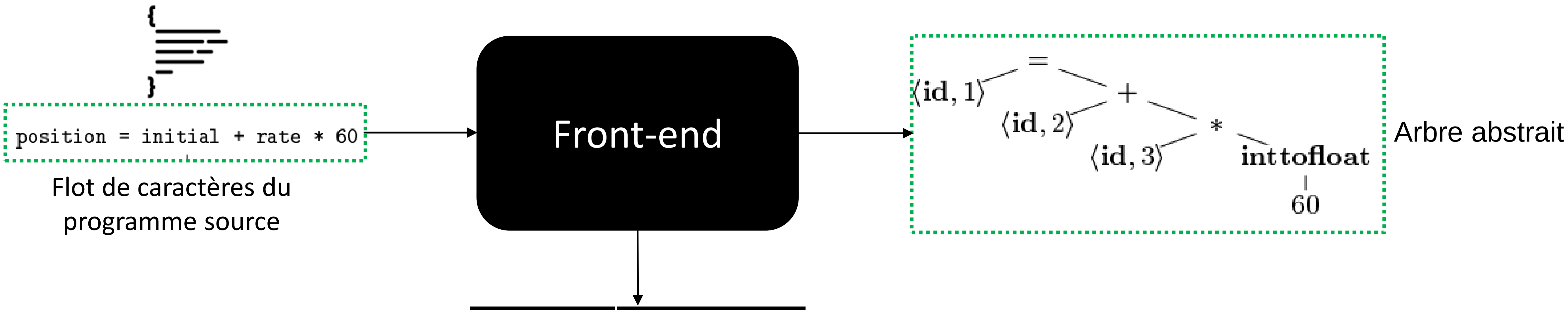
Compilation

Analyse syntaxique

Structure d’un compilateur (Front-end)

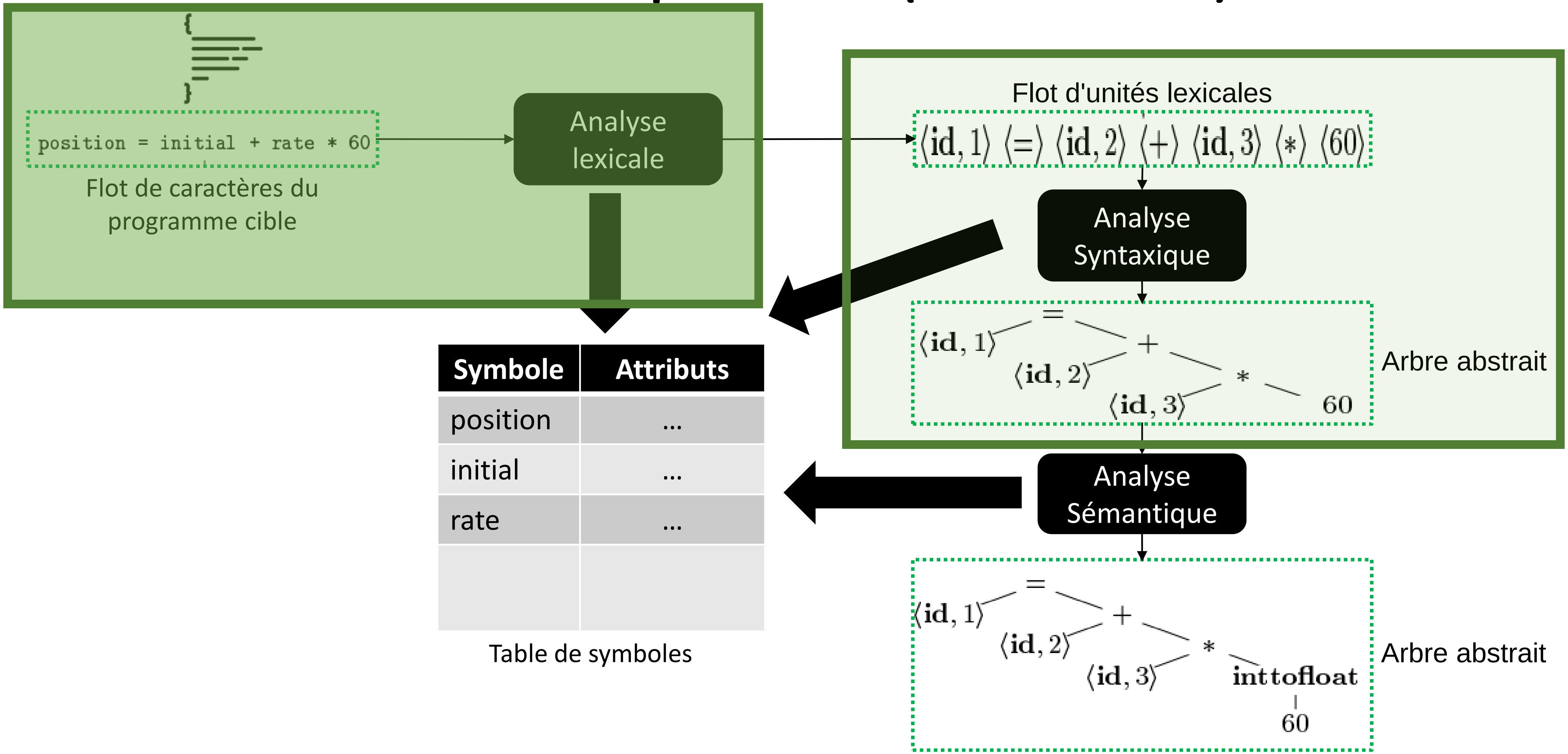


Symbole	Attributs
position	…
initial	…
rate	…

Table de symboles

Le front-end essaye de comprendre le programme source

Structure d'un compilateur (Front-end)



Symbole	Attributs
position	…
initial	…
rate	…

Table de symboles

C'est quoi l'analyse lexicale ? (Exemple)

- Chaine de caractères:

```
position = initial + rate * 60
```

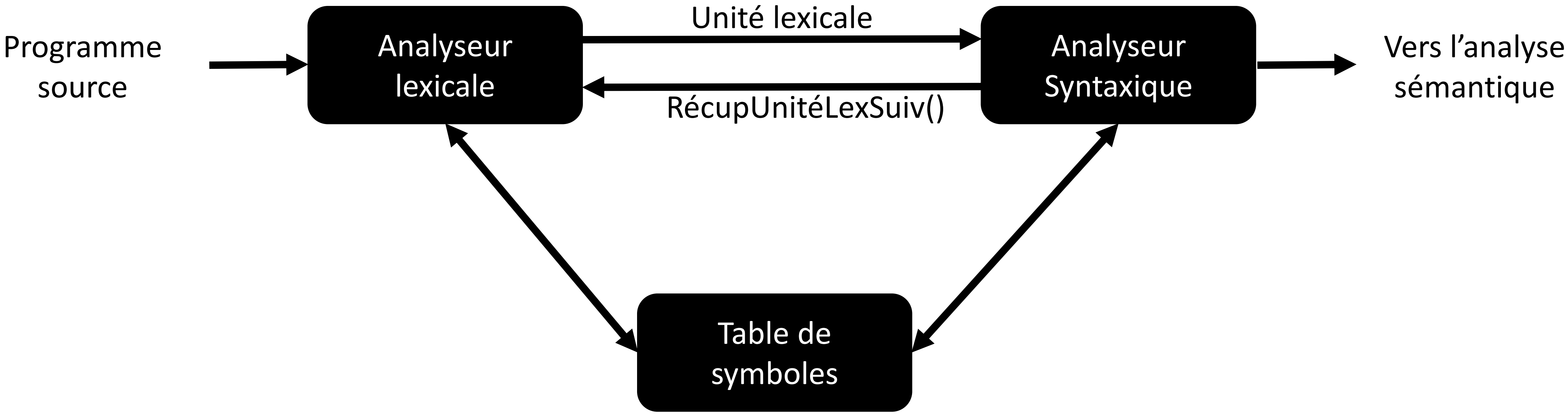
- Chaine des unités lexicales

< id, 1 >, **< = >**, **< id, 2 >**, **< + >**, **< id, 3 >**, **< * >**, **< nbr, 60 >**

- Table de symbols

Symbole	Attributs
`position`	…
`initial`	…
`rate`	…

C'est quoi l’analyse lexicale ?



C'est quoi l’analyse syntaxique ?

- Reçoit une chaine d’**unité lexicales** fournie par l’analyseur lexicale

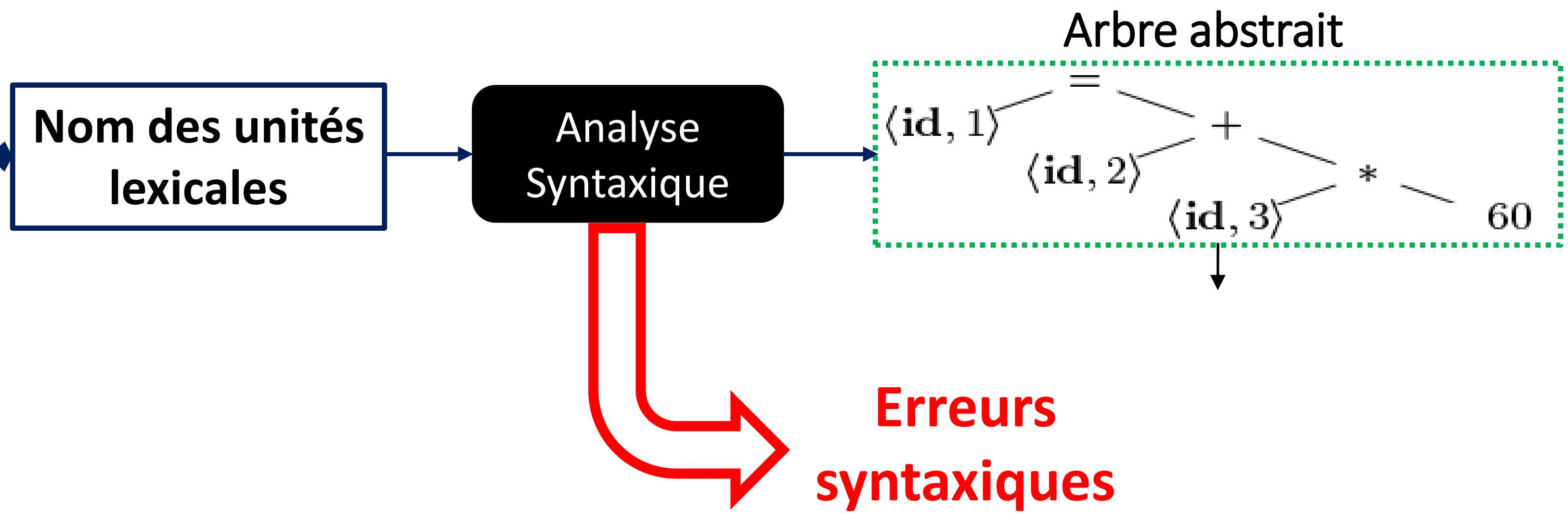
- Vérifier si cette chaine d’**unité lexicales** respecte la **grammaire** du langage source

- Signaler les **erreurs syntaxiques**

- Construire **un arbre d’analyse** pour les phases suivantes de compilation

C'est quoi l'analyse syntaxique ?

