungi ad F_1 le clausole $\neg A_\gamma \vee A_\alpha$ e $\neg A_\gamma \vee A_\beta$ e continua con la conversione di α e di β.
- Per ogni sottoformula $\gamma = \alpha \vee \beta$ di F, crea tre nuove lettere $A_\gamma, A_\alpha, A_\beta$, aggiungi ad F_1 la clausola $\neg A_\gamma \vee A_\alpha \vee A_\beta$ e continua con la conversione di α e di β.
- Aggiungi la clausola unitaria A_F ad F_1.

Esempio di $Convert2$

Usiamo la formula della slide precedente $((A_1 \wedge A_2) \vee (B_1 \wedge B_2)) \vee (C_1 \wedge C_2))$.
Diamo un nome a tutte le sottoformule $D_1 = A_1 \wedge A_2$, $D_2 = B_1 \wedge B_2$ e $D_3 = C_1 \wedge C_2$, $D_4 = D_1 \vee D_2$ e $F = D_4 \vee D_3$ ed inizializzamo $F_1 = \{\}$

$$
\begin{array}{ll}
((A_1 \wedge A_2) \vee (B_1 \wedge B_2)) \vee (C_1 \wedge C_2)) & F_1 = \{\} \\
((D_1 \vee (B_1 \wedge B_2)) \vee (C_1 \wedge C_2)) & F_1 = F_1 \cup \{(\neg D_1 \vee A_1), (\neg D_1 \vee A_2)\} \\
((D_1 \vee D_2) \vee (C_1 \wedge C_2)) & F_1 = F_1 \cup \{(\neg D_2 \vee B_1), (\neg D_2 \vee B_2)\} \\
((D_1 \vee D_2) \vee D_3) & F_1 = F_1 \cup \{(\neg D_3 \vee C_1), (\neg D_3 \vee C_2)\} \\
(D_4 \vee D_3) & F_1 = F_1 \cup \{(\neg D_4 \vee D_1 \vee D_2)\} \\
F & F_1 = F_1 \cup \{(\neg A_F \vee D_4 \vee D_3)\} \\
 & F_1 = F_1 \cup \{A_F\}
\end{array}
$$

Abbiamo quindi alla fine
$F_1 = \{(\neg D_1 \vee A_1), (\neg D_1 \vee A_2), (\neg D_2 \vee B_1), (\neg D_2 \vee B_2), (\neg D_3 \vee C_1), (\neg D_3 \vee C_2), (\neg D_4 \vee D_1), (\neg D_4 \vee D_2), (\neg D_1 \vee D_3 \vee D_4), (\neg A_F \vee D_4 \vee D_3), A_F\}$

Proprietà

- Data una formula α, si può facilmente dimostrare che $\alpha \equiv Convert1(\alpha)$

- Data una formula α, osserviamo che $\alpha \not\equiv Convert2(\alpha)$. Infatti le due formule non hanno nemmeno lo stesso alfabeto.
  Si può però dimostrare che α è soddisfacibile se e solo se $Convert2(\alpha)$ è soddisfacibile. Si dice che la trasformazione $Convert2$ preserva la soddisfacibilità

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);
   x := DPLL(F Union {l});
   y := DPLL(F Union {NOT l});
   return  x OR y;
```

◇ NOTA: L'algoritmo precedente non è ottimizzato. Infatti se è vero
`x := DPLL(F Union l)` non è necessario eseguire
`y := DPLL(F Union NOT l)` (la funzione ritorna comunque il valore vero)