On manipule ici des mots, qui sont des suites finies de lettres d'un alphabet. Par exemple, *car* est un mot sur l'alphabet latin. On appelle facteur d'un mot tout mot qui apparaît tel quel dans celui-ci. Par exemple *car* est un facteur de *decarcasser*. Il existe un unique mot ne contenant aucune lettre, appelé le mot vide et noté ε.

1) Mots et facteurs

En Caml, on représente les mots par le type `string` des chaînes de caractères. `a^ b` représente la concaténation de deux chaînes `a` et `b`, et `String.sub a i l` représente le facteur $a_i \dots a_{i+l-1}$ du mot `a` $= a_0 \dots a_{n-1}$ (de longueur $n =$ `String.length a`).

Dans cet exercice, on va avoir besoin de travailler de manière importante sur les facteurs d'un mot. Pour gagner en performance, on va représenter les facteurs d'un mot non pas avec le type `string`, mais avec le type `facteur` ci-dessous : le facteur est déterminé par le mot entier, l'indice de la lettre de début, et l'indice de la lettre de fin (exclu).

```
type facteur = {
  lettres : string ;
  debut : int ;
  fin : int
};;
```

a) Écrire la fonction `mot` qui représente un mot complet à l'aide de ce type (en observant que tout mot est facteur de lui-même). Elle est de type `string -> facteur`. L'intérêt de cette représentation est que couper un facteur en deux est très rapide (elle dispense de recopier les lettres du facteur).

b) Écrire la fonction `couper` de type `facteur -> int -> facteur * facteur` telle que `couper f i` coupe le facteur `f` au niveau de la lettre `i` du mot complet. Ainsi, si l'on coupe $f_a \dots f_{b-1}$, on obtiendra les facteurs $f_a \dots f_{i-1}$ et $f_i \dots f_{b-1}$.

2) Reconnaissance

Un langage est un sous-ensemble de $\mathscr{A}^*$. La représentation informatique usuelle d'un langage L se fait à travers une fonction de reconnaissance $\varphi : \mathscr{A}^* \to \{\texttt{true}, \texttt{false}\}$ telle que $L = \varphi^{-1}(\{\texttt{true}\})$ (autrement dit, la fonction renvoie `true` si le mot fourni est dans le langage, et `false` sinon). On va travailler ici avec des fonctions de reconnaissance, dont le type sera donc `facteur -> bool`.

Si L_1 et L_2 sont des langages reconnus par les fonctions φ_1 et φ_2, alors on note $\varphi_1|\varphi_2$ la fonction reconnaissant le langage $L_1 \cup L_2$.

Si L_1 et L_2 sont des langages, on définit le langage concaténé :

$$L_1 L_2 = \{ab, a \in L_1, b \in L_2\}$$

On note par ailleurs $\varphi_1\varphi_2$ la fonction de reconnaissance de ce langage.

Si L est un langage, on note $L^0 = \{\varepsilon\}, L^1 = L, L^2 = LL$ et plus généralement L^n la concaténation de n exemplaires de L.

On définit alors l'étoile :

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

a) Écrire la fonction `singleton` de type `string -> facteur ->bool` telle que `singleton s` soit la fonction de reconnaissance du langage ne contenant que le mot `s`.
On pourra utiliser = pour tester l'égalité entre deux chaines de caractères.

On pose `let epsilon = singleton ""` (fonction de reconnaissance du langage $\{\varepsilon\}$) et `let vide = function (_ : facteur) -> false` (fonction de reconnaissance du langage vide $\emptyset$).

b) Écrire la fonction `ou` de type `(facteur -> bool) ->(facteur -> bool)-> facteur -> bool` telle que `ou phi1 phi2` renvoie la fonction $\varphi_1|\varphi_2$, où φ_1 et φ_2 sont des fonctions de reconnaissance.
À l'aide des fonctions `singleton`, `vide` *et* `ou`, *on peut construire une fonction pour reconnaître n'importe quel langage fini.*

c) Écrire la fonction `puis` qui à deux fonctions de reconnaissance φ_1 et φ_2 associe la fonction de reconnaissance $\varphi_1\varphi_2$.

d) Écrire la fonction `etoile` qui a une fonction de reconnaissance φ d'un langage L associe la fonction de reconnaissance φ^* du langage L^*. On procédera récursivement.
L'ensemble des langages que l'on peut reconnaître à l'aide des seules fonctions définies dans cet énoncé est appelé ensemble des langages rationnels.

e) Reprendre la question précédente en utilisant la programmation dynamique : on construit une matrice $(m_{i,j})$ où $m_{i,j}$ vaut `true` si et seulement si le facteur de début i et de fin j est reconnu.
Pour ce faire, la matrice représentera initialement les facteurs reconnus par le langage $\{\varepsilon\} \cup L$, puis (à l'aide d'une boucle) par $L^0 \cup L^1 \cup L^2$, par $L^0 \cup L^1 \cup L^2 \cup L^3 \cup L^4$, par $L^0 \cup \ldots \cup L^8$, etc... On s'arrêtera lorsqu'on ne parvient plus à trouver de nouveaux facteurs reconnus.
L'intérêt de cette méthode est que les appels à la fonction de reconnaissance φ ne sont effectués que dans la phase d'initialisation.

3) On utilise maintenant des expressions rationnelles de type `exp_rat` donné par

```
type expr_rat =
   | Vide
   | Epsilon
   | Mot of string
   | Produit of expr_rat list
   | Somme of expr_rat list
   | Etoile of expr_rat ;;
```

Écrire une fonction `expToPhi : exp_rat -> (facteur -> bool)` qui associe à une expression rationnelle sa fonction de reconnaissance.