- **< id, 1 >**
- **< = >**
- **< id, 2 >**
- **< + >**
- **< id, 3 >**
- **< * >**
- **< nbr, 60 >**



Comment effectuer l’analyse syntaxique ?

- Dans l’analyse lexicale :
  - **Spécification** : Expressions régulières
  - **Implémentation** : Automates finis

- Question

Peut-on utiliser les expressions régulières & les automates finis pour l’analyse syntaxique ???

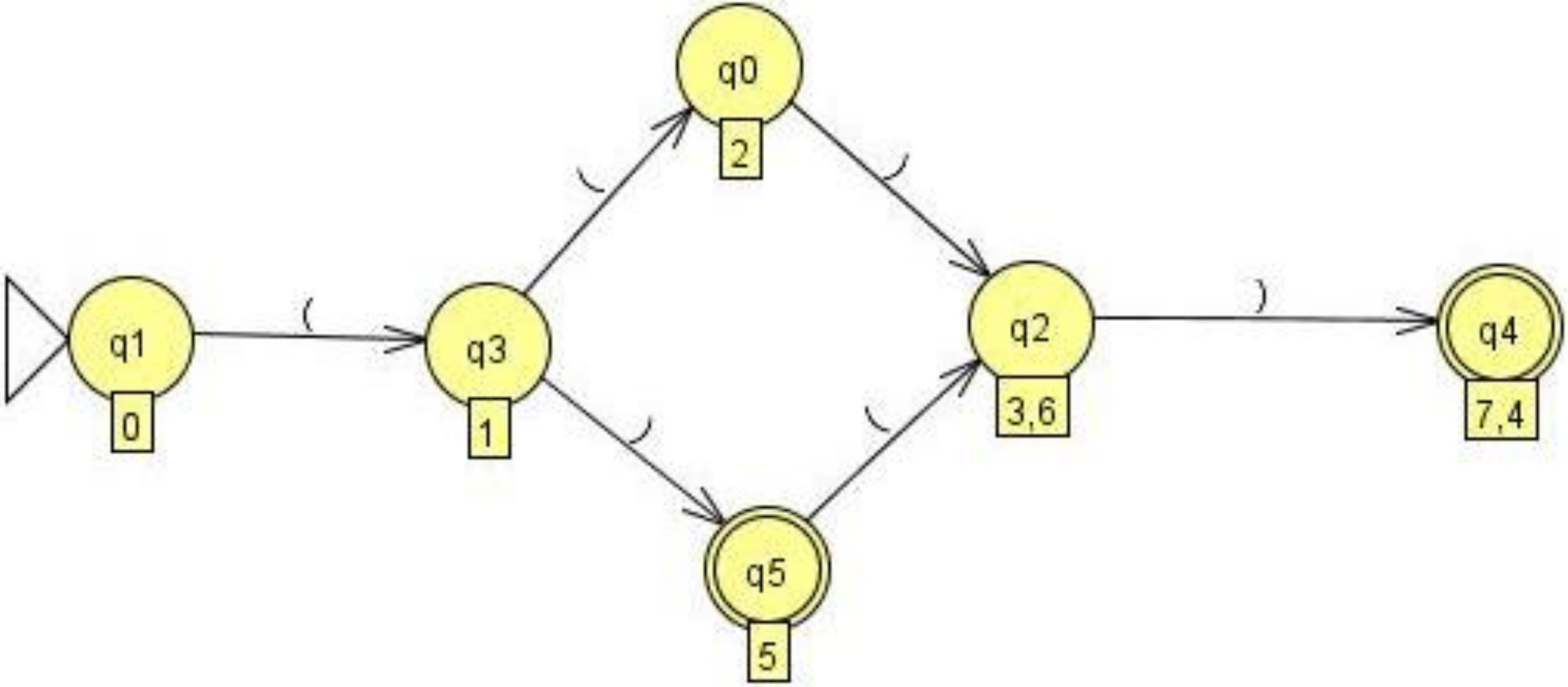
Exemple

- Une expression est bien parenthésée. si **le nombre de parenthèses ouvrantes** est **égal** au **nombre de parenthèses fermantes**

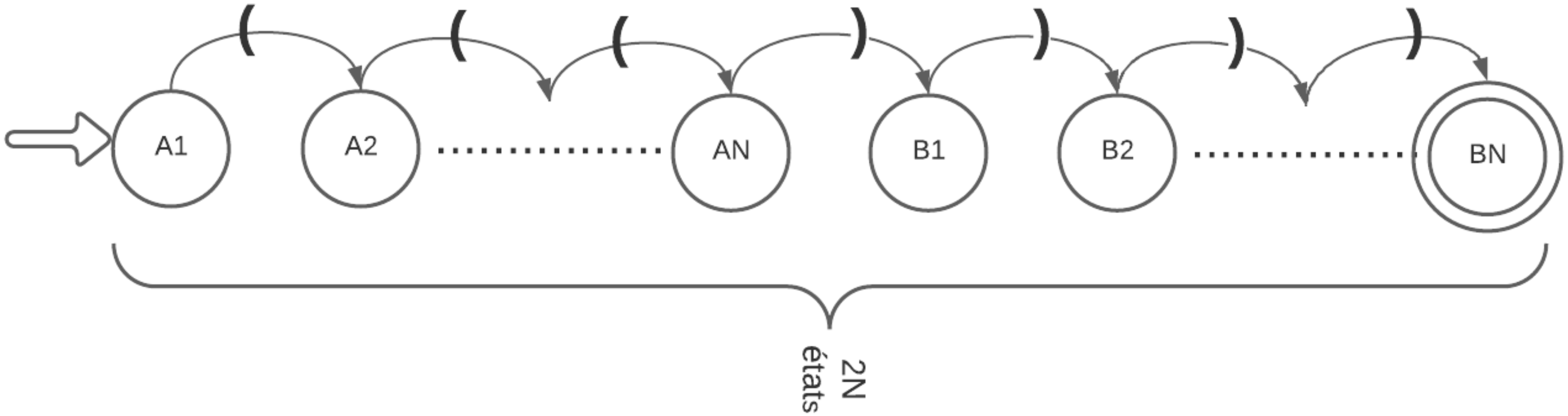
- (())
- ((()()))

-)(
- ((()

Exemple



Exemple



- Chemin avec **2N** états pour **N** « **(** » + **N** « **)** »
- Si on ajoute 2 « **(**» + « **)** » parenthèses → Ajout de 2 états
- Automate n'est pas fini ($N \rightarrow \infty$)

L'automate fini est incapable de compter

Problème des expression régulières et les automate finis

- L(Expressions régulières) = L(automates finis)
- Les automates finis

oublient l'historique

Expressions régulières (Automates finis)

- Incapable de spécifier la règle de parenthésage

- Plusieurs constructions de langages de programmation sont similaires au parenthésage et nécessitant le comptage :
  - **Pascal :** Begin End (nombre(**Begin**) = nombre (**End**))
  - **C:** { } (nombre(**{**) = nombre (**}**))

- En générale le langage $a^n b^n$ n'est pas un langage régulier

Nécessité d'un outil plus puissant

(Les grammaire non contextuelle)

Grammaire non contextuelle (Context-free grammar (CFG))

- $G =< S, T, N, P >$

- $S \in N$: **Symbole non terminal de départ**

- T: **les terminaux** sont les nom des unités lexicales
  - **Nbr, Id, =, +, if, then …..**

- N : **Les non terminaux** sont des variables syntaxiques utilisées durant les dérivations des chaines de terminaux
  - A,B, *expression, affectation* …..

- P: **les Productions** définissent les règles de remplacement de non terminaux
  - $\boldsymbol{N} \rightarrow (\boldsymbol{T} \cup \boldsymbol{N})^*$
  - **A**$\rightarrow$ $\boldsymbol{aA}$ **(exemple)**

Exemple:

- $G =< S, T, N, P >$
- Non terminal de départ :

$$S = Expr$$

- Terminaux:

$$T = \{id, nbr, +, -, *, (,)\}$$

- Non terminaux:

$$N = \{\boldsymbol{Expr}, \boldsymbol{Op}\}$$

- Productions:
  - $expr \ \rightarrow \ (expr) \mid expr\ op\ expr \mid \mathbf{id} \mid \mathbf{nbr}$
    $op \quad \rightarrow \ + \mid - \mid *$

Exemple:

- $expr \rightarrow (expr) \mid Expr\ op\ Expr \mid \mathbf{id} \mid \mathbf{nbr}$
  $op \rightarrow + \mid - \mid *$

- Mots acceptés par cette grammaire
  - $\mathbf{id}$
  - $\mathbf{nbr}$
  - $\mathbf{nbr + nbr}$
  - $\mathbf{id + nbr}$
  - $\mathbf{Id + (id + nbr)}$

- Mots non acceptés
  - $+$
  - $-$
  - $\mathbf{Id}\ +*$
  - $\mathbf{)nbr)}$

Conventions de notation

- Terminaux
  - **Premiers lettres miniscules :** a, b, c
  - **Opérateurs:** +,*, ….
  - **Signes de ponctuation**: Parenthèses, virgules ….
  - **Chiffres**: $0{,}1 \ldots \ldots, 9$
  - **Chaine de caractères gras : id, if, nbr,….** (une chaine représente **un seul alphabet**)

- Non terminaux
  - **Premiers lettres majuscules :** $\mathrm{A, B, C}$
  - S quand utilisé représente un **Non terminal de départ**
  - **Mots minuscules italique :** $expr, instr$
  - **D'autre lettres majuscules:** $\mathrm{E, T, F}$

Conventions de notation

- **Symboles grammaticaux génériques (Terminale ou Non-terminale):**
  - Derniers lettres majuscules: $\mathrm{X}, \mathrm{Y}, \mathrm{Z}$ (un seul symbole)
- **Chaine de terminaux (possiblement vide):**
  - Derniers lettres minuscules: $u, v, \ldots, z$ (chaine de terminaux)
- **Chaine de symbole grammaticaux (Terminaux ou Non-terminaux)**
  - Lettres grecques minuscules: α, β, γ (chaine qui contient tous les symboles)
- Si le **non terminal de départ** n'est pas spécifié, alors c'est **la partie gauche de la première production**.

