Specifiche Modulo1

N. Galesi—Dipartimento di Informatica

1. Preliminari

Come visto nell'introduzione il nostro progetto consiste nello scrivere una programma per decidere se una formula è soddisfacibile e in tal caso restituisce un assegnamento che soddisfa la formula. Si ricordi che la formula è una CNF e viene data in un formato conosciuto come formato DIMACS.

2. Obiettivo

L'obiettivo del primo modulo è quello di salvare in una opportuna struttura di dati una formula in formato CNF che verrà letta da un file di testo scritto in formato DIMACS.

3. Descrizione del formato DIMACS

Un file in formato DIMACS è un file di testo che codifica una formula in formato CNF. Il file è formato da linee di caratteri secondo il seguente schema

- Un certo numero (anche 0) di linee iniziali che iniziano con il carattere 'c' e che sono linee di commento.
- Una linea che inizia con il carattere 'p' dove sono inserite anche altre informazioni come il formato della formula (che nel nostro caso è solo cnf) , il numero di variabili totali che compongono la formula, e il numero di clausole totali che compongono la formula. Ad esempio una linea del tipo

  p……cnf ……485……10896

  ci sta informando che il file codifica una formula con 485 variabili e 10896 clausole
- Tante linee quanto il numero totale di clausole. Ogni linea codifica una clausola, salvando l'indice i (rispettivamente –i) di una variabile se quella variabile appare con polarità positiva (rispettivamente negativa) nella clausola. La linea termina con il carattere '0' (zero)

  Ad esempio la clausola ($x_{12} \vee \neg x_{203} \vee x_2 \vee \neg x_5$) viene codificata dalla linea

  12 -203 2 -5 0

4. Specifiche del programma

Il primo modulo consisterà di differenti "classi" che implementano degli oggetti che useremo nello sviluppo del solver. Le classi sono per il momento: `modulo1`, `solver`, `vettori`, `clausole` e `letterali`. Ognuna di queste classi verrà implementata da un file .c che contiene la definizione i metodi della classe e da un file .h, che contiene l'interfaccia della classe, vale a dire, dichiarazione di dati e prototipi di funzioni che devono essere visibili ad altre classi. Per il momento `letterali` contiene solo dei dati. I file forniti per il primo modulo sono:

```
Modulo1.c
Modulo1.h
Solver.c
Solver.h
Vettori.c
Vettori.h
```

```
Clausole.c
Clausole.h
Letterali.h
```

Il materiale include anche un `main` fornito dal docente in `main.c` e un `Makefile` che gli studenti dovranno usare per testare il proprio modulo1 e che sarà usato dal docente per valutarlo.
Alcuni di questi file dovranno essere completati secondo quanto descritto nelle specifiche qui riportate. La parti da completare a cura dello studente che costituiscono **il lavoro da svolgere nel modulo1**, sono contrassegnate dal simbolo ► **.**

5. Vettori generici come ADT: `vettori.h e vettori.c`

Nel corso dello sviluppo del solver, avremo bisogno di lavorare con vettori i cui elementi possono essere di differenti tipi, in particolare interi o indirizzi di oggetti di differente tipi. Il modulo `vettori` (`vettori.c` e `vettori.h`) contiene dichiarazione e funzioni che consentono di lavorare su vettori generici disinteressandosi della loro implementazione. Descriveremo qui il tipo `veci` che consente di lavorare su vettori di interi. Il tipo `vec_p` va sviluppato come parte del lavoro nel primo modulo.

```
struct veci {
    int    el;
    int index;
    int*  ptr;
};

typedef struct veci vec_i;
```

I metodi che consento di lavorare su `vec_i` sono i tipici metodi di un ADT. Un costruttore:

```
void veci_new (vec_i *v, int num_el) {
    v->el = num_el;
    v->index=0;
    v->ptr  = (int *) malloc(sizeof(int) * num_el);
}
```

Un distruttore:

```
void   veci_delete (vec_i* v){
    v->index=0;
    v->el=0;
    free(v->ptr);
}
```

Metodi informativi:

```
int *  veci_vett  (vec_i* v)
```

che restituisce il vettore.

```
int    veci_numel  (vec_i* v)
```

che restituisce il numero di elementi del vettore.

```
void   veci_ins(vec_i* v, int e)
```

Che inserisce un nuovo elemento nel vettore.

Si presti attenzione al fatto che il metodo costruttore `veci_new`, necessita di sapere a priori l'informazione sulla dimensione del vettore da salvare. È possibile implementare questi metodi anche senza questa informazione. Limitatamente al primo modulo (e probabilmente anche per gli altri) ciò non è necessario. Eventuali modifiche posteriori saranno descritte e fornite agli studenti. Chi volesse potrebbe direttamente implementare tali moduli senza la conoscenza delle dimensione (in questo caso si veda l'uso della funzione `realloc`). Le definizioni e i prototipi delle funzioni che devono essere visibili esternamente e usate da altri devono sono salvate in `vettori.h`.

► Definire i prototipi dei metodi per il tipo `vec_p`

***6. Classi preliminari*: `letterali.h`, `clausole.c` e `clausole.h`**

La classe `letterali.h` contiene le necessarie dichiarazioni di variabili, letterale e del tipo bool. Nel progetto un variabile viene codificata salvando come un intero il suo indice. Ad esempio x_{50} viene codificata dall'intero 50. Se la variabile appare negata la sua codifica è l'intero -50. il tipo bool codifica i valori di verità vero, 1 e falso 0. Useremo anche un tipo lbool, booleano liftato, a cui aggiungeremo un terzo valore di verità: -1, che codifica uno stato indeterminato.

