Problema B

Algoritmo DPLL em Lógica Proposicional

Pretende-se implementar uma versão simplificada do algoritmo de Davis-Putnam–Logemann–Loveland (DPLL) que faz parte da do corpo de conhecimento da disciplina de Lógica Computacional. Embora este algoritmo tenha sido formulado em 1962, o seu impacto e relevância ainda se fazem sentir nos dias de hoje. Incorpora, a título de exemplo, o núcleo de demonstradores de teoremas modernos utilizados industrialmente na verificação de hardware e de software (*SAT-solvers* e *Satisfiability Modulo Theories solvers*).

Problema

Dada uma fórmula em lógica proposicional o objetivo passa por transformá-la em Forma Normal Conjuntiva (FNC) e aplicar o algoritmo DPLL que verifica se esta é satisfeita. Para uma dada formula se a sua negação for contraditória então dizemos que a formula é válida.

Descrição do algoritmo DPLL

Relembramos que uma dada formula em FNC tem a seguinte forma

$$(l_{11} \vee l_{12} \vee \cdots \vee l_{1k_1}) \wedge \ldots \wedge (l_{n1} \vee l_{n2} \vee \cdots \vee l_{nk_n}),$$

onde cada l_i é um literal (i.e. uma variável proposicional ou a sua negação), cada componente $(l_1 \vee l_2 \vee \cdots \vee l_k)$ forma um conjunto de disjunções de literais designado por cláusula e existem n cláusulas. Recordemos igualmente que ambos $\top$ e $\bot$ são literais (negativo e positivo) onde $\top$ é o mesmo que dizer $\neg\bot$.

Considerando que a conjunção e a disjunção são idempotentes, associativas e comutativas temos que uma formula em FNC é um conjunto de conjuntos de literais

$$S \triangleq \{\{l_{11}, l_{12}, \cdots l_{1k_1}\}, \cdots, \{l_{n1}, l_{n2}, \cdots, l_{nk_n}\}\}.$$

Usando esta representação, se $\{\} \in S$ então S é equivalente a $\bot$. Se $S = \{\}$ então S é equivalente a $\top$.

Assuma a existência de um literal l e $-l$ o literal oposto, isto é, $-l = P$ se $l = \neg P$ e $-l = \neg P$ se $l = P$.

Considere a noção de simplificação de S na assunção de que l é válida

$$S|_l \triangleq \{c \setminus \{-l\} \mid c \in S \text{ e } l \notin c\}.$$

Uma cláusula constituída por um literal único é designada de *cláusula unitária*. Para uma formula em FNC ser satisfeita é necessário que, em particular, as suas cláusulas unitárias sejam satisfeitas. Neste caso, não há duvidas sobre o valor de verdade das variáveis proposicionais nelas contidas. Esta constatação permite simplificar uma formula em FNC.

A operação de simplificação `unit_propagate` realiza a simplificação de S e define-se por

`unit_propagate(`S`)` $\triangleq$
  enquanto $\{\} \notin S$ e S tiver uma cláusula unitária l fazer
    $S \leftarrow S|_l$.

Por exemplo se simplificássemos o conjunto $S \triangleq \{\{-a, b, c\}, \{a\}\}$ teríamos que *unit_propagate*(S) seria igual a $\{\{b, c\}\}$.

O algoritmo DPLL sobre uma entrada S (a formula por analisar em forma de conjunto de conjuntos de literais) define-se da seguinte forma

$\texttt{DPLL}(S) \triangleq$
  $\texttt{unit\_propagate}(S)$
  se $S = \{\}$ então devolver SAT
  senão se $\{\} \in S$ então devolver UNSAT
  senão
    $l \leftarrow$ um literal qualquer retirado de S
    se $\texttt{DPLL}(S|_l) = SAT$ então devolver SAT
    senão devolver $\texttt{DPLL}(S|_{-l})$

Para saber se uma formula, em FNC, S é satisfeita ou não, basta que `DPLL(S)` seja igual a SAT. Caso seja UNSAT significa que a formula é contraditória.

Como nota final à definição do algoritmo por implementar, acrescenta-se que esta versão é bastante simplista. As implementações modernas utilizados industrialmente na verificação de hardware e de software (*SAT-solvers* e *SMT solvers*) conseguem melhorar em grande medida o desempenho da procura.

Formato e estrutura para a fórmula

Considere para esse efeito os tipos de dados OCaml seguintes para representar fórmulas proposicionais:

```
type variable = string
type formula =                       (*    Sintaxe no INPUT     *)
    Implies of formula * formula     (* (Formula  → Formula)    *)
  | Equiv of formula * formula       (* (Formula ↔ Formula)     *)
  | Or of formula * formula          (* (Formula | Formula)     *)
  | And of formula * formula         (* (Formula & Formula)     *)
  | Not of formula                   (* !(Formula)              *)
  | Var of variable                  (* X1, X2 ...              *)
  | True                             (* TRUE  (Top)             *)
  | False                            (* FALSE (Bottom)          *)
```

A sintaxe da fórmula segue a gramática seguinte:

```
Formula ::=  Var | TRUE | FALSE | (Formula → Formula) | (Formula ↔ Formula)
          | (Formula | Formula) | (Formula & Formula) | !(Formula)
```

onde Var representa o conjunto das variáveis. Assume-se que seguem a forma X_n onde n é o índice inteiro maior compreendido entre 0 e 200, inclusive. Admitimos aqui que se uma fórmula contém uma variável X_n então esta contém igualmente X_1, $X_2 \cdots X_{n-1}$.

Para efetuar o *parsing* da sequência de caracteres deverá utilizar-se a função

```
val parse: string → formula list option
```

que lê uma string e devolve uma lista de fórmula (do tipo formula) correspondente. A função parse transforma assim a string "`((X1 | !(X2)) <-> X3)`" no elemento (que estará na cabeça da lista) do tipo formula, como por exemplo, Equiv (Or (Var "X1", Not (Var "X2")), Var "X3").

Compiliação e Utilização

Para poder utilizar a função parse deverá abrir a biblioteca utilizando **open** F_parser e chamar a função parse com a sequencia de caracteres de entrada proveniente do "STDIN" como habitual, por exemplo similar a parse "stdin".

Para compilar a solução devem colocar a pasta `f_parser` na mesma diretório do ficheiro fonte `dpll.ml`, onde vão trabalhar, e em seguida executar o comandos de compilação:

- `ocamlc -I f_parser/ f_parser.cma dpll.ml -o pbB`

A biblioteca disponibilizada suporta apenas a versão `4.14.1`.

Input

Assim sendo, o input é composto por uma string contendo uma fórmula qualquer f que pode não estar em FNC no formato anteriormente apresentado.

Output

Uma linha com a palavra `UNSAT` se a formula é uma contradição, `SAT` no caso contrário.

Constraints

O índice das variáveis lógicas é um inteiro natural compreendido entre 0 e 200.

Sample Input1

```
((X1 | !(X1)) & (X1 & !(X1)))
```

Sample Output1

```
UNSAT
```

Sample Input2

```
(!X1 & (!X2 & !X3)) & ((!X2) <-> (X3 & (X2 & X1)))
```

Sample Output2

```
UNSAT
```