Dérivation des mots

- **Grammaire :**
  - $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E} \mid \mathrm{E} * \mathrm{E} \mid \mathrm{E} \mid (\ \mathrm{E}\ ) \mid \mathbf{id}$

- Dérivation du mot $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$

$$
\begin{aligned}
\mathrm{E} &\xRightarrow{\mathrm{E} \rightarrow \mathrm{E} * \mathrm{E}} \mathrm{E} * \mathrm{E} \\
&\xRightarrow{\mathrm{E} \rightarrow (\mathrm{E})} (\mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E}} (\mathrm{E} + \mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E} \rightarrow \mathbf{id}} (\mathbf{id} + \mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E} \rightarrow \mathbf{id}} (\mathbf{id} + \mathbf{id}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E} \rightarrow \mathbf{id}} (\mathbf{id} + \mathbf{id}) * \mathbf{id}
\end{aligned}
$$

Dérivation des mots

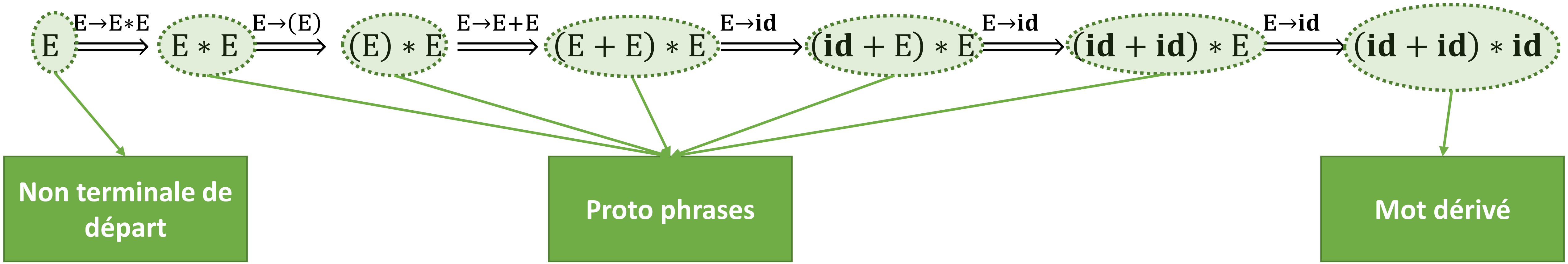
- **Grammaire :**
  - $\mathrm{E} \rightarrow \mathrm{E} + \mathrm{E}\,|\mathrm{E} * \mathrm{E}\,|\,\mathrm{E}\,|\,(\,\mathrm{E}\,)\,|\,\mathbf{id}$

- Dérivation du mot $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$

$$
\begin{aligned}
\mathrm{E} &\xRightarrow{\mathrm{E}\rightarrow\mathrm{E}*\mathrm{E}} \mathrm{E} * \mathrm{E} \\
&\xRightarrow{\mathrm{E}\rightarrow(\mathrm{E})} (\mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E}\rightarrow\mathrm{E}+\mathrm{E}} (\mathrm{E} + \mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E}\rightarrow\mathbf{id}} (\mathbf{id} + \mathrm{E}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E}\rightarrow\mathbf{id}} (\mathbf{id} + \mathbf{id}) * \mathrm{E} \\
&\xRightarrow{\mathrm{E}\rightarrow\mathbf{id}} (\mathbf{id} + \mathbf{id}) * \mathbf{id}
\end{aligned}
$$

Dérivation des mots

- Dérivation du mot $(\mathbf{id} + \mathbf{id}) * \mathbf{id}$



- $E \Rightarrow E * E \Longrightarrow (E) * E \Longrightarrow (E + E) * E \Longrightarrow (\mathbf{id} + E) * E \Longrightarrow (\mathbf{id} + \mathbf{id}) * E \Longrightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id}$

- $\boldsymbol{E} \overset{+}{\Rightarrow} (\mathbf{id} + \mathbf{id}) * \mathbf{id}$

Dérivation des mots

- Commencer à partir le non terminale de départ

- Remplacer à chaque étape **un non-terminale «A»** par **la partie droite « α »** d'une **production de la grammaire « $\mathbf{A} \rightarrow \boldsymbol{\alpha}$ »** :

$$S \overset{*}{\Rightarrow} \beta {\color{red}A} \gamma \xRightarrow{\mathrm{A} \rightarrow \alpha} \beta {\color{green}\alpha} \gamma$$

- Après plusieurs dérivations une proto phase contenant que des alphabets est produite. Cette proto phrase finale représente un mot **«** w **»** reconnu par la grammaire

$$S \overset{+}{\Rightarrow} w$$

- Le langage reconnu par une grammaire **« G »** inclus tous les mots qui peuvent être dérives par cette grammaire

$$LG(G) = \{w | S \overset{+}{\Rightarrow} w\}$$

Dérivations gauches (Leftmost derivations)

- Choisir toujours le non-terminal le plus à gauche

Chaine de terminaux

$$w\mathbf{A}\gamma \xRightarrow{A\to\alpha} w\boldsymbol{\alpha}\gamma$$

Chaine mixte
(terminaux & non terminaux)

- Exemple:
- $E \Rightarrow \mathbf{E} * E$

$\Rightarrow (\mathbf{E}) * E$

$\Rightarrow (\mathbf{E} + E) * E$

$\Rightarrow (\mathbf{id} + \mathbf{E}) * E$

$\Rightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{E}$

$\Rightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id}$

Dérivations droites (Rightmost derivations)

- Choisir toujours le non-terminal le plus à gauche

Chaine mixte
(terminaux & non terminaux)

$$\gamma A w \overset{A \to \alpha}{\Longrightarrow} \gamma \alpha \mathbf{w}$$

Chaine de terminaux

- Exemple:
- $E \Longrightarrow E * \mathbf{E}$

$\Longrightarrow E * \mathbf{E}$

$\Longrightarrow \mathbf{E} * \mathbf{id}$

$\Longrightarrow (\mathbf{E}) * \mathbf{id}$

$\Longrightarrow (E + \mathbf{E}) * \mathbf{id}$

$\Longrightarrow (\mathbf{E} + \mathbf{id}) * \mathbf{id}$

$\Longrightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id}$

Arbre d'analyse syntaxique

- $E \Longrightarrow \mathbf{E} * E$

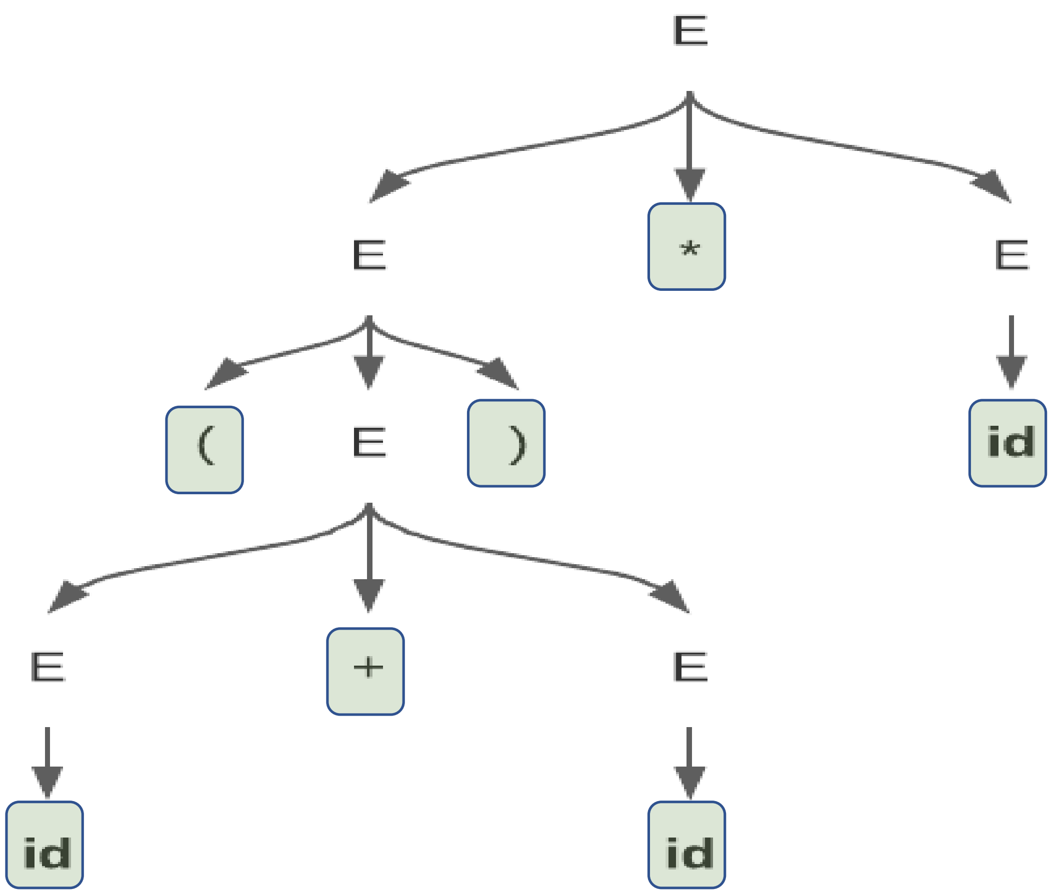
$\Longrightarrow (\mathbf{E}) * E$

$\Longrightarrow (\mathbf{E} + E) * E$

$\Longrightarrow (\mathbf{id} + \mathbf{E}) * E$

$\Longrightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{E}$

$\Longrightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id}$



Analyse syntaxique descendante

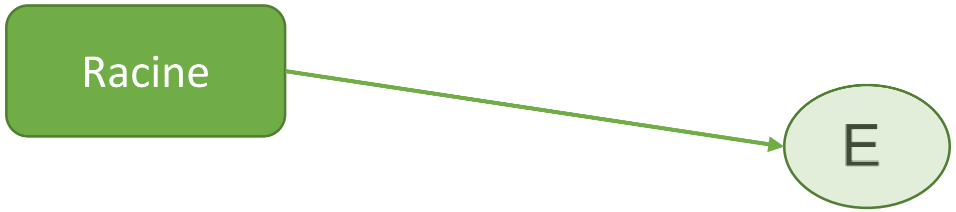
- Commencer à partir du non terminal de départ **(Racine de l'arbre)** pour arriver aux alphabets du mots (**Feuillies de l'arbre**)

- Remplacer un non terminale **«$\boldsymbol{A}$»** de la limite de l'arbre par une partie gauche **« $\boldsymbol{\alpha}$ »** d'une production **« $\mathbf{A} \rightarrow \boldsymbol{\alpha}$ »**