La classe `clausole`, conterrà tutte le dichiarazione di tipi e funzioni che si occuperanno di lavorare su clausole. Limitatamente al modulo 1 (ma verrà estesa in seguito) `clausole` contiene la dichiarazione del tipo clausola:

```
struct clause{
    bool learnt;
    int size;
    float activity;
    lit *lits;
};

typedef struct clause clause ;
```

Il campo `size` contiene il numero di letterali della clausola, il campo `*lits` può pensarsi come un vettore che contiene la codifica dei letterali che formano la clausola. `learnt` e `activity` sono due parametri che verrano usati in seguito e che adesso devono semplicemente essere settati rispettivamente a falso e 0.0.

Per il momento la classe `clausole` contiene due metodi. Il primo:

```
extern bool new_clause(bool appr, int size, float act,
                       vec_i *v, clause *c);
```

consente di creare una nuova clausola. Come precondizione questa funzione si aspetta che `v` contenga un vettore interi che codificano i letterali della clausola, in un ordine che per ora non è ancora specificato ed irrilevante. `Size` contiene la dimensione della clausola (ci si attende quindi un valore >0), `float e appr` contengono rispettivamente un valore che codificherà l'attività di tale clausola (si veda il prossimo modulo per una spiegazione di tali parametri) e un valore booleano `appr` che indica se si tratta di una clausola della formula iniziale (falso) o una clausola appresa durante il processo (vero)- vedremo più avanti un spiegazione più dettagliata su questo parametro. Il parametro di output è c, in cui salveremo i dati relativi ad una nuova clausola. La funzione rilascia un booleano che indica se la creazione è andata a buon fine.

- **Definire il metodo `new_clause`.**

La classe clausole contiene anche un'altra funzione.

```
extern void print_clauses (vec_p *pre);
```

Lo scopo di tale funzione è quella di stampare in output le clausole contenute in `pre`, un vettore di `*void`, che devono essere interpretati come `*clause`. Questo metodo è funzionale alla valutazione del primo modulo e non rientra nell'economia del solver. Sarà quindi eliminato a partire dal modulo2 se non necessario. La funzione è fornita dal docente.

7. La classe solver.

La classe `solver`, costituirà la parte centrale de nostro progetto. Conterrà tutte le dichiarazione necessarie alla costruzione del solver vero e proprio e i metodi principali. Nel modulo 1 l'informazione in `solver` è limitata. Per prima cosa `solver` contiene la definizione del tipo di dato `solver`, centrale nel programma e che conterrà tutte le informazioni necessarie: Per il momento (ma in seguito sarà completato con molti altri campi) la sua definizione è la seguente:

```
struct solver {
    int nof_var;
    int nof_clauses;
    vec_p clauses;

    /* Da completare nei prossimi moduli */

};
```

`nof_var`, deve contenere il numero di variabili della formula CNF in input, `nof_clauses` il numero di clausole, mentre clauses contiene un oggetto di tipo `vec_p` (inteso come contenitore di tipi `*clause`). In questo campo dovremo salvare la nostra formula in input.

Tra i metodi necessari per questo modulo1 abbiamo:

```
bool solver_aggiungiclausola(solver *s, clause *c)
```

che aggiunge la clausola `* c` nel solver `* s` nel campo corretto (nel modulo1 una clausole può essere aggiunta solo al campo `clauses`, cioè il database delle clausole iniziali). Restituisce un bool che segnala se l'operazione è riuscita. L'implementazione è fornita dal docente è verrà completata nei moduli successivi.

Vi sono poi due moduli di inzializzazione: il primo che alloca gli spazi necessari pe la struttra solver:

```
solver * solver_new()
```

e il secondo che alloca gli spazi ed inizializza i differenti campi del solver:

```
bool solver_init (solver *s)
```

Al momento attuale si occupano solo dei 3 campi che definiscono solver. In seguito potrebbero essere fusi in un unico metodo. Sono forniti dal docente e sono usate solo nel main che a sua volta è fornito dal docente.

► Salvare in `solver` il numero di variabili, il numero di clausole e le clausole in input .

8. Descrizione di `modulo1.c`

La classe `modulo1` è quella che richiede per il primo modulo la gran parte del lavoro da svolgere. Contiene una funzione principale che deve essere completata:

```
bool main_modulo1(FILE *in, solver *s)
```

Questa funzione deve leggere il file in input `*s` e salvare le opportune informazioni (numero variabili, numero clausole e clausole) nel solver secondo quanto specificato in solver.

Il file dovrebbe inizialmente essere letto in una stringa, in modo da accedere più velocemente. Di questo si deve occupare la funzione

```
static char* readFile(FILE *  in)
```

che dovrebbe restituire la stringa contenente il file.

La funzione `main_modulo1`, deve poi usare la funzione

```
static void crea_clausola(char **in, vec_i *v) {
```

che legge una linea dalla stringa `*in` che contiene il file in input e salva in un vettore `v` tipo `vec_i` le informazioni sui letterali.

Tale vettore deve essere poi trasformato in una clausola vera e propria con il metodo `new_clause` della classe `clausole` e, infine, il suo l'indirizzo salvato nel campo `clauses` del `solver`, usando il metodo `vecp_ins` della classe `vettori`. Il metodo `solver_init` della classe `solver` dovrà essere usato nel `modulo1` non appena conosciuta l'informazione relativa al numero di variabili e numero di clausole.

`Main_modulo1` rilascia un booleano che, per il momento, indica che tutto è andato a buon fine.

► Completare modulo1.c

9. Main

Il main contiene un'unica funzione che dopo aver inizializzato il solver, chiama main_modulo1 e ne stampa il database delle clausole in input, con `print_clauses`. È fornito dal docente e chiama alcune funzioni che per il momento non eseguono istruzioni. Sarà usato per valutare il modulo1.