Algorithme de décomposition d'un texte en lexèmes.

Inspiration : Partie 2 de Compilation : analyse lexicale et syntaxique

Idée : Reconnaître une suite de lexèmes caractérisés par des expressions rationnelles dans un texte donné.

Exemple : $x + 13 \times y \longrightarrow$ id | plus | nombre | mult | id

Soient Σ un alphabet de caractères, et T l'ensemble des types des lexèmes.

Definition 1 : Une règle d'analyse lexicale est un couple (e, t) où e est une expression rationnelle sur Σ telle que $\varepsilon \notin L(e)$, et $t \in T$. On notera $e \longrightarrow t$.

Problème 1 : On cherche à résoudre le problème suivant :
Entrées : $(e_i \longrightarrow t_i)_{i \in [\![1;n]\!]}$ une liste ordonnée de règles.
texte, une chaîne de caractères
Sortie : Liste de lexème décomposant le texte.

Remarque 3 : D'après le théorème de Kleene, pour chaque e_i il existe (et on peut construire) un automate fini reconnaissant e_i. De plus, comme les règles sont fixées et que cet algorithme est usuellement utilisé plus d'une fois, on peut supposer posséder des automates A_i reconnaissant e_i obtenus par pré-calcul.

```
Fonction analyse_lexicale(texte) :
    Si texte == "" alors
        Renvoyer []
    Sinon
        (lexeme, nouveau_texte) := premier_lexeme(texte)
        Renvoyer lexeme :: (analyse_lexicale(nouveau_texte))
    Fin si
Fin fonction

Fonction premier_lexeme(texte) :
    pos_fin_meilleur := -∞
    meilleur_type := ⊥
    Initialiser l'automate Ai pour i de 1 a' n
    Pour i de 1 a' longueur(texte) faire
        Pour j allant de 1 a' n faire
            Faire lire a' Aj le caractère texte[i]
            Si Aj est dans un état final faire
                pos_fin_meilleur := i
                meilleur_type := type de Aj
            Fin si
        Fin pour
    Fin pour
```

```
Si indice_meilleur == -1 faire
    | Erreur: le texte ne contient aucun préfixe qui
    |         soit un lexème
Fin si
prefixe := texte[1 : pos_fin_meilleur]
lexeme := (prefixe, type de A_indice_meilleur)
Retourner (lexeme, texte[pos_fin_meilleur + 1 :])
Fin fonction
```

Théorème 4: Dans le pire cas, l'algorithme est quadratique en la taille du texte.

Preuve: Pour un texte de longueur n, analyse_lexicale sera appelé au plus n fois, faisant appel à premier_lexeme qui parcourt dans le pire cas tout le texte restant. La complexité est donc de $O(n^2)$.

Exemple 5: $a \longrightarrow$ lexeme_1

$a^* b \longrightarrow$ lexeme_2

sur le texte a^n.