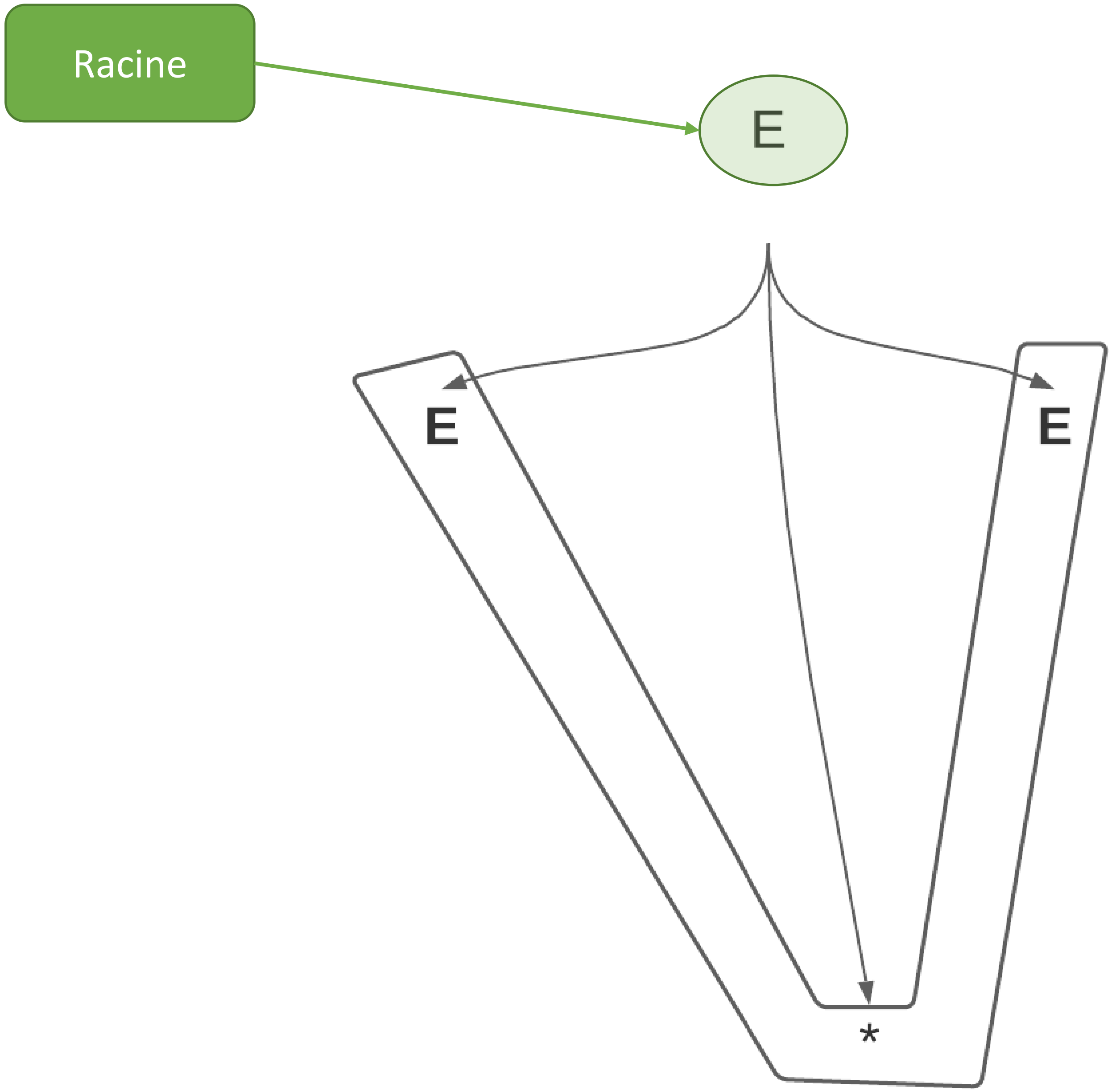
- Arrêter quand tous les non terminaux sont remplacés

- Construire l'arbre à partir la racine tous en descendant vers les feuillies

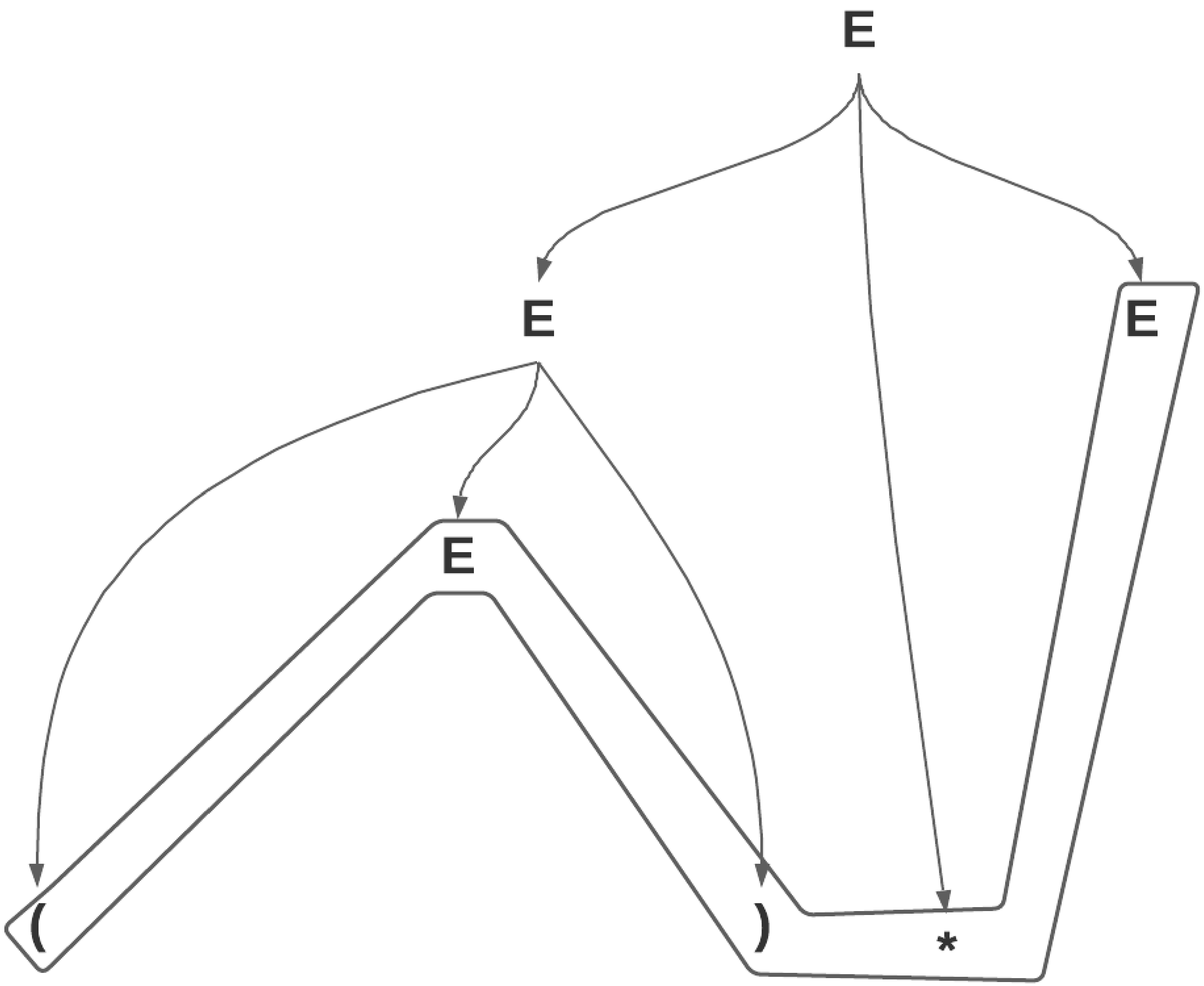
$\mathrm{E} \Longrightarrow$



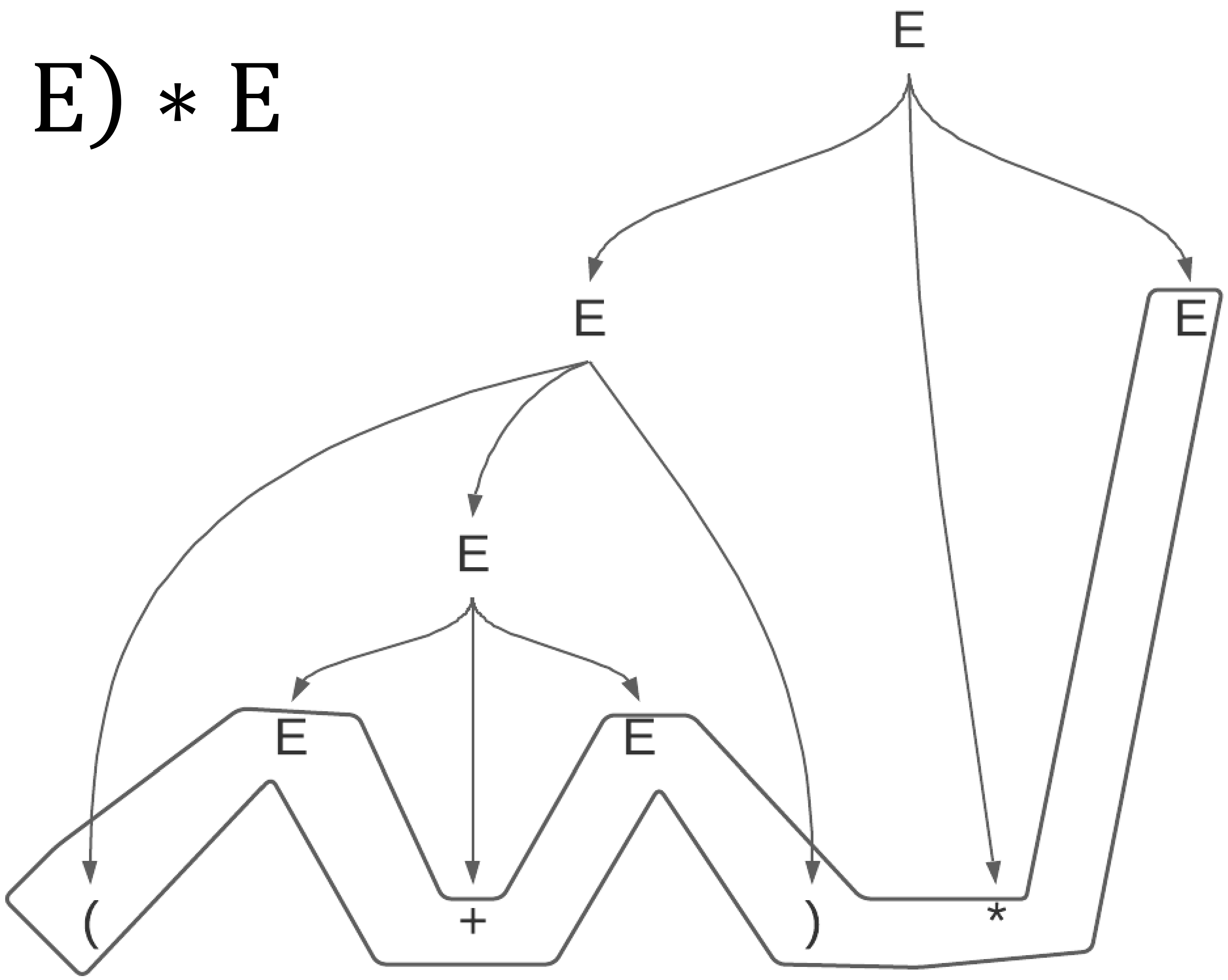
$$E \xRightarrow{E \to E * E} \color{green}{\mathbf{E}} \color{black} * E$$



