ESERCIZIO

8.1 Il problema 3-SODDISFACIBILITÀ e la codifica χ_1

Il problema SODDISFACIBILITÀ (in breve, SAT) chiede se, data una funzione booleana f in forma congiuntiva normale, esiste una assegnazione di verità alle variabili sulle quali f è definita che soddisfa f, ossia, che rende vera f. Formalmente, quindi, SAT è definito dalla tripla seguente:

$I_{SAT} = \{\langle f, X\rangle$ tale che f è in forma congiuntiva normale $\}$;

$S_{SAT}(f,X) = \{a : X \rightarrow \{\mathtt{vero}, \mathtt{falso}\}\}$;

$\pi_{SAT}(f, S_{SAT}(f,X)) = \exists a \in S_{SAT}(f,X) : f(a(x_1), \ldots, a(x_n)) = \mathtt{vero}$, ossia, sostituendo in f ogni occorrenza della variabile x_i con il valore $a(x_i)$ (ed ogni occorrenza di $\neg x_i$ con $\neg a(x_i)$), per ogni $i = 1, \ldots, n$, la funzione f assume il valore `vero`.

Nel resto di questo esercizio ci concentreremo sul problema 3SAT le cui istanze sono funzioni booleane in forma 3-congiuntiva normale - ossia, tutte le clausole in una istanza di 3SAT sono costituite da esattamente 3 di variabili (semplici o negate).

Ricordiamo, ora, la codifica χ_1 per l'insieme I_{3SAT}. Sia $\Sigma = \{0,1\}$. Codifichiamo ciascuna delle variabili in X con n caratteri di Σ: in particolare, la variabile x_i è codificata dalla parola di n caratteri il cui unico '1' è quello in posizione i (e, di conseguenza, tutti gli altri caratteri sono '0'). Codifichiamo, poi, ciascuna clausola c_j con la lista delle codifiche delle variabili che essa contiene ciascuna preceduta da uno '0' o da un '1': se la clausola c_j contiene x_i allora, nella codifica di c_j la codifica di x_i sarà preceduta da '0', se, invece, la clausola c_j contiene $\neg x_i$ allora, nella codifica di c_j la codifica di x_i sarà preceduta da '1'. Le clausole saranno codificate una di seguito all'altra, senza interruzione. Infine, per riuscire a capire quando termina una variabile e inizia la successiva (e, quindi, anche quando termina una clausola e inizia la successiva), abbiamo bisogno di conoscere il valore n: a questo scopo, la codifica di f inizia con una sequenza di n '1' seguita da uno '0'. Ad esempio, se $\hat{f} = f(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, la codifica di $\langle \hat{f}, \{x_1,x_2,x_3\}\rangle$ sarà:

$$\chi_1(\hat{f}, \{x_1,x_2,x_3\}) = 11100100101010011100101010 01.$$

L'algoritmo `A:SAT`(χ_1) descritto in Tabella 8.1, codificato in linguaggio **PascalMinimo**, prende in input la codifica $\chi_1(f)$ di f memorizzata nell'array binario F ed il numero m di clausole contenute in f:

L'algoritmo è una successione di (al più 2^n) tentativi di assegnazioni di valori di verità alle variabili che compaiono in f.
La linea 2 conta il numero n di variabili in f.

Input: array binario $F[\,]$, costruito in accordo alle regole di χ_1, e $m \in \mathbb{N}$.
Output: accetta o rigetta.

```
n ← 0;
while (F[n+1] = 1) do n ← n+1;
for (i = 0; i ≤ n; i ← i+1) do a[i] ← 0;
soddisfatta ← falso;
finito ← falso;
while (¬ soddisfatta ∧ ¬ finito) do begin
    i ← n+2;
    for (j ← 1; j ≤ m; j ← j+1) do begin
        clausolavera ← falso;
        neg ← F[i];
        i ← i+1;
        while (i ≤ n+1+3(j−1)(n+1)+n+1 ∧ F[i] = 0) do i ← i+1;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+1)] = 1 ∧ neg= 0) then clausolavera ← vero;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+1)] = 0 ∧ neg= 1) then clausolavera ← vero;
        neg ← F[i];
        i ← i+1;
        while (i ≤ n+1+3(j−1)(n+1)+2(n+1) ∧ F[i] = 0) do i ← i+1;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+n+2)] = 1 ∧ neg= 0) then clausolavera ← vero;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+n+2)] = 0 ∧ neg= 1) then clausolavera ← vero;
        neg ← F[i];
        i ← i+1;
        while (i ≤ n+1+3(j−1)(n+1)+3(n+1) ∧ F[i] = 0) do i ← i+1;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+2n+3)] = 1 ∧ neg= 0) then clausolavera ← vero;
        if (F[i] = 1 ∧ a[i−(n+1+(j−1)(n+1)+2n+3)] = 0 ∧ neg= 1) then clausolavera ← vero;
        if (clausolavera= falso) then j ← m+1;
    end;
    if (clausolavera) then soddisfatta← vero;
    else begin
        r ← 1;
        for (h ← 1; h ≤ n; h ← h+1) do begin
            if (r = 1 ∧ a[h] = 0) then begin
                a[h] ← 1;
                r ← 0;
            end;
            else if (r = 1 ∧ a[h] = 1) then a[h] ← 0;
        end:
        if (r = 1) then finito ← vero;
    end;
end;
if (trovata) then Output: accetta;
else Output: rigetta.
```

