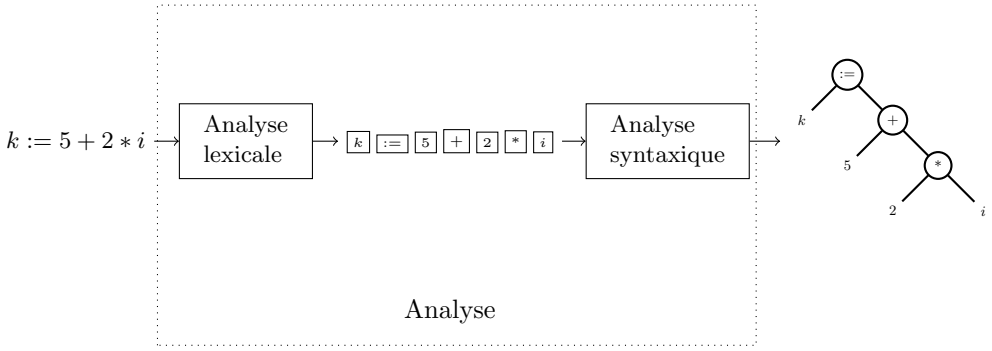
3.2 Analyse lexicale

En compilation, on analyse une chaîne de caractères pour obtenir sa structure appelée *arbre syntaxique*. Voici l'analyse de $k := 5 + 2 * i$.



La première phase s'appelle l'*analyse lexicale* et transforme la chaîne de caractères en *lexèmes*.

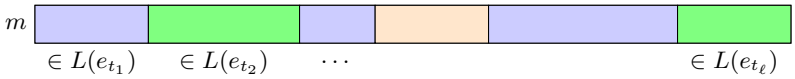
La phase d'analyse lexicale est utile car :

- elle simplifie la tâche à l'analyse syntaxique ;
- elle permet de faire de la coloration syntaxique dans un éditeur de texte.

3.2.1 Spécification

Soit $(e_1, \ldots, e_k)$ une suite finie d'expressions rationnelles.

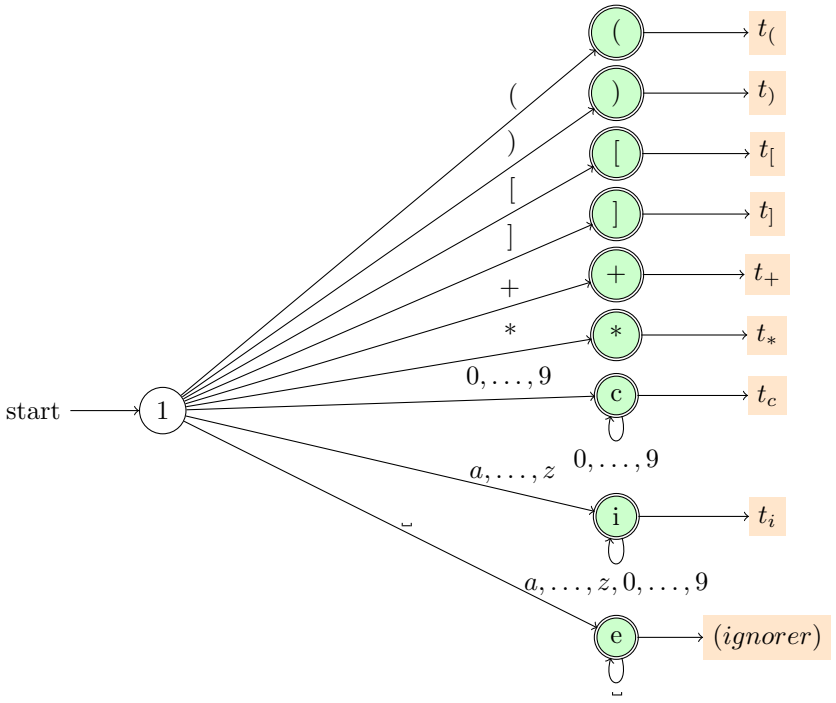
- Entrée : un mot fini m ;
- Sortie : un découpage $i_1 = 1 < i_2 < \cdots < i_\ell < i_{\ell+1} = |m| + 1$, et $(t_1, \ldots, t_\ell) \in \{1, \ldots, k\}$ tels que pour tout $j \in \{1, \ldots, \ell\}$:
  1. $m[i_j..i_{j+1} - 1] \in L(e_{t_j})$;
  2. $m[i_j..i_{j+1} - 1] \notin L(e_t)$ pour $t < t_j$;
  3. Si $j < \ell$ alors $m[i_j..i] \notin L(e_1 + \ldots e_k)$ pour tout $i \geq i_{j+1}$.