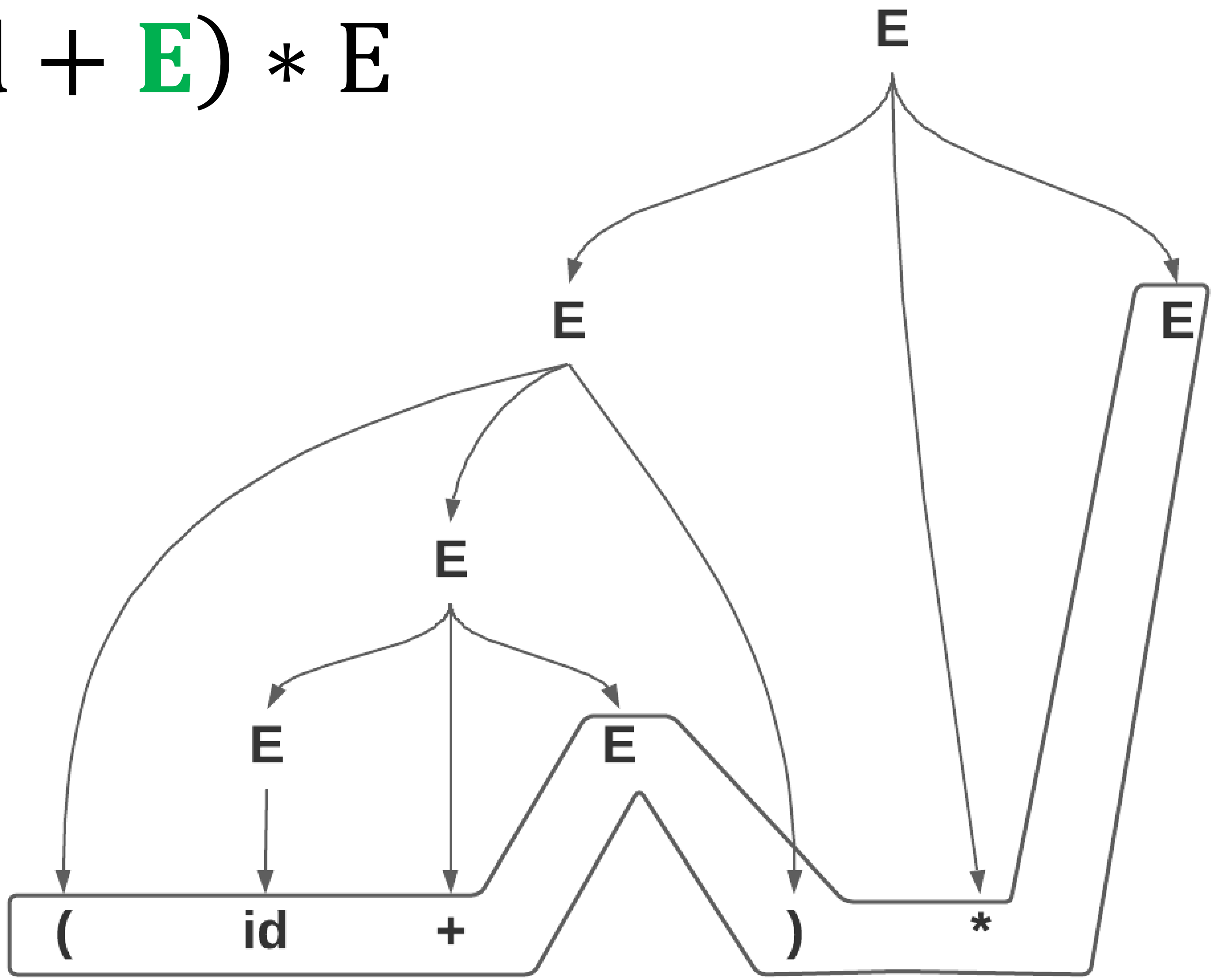
$$\mathbf{E} * \mathrm{E} \xRightarrow{\mathrm{E}\to(\mathrm{E})} (\mathbf{E}) * \mathrm{E}$$



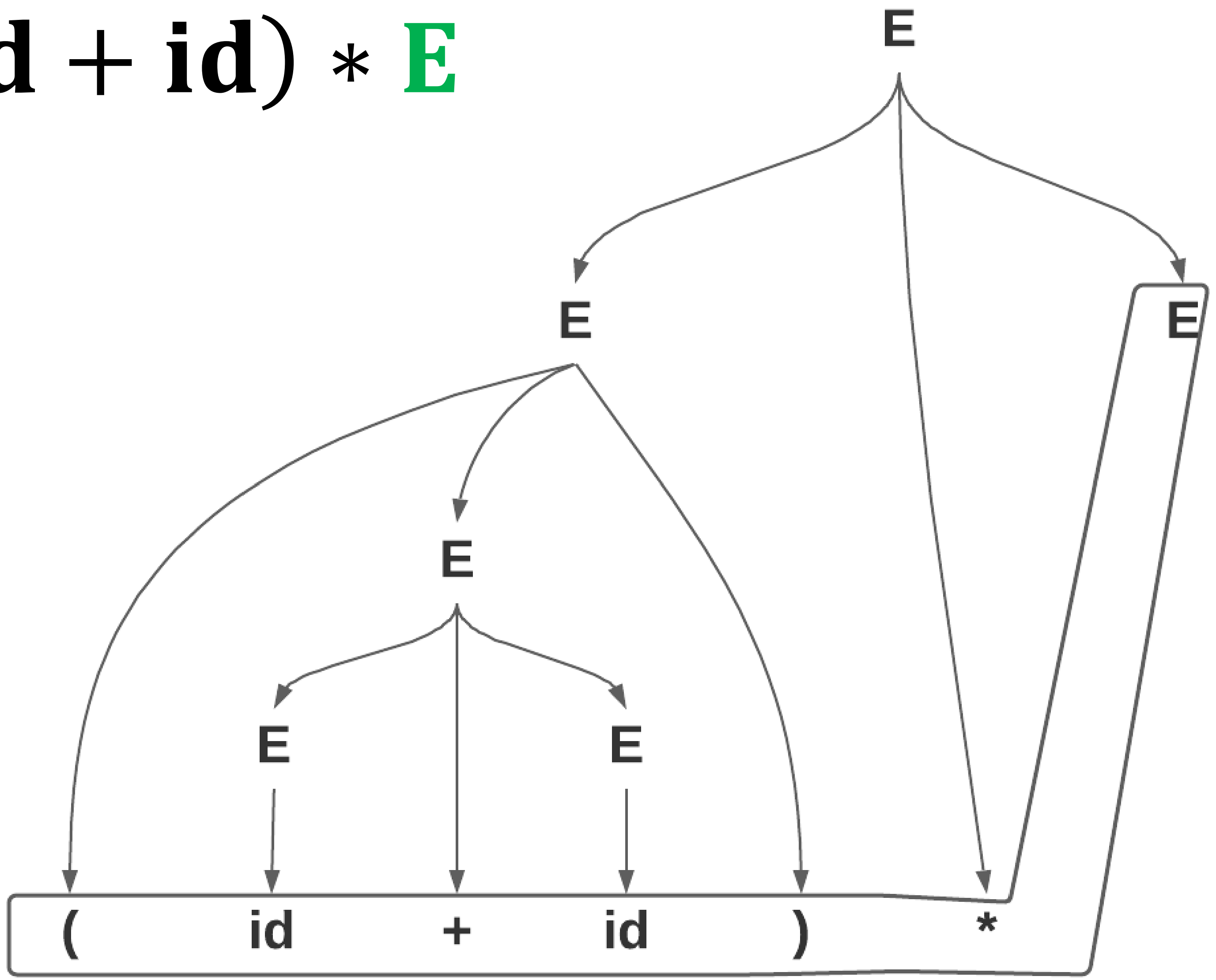
$$(\mathbf{E}) * \mathrm{E} \xRightarrow{\mathrm{E} \to \mathrm{E+E}} (\mathbf{E} + \mathrm{E}) * \mathrm{E}$$



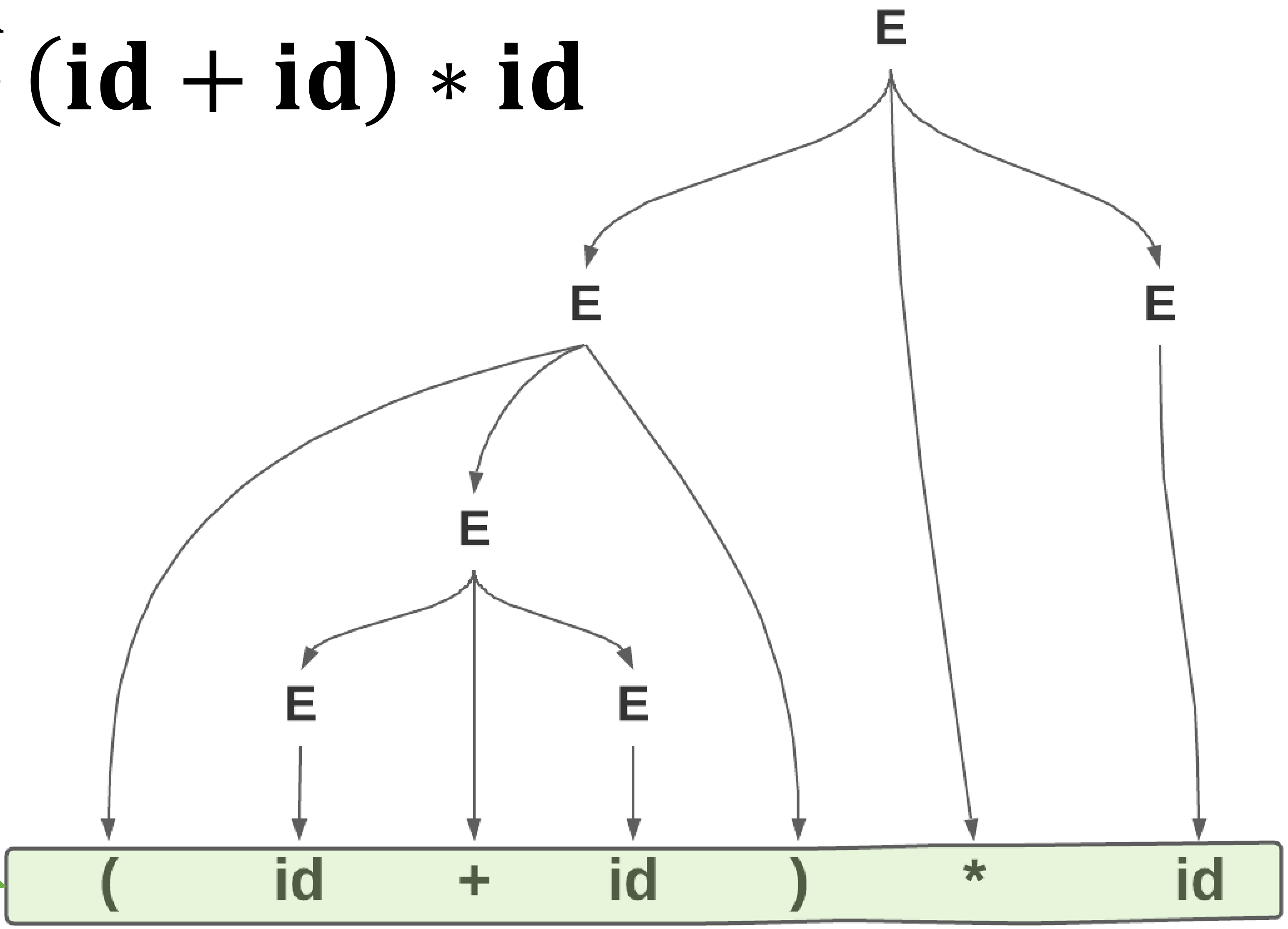
$$(\mathbf{E} + \mathrm{E}) * \mathrm{E} \xRightarrow{\mathrm{E}\to\mathbf{id}} (\mathbf{id} + \mathbf{E}) * \mathrm{E}$$



$$(\mathbf{id} + \mathbf{E}) * \mathrm{E} \xRightarrow{\mathrm{E}\to\mathbf{id}} (\mathbf{id} + \mathbf{id}) * \mathbf{E}$$



$$(\mathbf{id} + \mathbf{id}) * \mathbf{E} \xRightarrow{\mathbf{E \to id}} (\mathbf{id} + \mathbf{id}) * \mathbf{id}$$