Tabella 8.1: Algoritmo A:SAT(χ_1).

La linea 3 inizializza l'array binario a, utilizzato per memorizzare le assegnazioni di verità alle n variabili in f: l'assegnazione iniziale, costituita di soli 0, corrisponde ad assegnare il valore `falso` a tutte le variabili.
La variabile soddisfatta, inizializzata a `falso` alla linea 4, indica se è stata trovata una assegnazione di verità che soddisfa f.
La variabile finito, inizializzata a `falso` alla linea 5, indica se sono state esaminate tutte le assegnazioni di verità per le variabili in f.

Il ciclo **while** alle linee 6-39 verifica se una data assegnazione di verità soddisfa f (linee 8-26) e, se così non è, genera la assegnazione di verità successiva (linee 28-38). Più in dettaglio:

- la prima clausola è codificata negli elementi $F[n+2], F[n+3], \ldots, F[n+1+3(n+1)]$ dell'array F: il primo letterale negli elementi $F[n+1+1], \ldots, F[n+1+n+1]$ (e l'elemento $F[n+1+1]$ indica se il primo letterale è una variabile semplice o negata), il secondo letterale negli elementi $F[n+1+n+1+1], \ldots, F[n+1+2(n+1)]$ (e l'elemento $F[n+1+n+1+1]$ indica se il secondo letterale è una variabile semplice o negata), il terzo letterale negli elementi $F[n+1+2(n+1)+1], \ldots, F[n+1+3(n+1)]$ (e l'elemento $F[n+1+2(n+1)+1]$ indica se il terzo letterale è una variabile semplice o negata);

- la clausola j è codificata negli elementi $F[n+1+3(j-1)(n+1)], \ldots, F[n+1+3j(n+1)]$ dell'array F: il primo letterale negli elementi $F[n+1+3(j-1)(n+1)+1], \ldots, F[n+1+3(j-1)(n+1)+n+1]$ (e l'elemento $F[n+1+3(j-1)(n+1)+1]$ indica se il primo letterale è una variabile semplice o negata), il secondo letterale negli elementi $F[n+1+3(j-1)(n+1)+n+1+1], \ldots, F[n+1+3(j-1)(n+1)+2(n+1)]$ (e l'elemento $F[n+1+3(j-1)(n+1)+n+1+1]$ indica se il secondo letterale è una variabile semplice o negata), il terzo letterale negli elementi $F[n+1+3(j-1)(n+1)+2(n+1)+1], \ldots, F[n+1+3(j-1)(n+1)+3(n+1)]$ (e l'elemento $F[n+1+2(n+1)+1]$ indica se il terzo letterale è una variabile semplice o negata)