Le point 1 signifie que la portion du mot numéro j est reconnu par l'expression régulière e_{t_j}. Le point 2 signifie que cette portion n'est pas reconnue par une expression régulière de plus forte priorité. Le point 3 signifie que si on continue à lire la portion numéro j alors on ne reconnaîtra jamais de lexèmes. En d'autres termes, on reconnaît à chaque fois le plus long préfixe.

3.2.2 Automate

On construit un automate déterministe pour $L(e_1 \cup \dots e_k)$. Soit $\mathcal{A} = (Q, \delta, q_0, F)$ un automate déterministe complet pour le langage $L(e_1 + \dots + e_k)$. A chaque état final, on dit si quelle expression régulière est concernée. C'est modélisé par une fonction $\tau : F \to \{1, \dots, k\}$, qui à tout état final donne le numéro de l'expression rationnelle concerné. Ci-dessous un exemple de tel automate (l'état puits n'est pas représenté).



3.2.3 Algorithme naïf

L'algorithme utilise l'automate $\mathcal{A}$ décrit dans la sous-section précédente.

```
function analyseLex(m)
    if m = ε then
        return []
    else
        t, m' := reconnaitreLexeme(m)
        return t :: analyseLex(m')
    endIf
endFunction
```

```
function reconnaitreLexeme(m)
    q := q_0
    q_fin := ⊗
    i_fin := ⊗
    i := 1
    while i ≤ |m| et δ(q, m[i]) ≠ ∅ (différent de l'état puits) do
        if q ∈ F then
            q_fin := q
            i_fin := i
        endIf
        q := δ(q, m[i])
        i := i + 1
    endWhile
    if q ∈ F then
        q_fin := q
        i_fin := i
    endIf
    if q_fin = ⊗ then
        ERROR
    endIf
    return τ(q_fin), m[i_fin..]
endFunction
```

Complexité de analyseLex : $O(|m|^2)$

3.2.4 Programmation dynamique

🎯 avoir un algorithme en temps linéaire en la longueur du mot

💡 Ne pas refaire toujours les même calculs. Stocker les résultats.
→ Programmation dynamique

Ce qui nous intéresse vraiment : pour tout $i \in \{1, \ldots, n+1\}$, le préfixe maximal reconnu de $m[i..]$ à partir de l'état initial q_0 et son type.

On découpe en sous-problèmes. Voici un sous-problème : pour tout $i \in \{1, \ldots, n+1\}$, pour tout état q de l'automate $\mathcal{A}$, calculer, s'il existe, le préfixe maximal reconnu de $m[i..]$ à partir d'un état q et son type.

Formellement, on définit

$$T : \{1, \ldots, n+1\} \times Q \to (\{1, \ldots, k\} \times \{1, \ldots, n+1\}) \cup \{\otimes\}$$

de la façon suivante :

- $T(i, q) := (t, i_{fin})$ où le préfixe maximal de $m[i..]$ reconnu par $\mathcal{A}$ à partir de l'état q est $m[i..i_{fin} - 1]$ et t est son type, s'il existe ;

$\in L(\mathcal{A}, q)$

(et en lisant plus, on arrive dans l'état puits)

- $T(i,q) :=$ ⊗ s'il n'existe pas de préfixe reconnu par $\mathcal{A}$ à partir de l'état q de $m[i..]$.

Ce qui nous intéresse vraiment : pour tout $i \in \{1, \ldots, n+1\}$, la valeur $T(i, q_0)$.

Cherchons une équation récursive pour calculer $T(i,q)$.

- Cas de base :
  - $T(n+1,q) = \begin{cases} (\tau(q), n+1) & \text{si } q \in F \\ \otimes & \text{si } q \notin F \end{cases}$
  - Si $\emptyset$ est l'état puits,
    $T(i, \emptyset) =$ ⊗
- Cas récursif :
  - Si q n'est pas l'état puits,
    $T(i,q) = \begin{cases} (\tau(q), i) & \text{si } T(i+1, \delta(q, m[i])) = \otimes \text{ et } q \in F \\ T(i+1, \delta(q, m[i])) & \text{sinon} \end{cases}$

On calcule le tableau T en $O(|Q| \times |m|)$. Puis on utilise ce tableau pour faire l'analyse lexicale comme suit.

```
function analyseLex(m)
    calculer le tableau T avec les équations données plus haut
    i := 1
    L := []
    while i ≤ |m|
        if T[i, q0] = ⊗ then
            ERROR
        else //T[i, q0] s'écrit (t, i_fin)
            i := i_fin
            L = L :: t
        endIf
    return L
endFunction
```

Complexité de l'analyse lexicale : $O(|Q| \times |m|)$