Analyse syntaxique ascendante

- Commencer à partir le mot à reconnaitre (**Feuillies de l'arbre)** pour arriver à la (**Racine de l'arbre**)

- Remplacer une partie droite **«** α **»** d'une production **«** $\mathbf{A} \rightarrow \boldsymbol{\alpha}$ **»**

 par son non terminale du partie gauche **«**A**» (Réduction)**

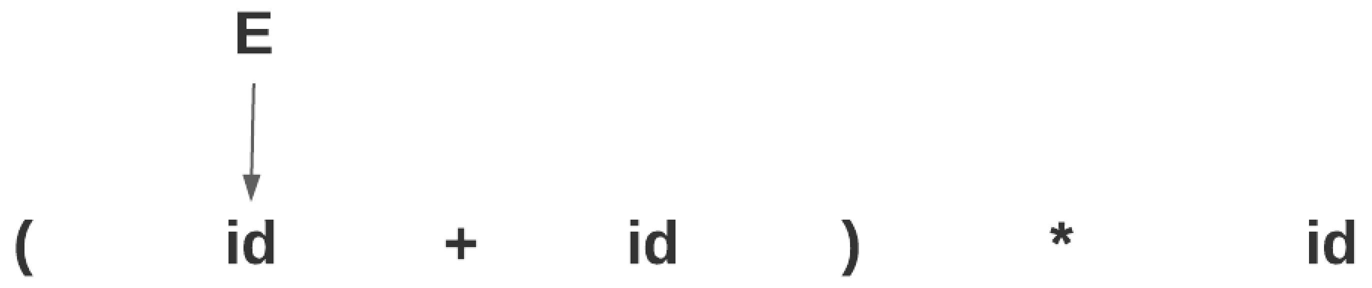
- Arrêter quand le non terminale de départ est atteint

- Construire l'arbre à partir les feuilles tous en montant vers la racine

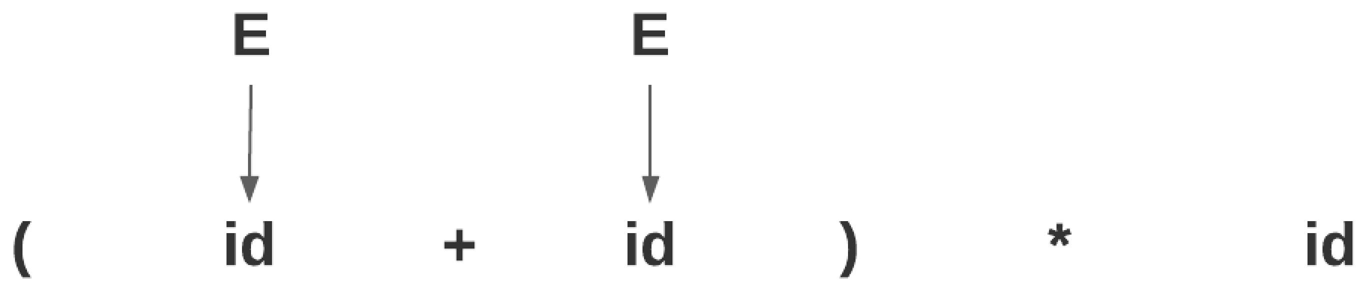
Mot dans les feuilles

(id + id) * id

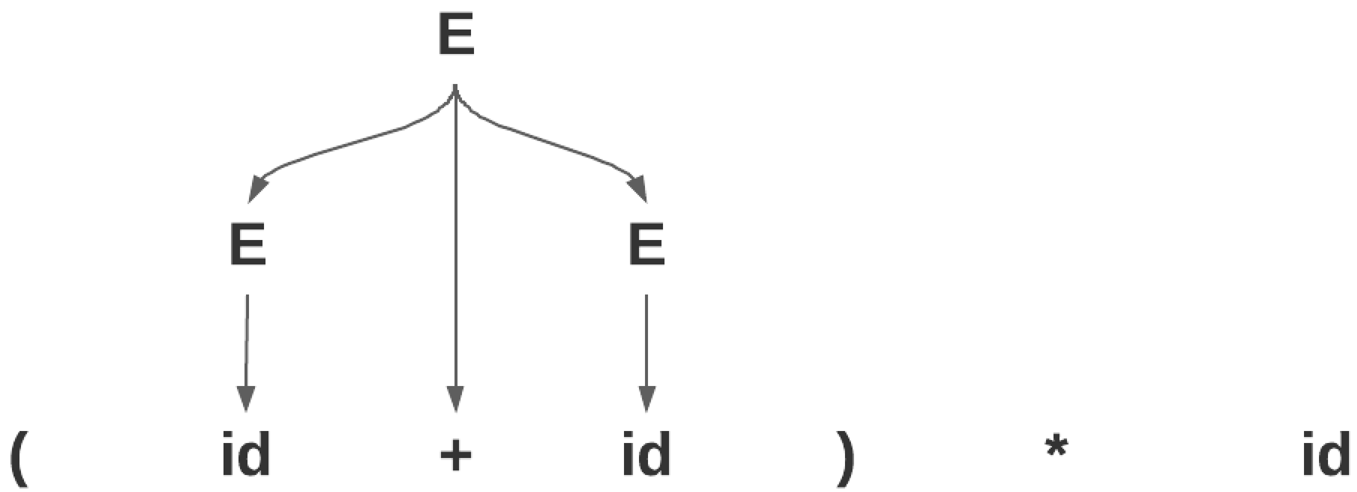
$$(\mathbf{E} + \mathbf{id}) * \mathbf{id} \Longrightarrow (\mathbf{id} + \mathbf{id}) * \mathbf{id}$$



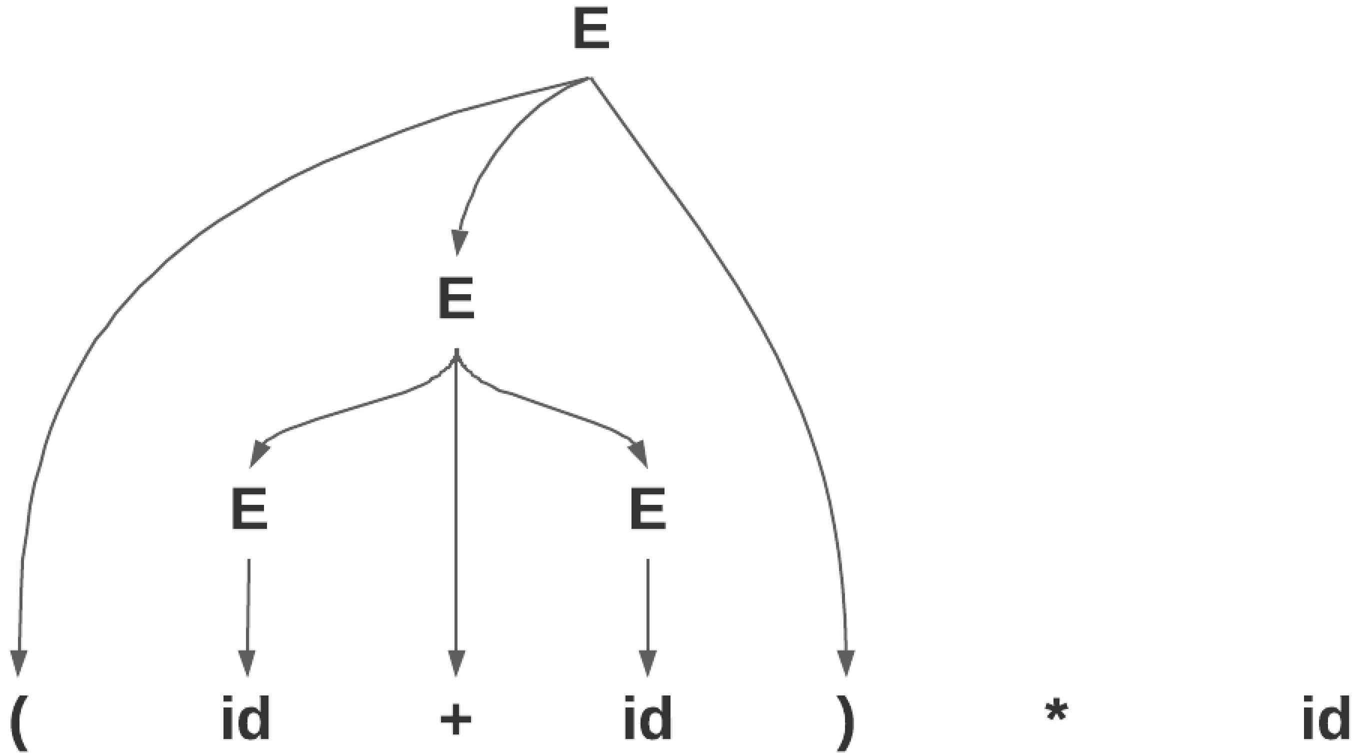
$$(\mathrm{E} + \mathbf{E}) * \mathbf{id} \Longrightarrow (\mathbf{E} + \mathbf{id}) * \mathbf{id}$$



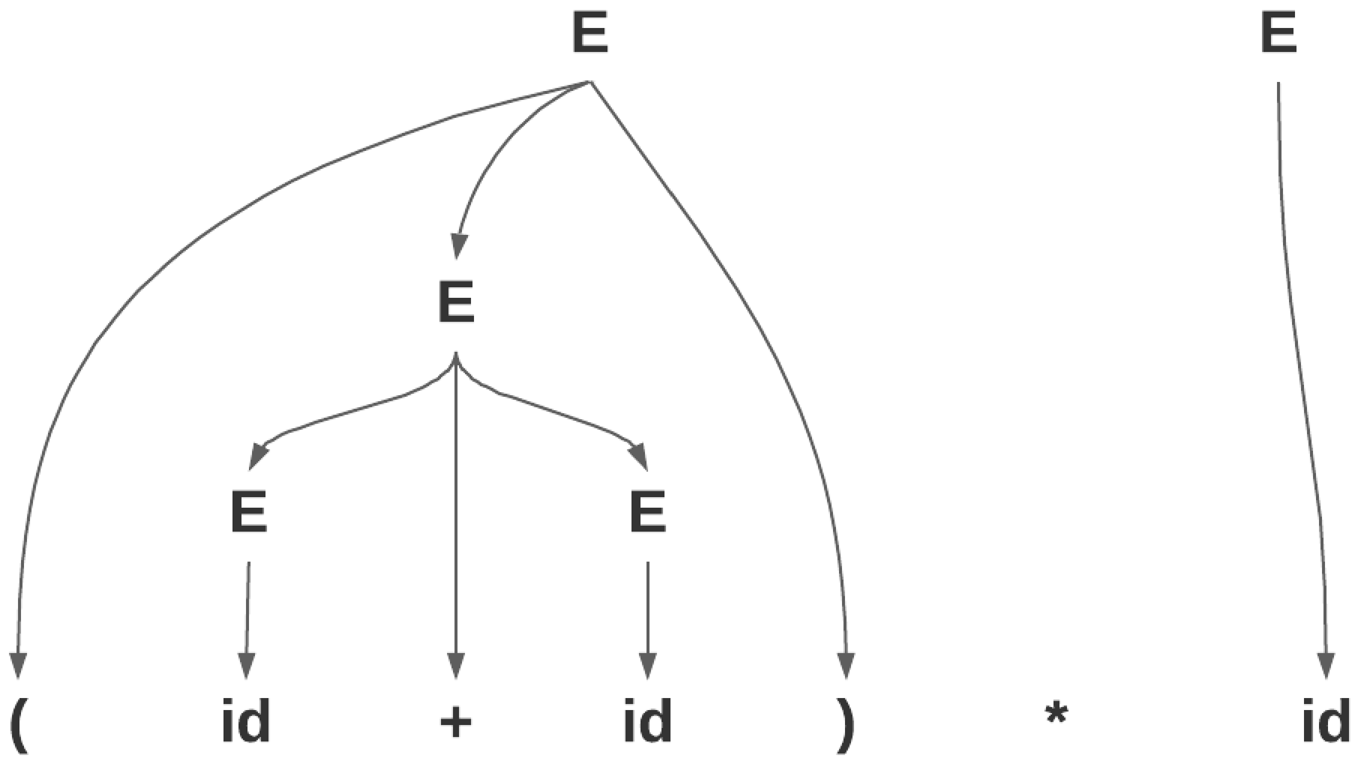
$$(\mathrm{E}) * \mathbf{id} \Longrightarrow (\mathrm{E} + \mathrm{E}) * \mathbf{id}$$