Ciò premesso, le linee 11-14, all'iterazione j del ciclo **for**, verificano se l'assegnazione di verità memorizzata nell'array a rende vero il primo letterale della clausola j. Innanzi tutto, l'istruzione alla linea 11 dell'iterazione j del ciclo **for** viene eseguita quando $i = n+1+(j-1)(n+1)+1$, ossia, quando l'elemento i dell'array F è quello che indica se il primo letterale della clausola j è una variabile semplice, se $F[i] = 0$, oppure negata, se $F[i] = 1$, e il valore di $F[i]$ viene memorizzato nella variabile neg. Poi, la clausola j contiene come primo letterale la variabile x_ℓ se $F[n+1+(j-1)(n+1)+\ell] = 1$; quindi, una volta individuato l'unico i compreso fra $n+1+(j-1)(n+1)+2$ e $n+1+(j-1)(n+1)+n+1$ tale che $F[i] = 1$ (linea 12), si verifica se la clausola j è soddisfatta dal suo primo letterale: questo accade solo se, detto $\ell = i-[n+1+(j-1)(n+1)+1$, accade che il primo letterale della clausola j è x_ℓ e ad x_ℓ è stato assegnato il valore `vero` ($F[i] = 1 \wedge neg = 0 \wedge a[i-(n+1+(j-1)(n+1)+1)] = 1$) oppure che il primo letterale della clausola j è $\neg x_\ell$ e ad x_ℓ è stato assegnato il valore `falso` ($F[i] = 1 \wedge neg = 1 \wedge a[i-(n+1+(j-1)(n+1)+1)] = 0$).
Analogamente, all'iterazione j del ciclo **for**, le linee 15-19 verificano se l'assegnazione di verità memorizzata nell'array a rende vero il secondo letterale della clausola j e le linee 20-24 verificano se l'assegnazione di verità memorizzata nell'array a rende vero il terzo letterale della clausola j.
Se l'assegnazione di verità memorizzata nell'array a permette di soddisfare la clausola j, alla variabile clausolavera (inizializzata a `falso` alla linea 9) viene assegnato il valore `vero`; se questo non accade significa che l'assegnazione di verità non soddisfa la clausola j e, dunque non può soddisfare f: in questo caso, è inutile verificare se l'assegnazione di verità memorizzata in a riesce a soddisfare le altre clausole e, dunque, il ciclo **for** viene interrotto assegnando alla variabile j il valore $m+1$ (linea 25).

Dunque, è possibile uscire dal ciclo **for** (linee 8-26) con la variabile clausolavera settata a `vero` oppure a `falso`.
Se si verifica la prima eventualità significa che in nessuna delle iterazioni del ciclo è stata eseguita l'istruzione alla linea 25: dunque, l'assegnazione di verità soddisfa tutte le clausole in f - ossia soddisfa f. In questo caso, la linea 27 assegna il valore `vero` alla variabile soddisfatta, salta l'esecuzione dell'**else** (linee 28-38) e induce la terminazione del ciclo **while** (linee 6-39).
Se, invece, si esce dal ciclo **for** con la variabile clausolavera settata a `falso`, allora l'assegnazione di verità memorizzata in a non soddisfa f. In questo caso, viene eseguito l'**else** alle linee 28-38 che genera una nuova assegnazione di verità. Una assegnazione di verità, ricordiamo, è una sequenza di n elementi 0 e 1 - l'assegnazione iniziale utilizzata dall'algoritmo è una sequenza di n elementi 0, ossia, se $n = 4$;

$$0000.$$

Una sequenza che, altro non è che il numero 0 rappresentato da 4 bit. Se sommiamo 1 a questa rappresentazione, otteniamo

$$0001,$$

ossia, il numero 1 rappresentato con 4 bit - ma anche una nuova assegnazione di verità per un insieme di 4 variabili booleane. E, se sommiamo ancora 1, otteniamo

$$0010,$$

la codifica binaria di 2, rappresentata in 4 bit - ma anche una nuova assegnazione di verità.
Questo significa che, data una assegnazione di verità per un insieme di n variabili, sommando ad essa 1 otteniamo l'assegnazione di verità successiva. E questo è quanto realizzato dalle linee 29-36: la variabile r è inizializzata ad 1 (il valore da sommare al numero rappresentato dall'array a), e viene sommata in binario all'elemento $a[1]$ - generando eventualmente un riporto memorizzato nella variabile r stessa. Poi si somma il nuovo valore di r ad $a[1]$, e così via fino ad $a[n]$.
Come ci si accorge di aver generato tutte le assegnazioni di verità possibili? Ebbene: l'ultima assegnazione di verità che viene generata in questo modo è quella costituita da soli elementi 1 - nel caso $n = 4$ essa è

$$1111.$$

Se sommiamo 1 a questa sequenza quello che otteniamo è la sequenza 0000 *con riporto di* 1: è facile convincersi che il ciclo **for** alle linee 30-34 termina con il valore 1 memorizzato nella variabile r se e soltanto se si è tentato di sommare 1 ad una sequenza di n elementi 1. Quando, dunque, viene eseguita l'istruzione **if** alla linea 37 significa che sono state considerate tutte le assegnazioni di verità alle n variabili e nessuna di esse ha soddisfatto f. Il ciclo **while** alla linea 6 viene terminato dalla variabile finito cui è stato assegnato il valore `vero` alla linea 37, e nella variabile soddisfatta rimane memorizzato il valore `falso`.