Problem D

Derivadas parciais de expressões regulares para o calculo de equivalência

O objectivo deste problema é a implementação de um algoritmo, baseado sobre as derivadas parciais de expressões regulares, que permite determinar se duas expressões regulares são equivalentes (ou não).

Problem

A sua tarefa é, dadas duas expressões regulares, aplicar o algoritmo de verificação de equivalência apresentado no documento seguinte:

Nelma Moreira, David Pereira, Simão Melo de Sousa. Deciding Regular Expressions (In-)Equivalence in Coq. In Relational and Algebraic Methods in Computer Science - RAMICS 2012: 98-113

Este artigo encontra-se disponível nos links seguintes: http://www.liacc.up.pt/ kat/pdcoq/ e http://www.liacc.up.pt/ kat/pdcoq/decide.pdf

O algoritmo por implementar, um processo de decisão para o cálculo directo de equivalência de expressões regulares, encontra-se na página 5 do artigo previamente citado.

Considere para esse efeito os tipos de dados OCaml seguintes para representar as expressões regulares:

```
type expreg =                (*      Sintaxe no INPUT       *)
     V                       (*         o símbolo $         *)
   | E                       (*         o símbolo %         *)
   | C of char               (*    uma letra entre a-z A-Z  *)
   | U of expreg * expreg    (*           (r + s)           *)
   | P of expreg * expreg    (*            r.s              *)
   | S of expreg             (*              r*             *)
```

Input

A sintaxe das duas expressões regulares segue a gramática seguinte:

```
RegExp ::= $ | % | Char | (RegExp + RegExp) | (RegExp . RegExp) | (Formula)*
```

em que *Char* representa o alfabeto (aqui, um subconjunto do tipo `char`, o conjunto das letras possíveis).

O processo de leitura de tais expressões dado no ficheiro leitura_regexp.ml que usa os módulos `Str`, `Stream` e `Genlex` do OCaml para realizar a analise léxica e sintáctica da entrada.

Assim sendo, o input é organizado da seguinte forma:

Primeira linha: uma string contendo a expressão regular r no formato anteriormente apresentado;

Segunda linha: uma string contendo a expressão regular s no formato anteriormente apresentado;

Output

Uma linha com a string "YES" se r é equivalente a s, "NO" senão.

Constraints

Não há restrições assinaláveis.

Sample Input

```
(a.((a + b)*.(b.a)))
((a.b)* + ((b.a)* + ((a)* + (b)*)))
```

Sample Output

```
NO
```