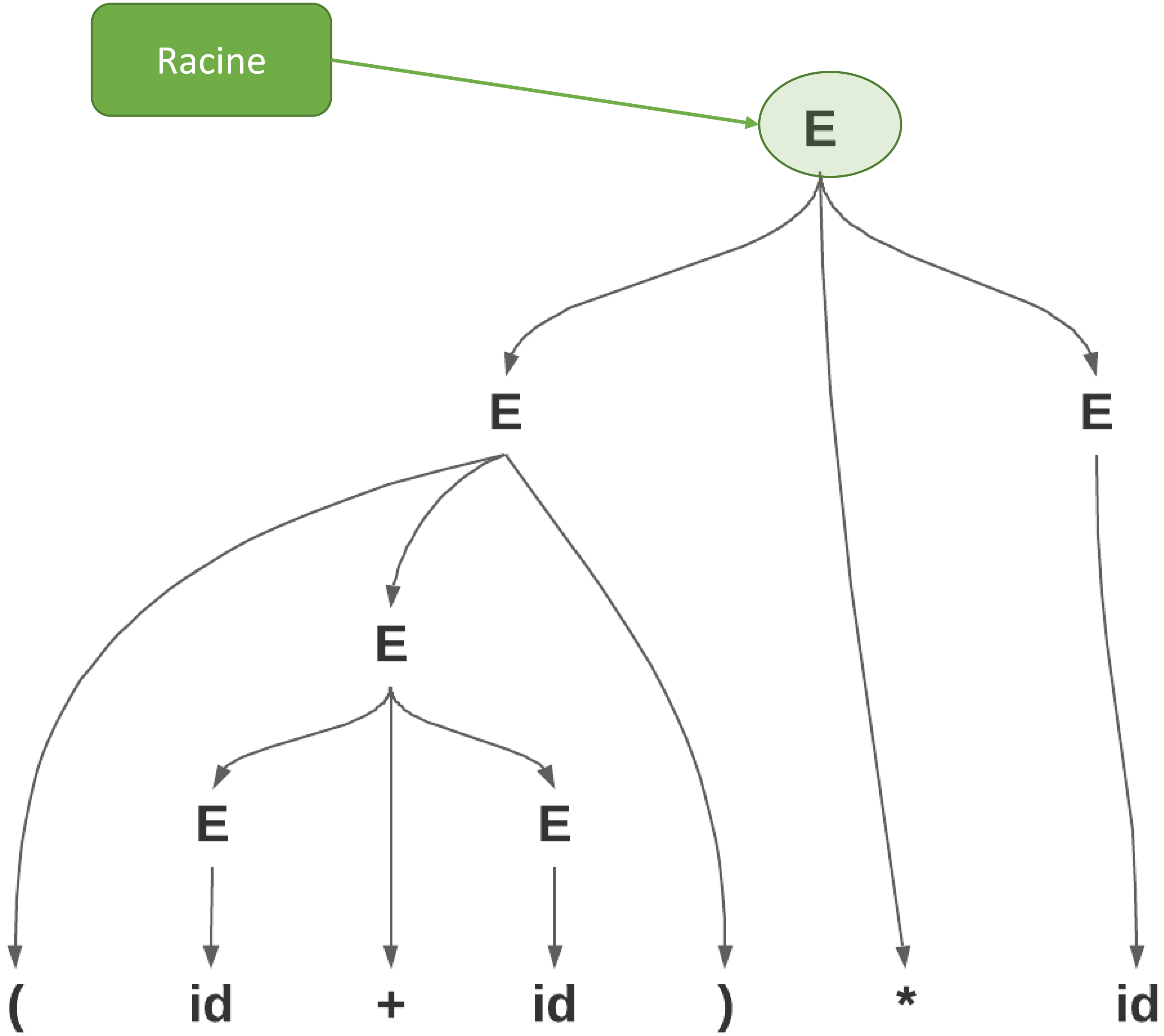
$$\mathbf{E} * \mathbf{id} \Longrightarrow (\mathbf{E}) * \mathbf{id}$$



$$E * \mathbf{E} \implies \mathbf{E} * \mathbf{id}$$



$$E \Longrightarrow E * \mathbf{E}$$



Ambiguïté

- Pour un mot donné il existe **plus d'un arbre d'analyse**

- Grammaire des expressions arithmétiques

$$E \;\rightarrow\; E + E \;|E * E \;|\; E \;|\; (\, E\,) \;|\; \mathbf{id}$$

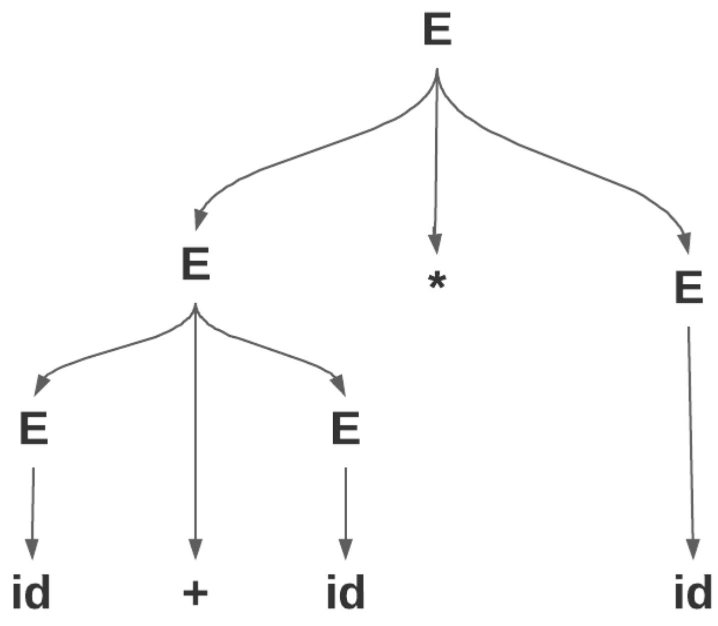
$$\boldsymbol{w = \; a + b * c}$$

Exemple

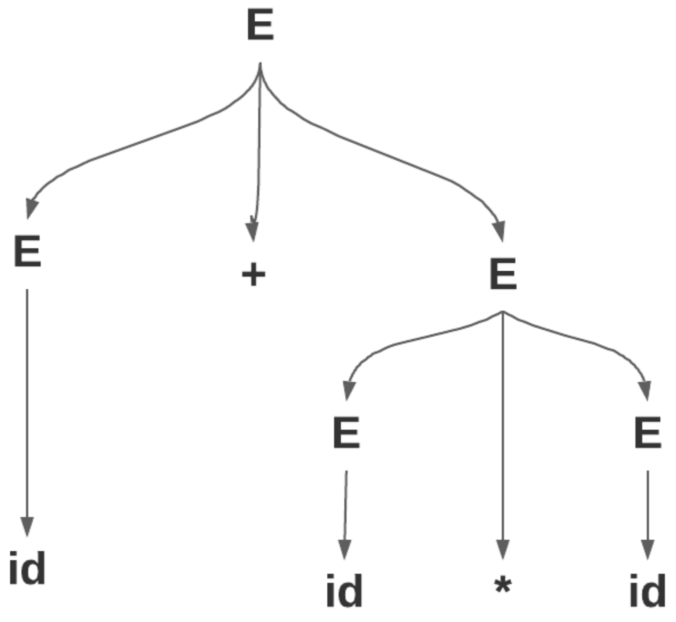
- Grammaire des expressions arithmétiques

