Problema A - Lógica Computacional 2022/2023

Minimal effort for Maximal Profitability

Problema

A competição na industria do hardware é rude e todo cêntimo ganho na feição dos sistemas de circuitos lógicos põe qualquer empresa a frente da concorrência.

Entretanto, em Taiwan, fizeram uma descoberta maravilhosa que permite agora construir portas *NOR* (que representam a conectiva booleana $\oplus$) extremamente eficientes e de forma extremamente barata.

Aqui na UBI, soubemos de antemão destas notícias. Fortes e seguros da nossa cultura de engenheiros informáticos e de empreendedores, adivinhamos um negócio multimilionário. Eis o nosso plano. Sabemos que é possível representar circuitos lógicos por fórmulas da lógica proposicional. Assim $A \wedge (B \vee C)$ representa um circuito.

Sabemos que empresas como AMD ou INTEL estão interessados em comprar os serviços a quem conseguir optimizar os circuitos escritos neste formato.

Mais, da lógica proposicional sabemos que é possível transformar qualquer fórmula proposicional em fórmulas equivalentes que só contemplam portas/conectivas *nor* ($\oplus$).

O principio da transformação é simples, basta saber que:

$$\begin{array}{lcl} \neg A & \equiv & A \oplus A \\ A \wedge B & \equiv & (A \oplus A) \oplus (B \oplus B) \\ A \vee B & \equiv & (A \oplus B) \oplus (A \oplus B) \end{array}$$

Assim, temos os ingredientes principais para desenhar e implementar um construtor de circuitos lógicos a partir dos dados fornecidos e vencer a concorrência. É esta a vossa tarefa.

Para tal vamos considerar que a sintaxe das fórmulas fornecida segue a gramática seguinte:

```
Formula ::=  Var | TRUE | FALSE | (Formula -> Formula) | (Formula <-> Formula)
          | (Formula | Formula) | (Formula & Formula) | !(Formula)
```

em que Var representa o conjunto das variáveis proposicionais (são simplesmente identificadores, como "A"ou "P").

Por exemplo, a fórmula $(A \Rightarrow B)$ escreve-se, neste formato

```
(A -> B)
```

A sua transformação em fórmula da lógica minimal é

```
(((A % A) % B) % ((A % A) % B))
```

Onde % é a sintaxe ASCII para a conectiva $\oplus$.

Iremos assumir que nenhuma fórmula considerada neste problema se refere na prática à variável Z, pelo que poderemos considerar sem prejuízo que a transformação de $\bot$ ou de $\top$ resulta numa fórmula que envolve a variável Z sempre que não exista nenhuma outra variável presente na fórmula (e.g. `B`). Caso exista então alguma outra variável na fórmula, deverão utilizar a variável que for lexicograficamente mais pequena para efetuar a transformação.

Para realizar estas transformações em OCaml, contemplamos então os seguintes tipos de dados que codificam as formulas lógicas proposicionais:

```
type formula =
  | Var of variable
  | Not of formula
  | And of formula * formula
  | Or of formula * formula
  | Implies of formula * formula
```

```
  | Equiv of formula * formula
  | True
  | False
```

e a seguinte função (fornecida na página da disciplina)

```
val parse: string -> formula list option
```

que lê uma string e devolve uma lista de fórmula (de tipo `formula`) correspondente. A função `parse` transforma assim a string "`((A | !(B)) <-> C)`" no elemento (que estará na cabeça da lista) do tipo `formula`:

```
Equiv (Or (Var "A", Not (Var "B")), Var "C")
```

Compiliação e Utilização

Para poder utilizar a função `parse` terá que satisfazer as seguintes condições:

- fazer `open F_parser`
- invocar a função da seguinte maneira: `parse "stdin"`

Por forma a conseguir compilar a vossa proposta de solução deverão ter a pasta `f_parser` na mesma directoria que o vosso ficheiro fonte, e em seguida executar **apenas um** dos comandos de compilação seguintes:

- `ocamlopt -I f_parser/ f_parser.cmxa YOUR_SOURCE_FILE.ml -o NAME_OF_EXECUTABLE`
- `ocamlc -I f_parser/ f_parser.cma YOUR_SOURCE_FILE.ml -o NAME_OF_EXECUTABLE`

Input

Uma linha com a string representando a fórmula lógica proposicional f no formato apresentado anteriormente.

Output

Uma linha com a string representando a fórmula lógica proposicional minimal f' no formato apresentado anteriormente onde f' é equivalente a f e é resultante da transformação anteriormente sugerida.

Sample Input 1

```
(A -> B)
```

Sample Output 1

```
(((A % A) % B) % ((A % A) % B))
```

Sample Input 2

```
(A & B)
```

Sample Output 2

```
((A % A) % (B % B))
```