$$E \rightarrow E + E \mid E * E \mid E \mid (E) \mid \mathbf{id}$$

$$\mathbf{w = id + id * id}$$



Grammaire ne considère pas la priorité des opérateurs

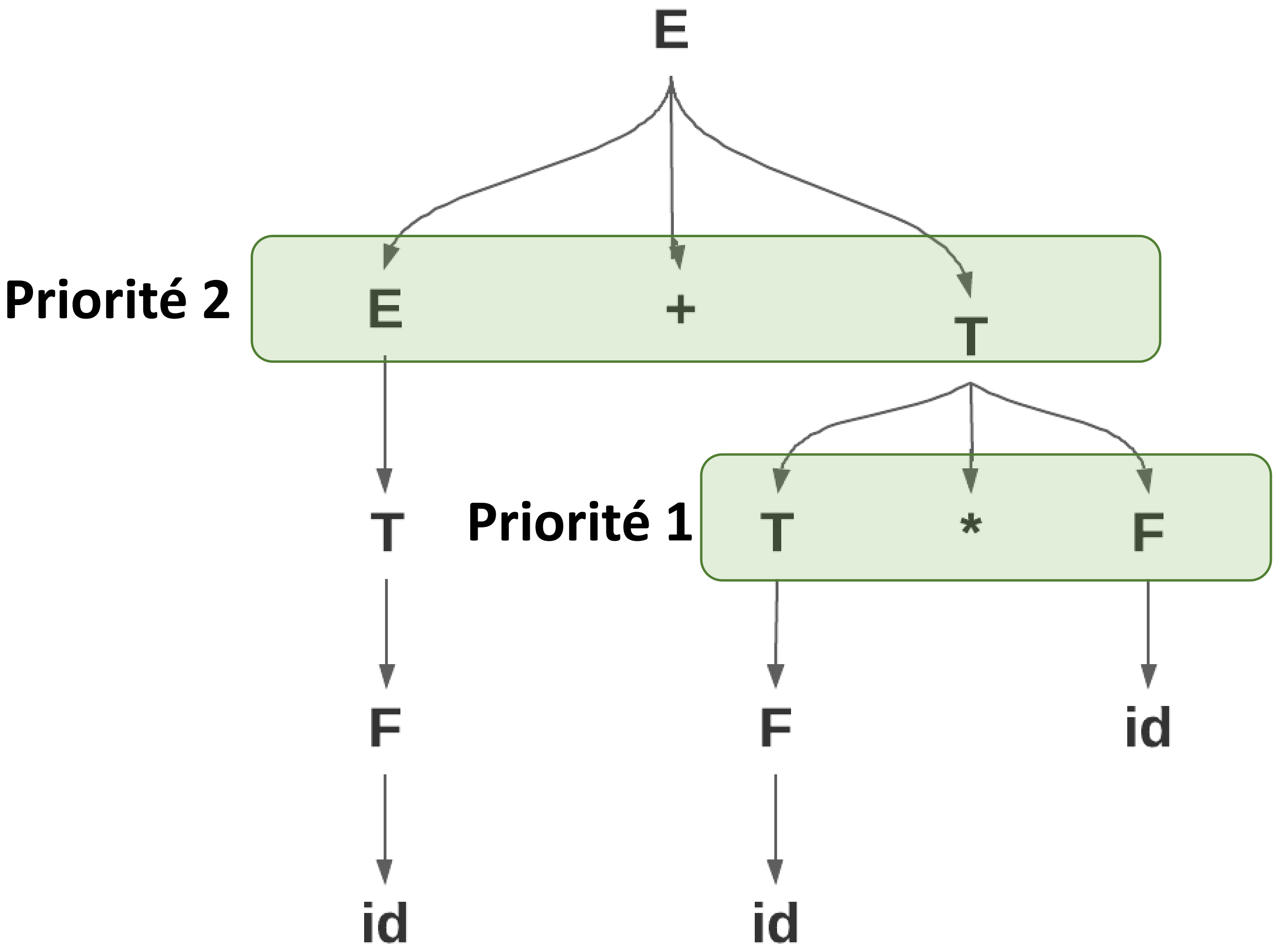


Enlever l'ambigüité

- $E \rightarrow E + T \quad | \, E - T \quad | \quad T$

- $T \rightarrow T * F \quad | \quad F$

- $F \rightarrow (E) \quad | \quad \mathbf{id}$



Enlever l'ambigüité

- Il n'existe pas un algorithme pour enlever l'ambiguïté de toutes les grammaires ambigües

- **Un langage n'est pas ambigu** peut avoir **une grammaire ambigüe**
  - L' ambiguïté de la grammaire et n'est pas du langage
  - On peut enlever l'ambiguïté

- Il existe des langages **intrinsèquement ambigus**
  - Toutes les grammaires de ce type de langage est ambiguës
  - On ne peut jamais enlever l'ambiguïté

Exemple:

- $L = \{a^l b^m c^n \mid m = l \ ou \ m = n\}$ est un langage **intrinsèquement ambigu**

Analyse syntaxique descendante

- Tester toutes les production pour un non terminale

- Si une production est échoues
  - Retour en arrière
  - Essayer une autre production

- Si une production est réussie
  - Si on est dans le non terminal de départ alors le mot es reconnu

Analyse syntaxique descendante

```
A ( )
Begin
        temp = pos /*position courante le mot d'entrée*/
        Pour chaque production Pi | A → X1X2 ... Xk /* tester toutes les productions*/
        Début
                pos ← temp
                Pour i allant de 1 à k faire /* parcourir une production*/
                Début
                        Si (Xi est un non terminal) Alors
                                Xi(   )
                        Sinon si (Xi est un terminal ET  Xi = mot[pos]) Alors
                                pos ← pos+1
                        Sinon
                                break /*production échouée*/
                Fin
                si (i == k)
                        break  /* production réussie*/
        Fin
        si (pos == temp)Alors
                Ecrire('Analyse échouée')
                Quitter le programme
        sinon si(A est le non terminal de départ ET pos =longueur(mot)) Alors
                Ecrire('mot reconnu')
Fin
```

Enlever la récursivité à gauche

- La **récursivité gauche** peut conduire à **une boucle infinie** quand **les dérivations gauches** sont utilisées

- Récursivité gauche directe

$$A \to \boldsymbol{A}\alpha|\beta$$

- $A \to A\alpha \to A\alpha\alpha \to A\alpha\alpha\alpha \to \cdots \to A\alpha^* \to \beta\alpha^*$

$$A \quad \to \beta A'$$
$$A' \ \to \alpha A' \mid \varepsilon$$

Exemple

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (E) \mid \mathbf{id}
\end{array}
$$

Factorisation de la grammaire

$$A \rightarrow \boldsymbol{\alpha}\beta_1 | \boldsymbol{\alpha}\beta_2 | \boldsymbol{\alpha}\beta_3 | \ldots | \boldsymbol{\alpha}\beta_n | \gamma$$

Après factorisation

$$A \rightarrow \boldsymbol{\alpha}B | \gamma$$
$$B \rightarrow \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n |$$

Exemple

- $instr \ \rightarrow \mathbf{si}\ expr\ \mathbf{Alors}\ instr\ \mathbf{sinon}\ instr\ \mathbf{|}\ \mathbf{si}\ expr\ \mathbf{Alors}\ instr$

- $instr \ \rightarrow \mathbf{si}\ expr\ \mathbf{Alors}\ instr\ sinon\_instr$
- $sinon\_instr \ \rightarrow \mathbf{sinon}\ instr|\ \varepsilon$

Analyse descendante LL(1)

- Elle analyse un mot d'entrée de gauche à droite (***L**eft to right* en anglais)

- Construit une dérivation à gauche (***L**eftmost derivation* en anglais).

- Un seul caractère de prédiction

Ensemble Premier

- $Premier\ (\alpha) = \{a \in T \cup \{\varepsilon\} |\ \alpha \overset{+}{\Rightarrow} a\beta\}$

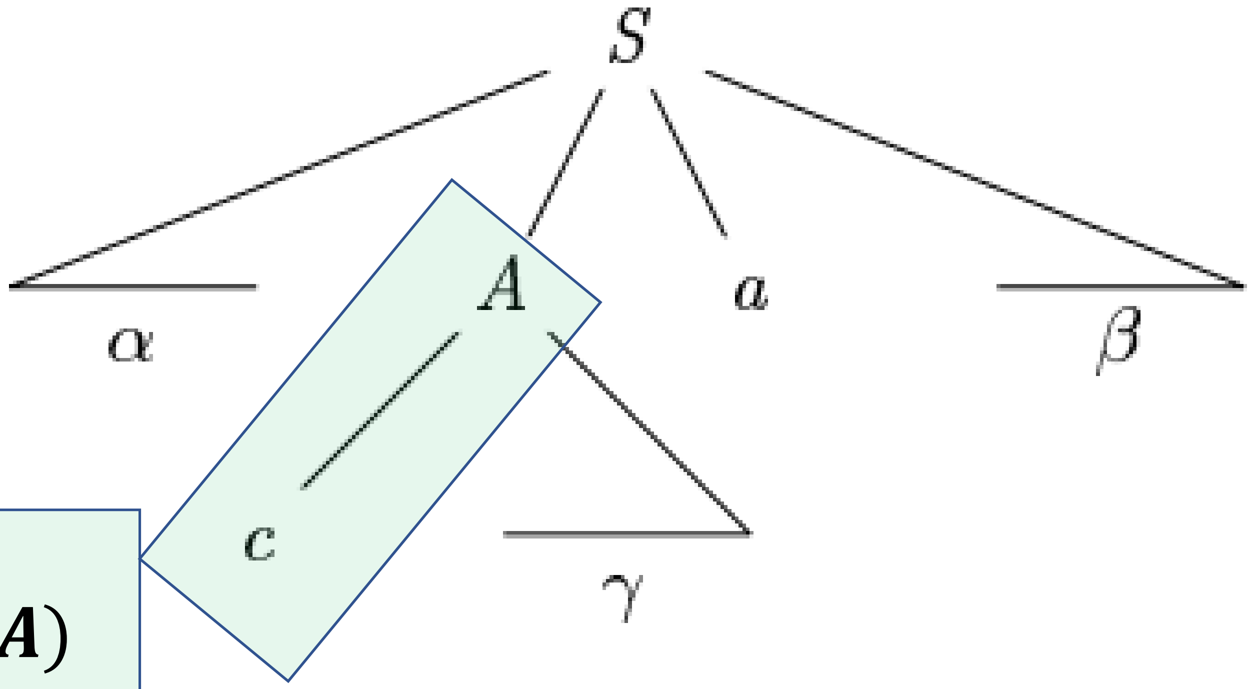
- L’ensemble $\boldsymbol{Premier}$ continent des **terminaux** ou $\boldsymbol{\varepsilon}$

- L'ensemble des terminaux qui peuvent apparaître **en premier (en début)** d’une proto-phrase après **une ou plusieurs dérivations** commençant de la chaine $\boldsymbol{\alpha}$

$$A \overset{+}{\Rightarrow} \boldsymbol{a}B$$

Ensemble Premier

- $Premier\ (\alpha) = \{a \in T \cup \{\varepsilon\} |\ \alpha \overset{+}{\Rightarrow} a\beta\}$



$\boldsymbol{c\ \in Premier\ (A)}$

Calcule Premier

- Si X est un terminal

  Premier(X) = {X}

- Si X est un non-terminal et $X \rightarrow Y_1 Y_2 \dots Y_k$

  Premier(X) = Premier(Y_1) Si ε ∉ Premier(Y_1)

  Premier(X) = (Premier(Y_1) ∪ Premier(Y_2))-{ε}

  Si ε ∈ **Premier($\boldsymbol{Y_1}$)** et ε ∉ **Premier($\boldsymbol{Y_2}$)**

  Premier(X) = Premier(Y_1) ∪ ⋯ ∪ Premier(Y_j))−{ε}

  Si ε ∈ **Premier($\boldsymbol{Y_1}$)** et ε ∈ **Premier($\boldsymbol{Y_2}$)** … ε ∈ **Premier($\boldsymbol{Y_{j-1}}$)** et ε ∉ **Premier($\boldsymbol{Y_j}$) : $\boldsymbol{j \le k}$**

  ε ∈ Premier(X)

  Si ε ∈ **Premier($\boldsymbol{Y_1}$)** et ε ∈ **Premier($\boldsymbol{Y_2}$)** … ε ∈ **Premier($\boldsymbol{Y_k}$)**

Calcule Premier

- Après avoir calculer les ensembles **Premier** pour tous les symboles, on peut calculer **Premier** pour chaque n'importe chaine **mixte** $\boldsymbol{\alpha}$

$$\text{Si } \alpha = Y_1 Y_2 \ldots Y_k$$

- Premier (α) = Premier$(Y_1) \cup \cdots \cup$ Premier$(Y_j) - \{\varepsilon\}$

  Si $\varepsilon \in$ **Premier**$(\boldsymbol{Y_1})$ et $\varepsilon \in \cdots \varepsilon \in$ **Premier**$(\boldsymbol{Y_{j-1}})$ et $\varepsilon \notin$ **Premier**$(\boldsymbol{Y_j})$

- $\varepsilon \in$ Premier(X)

  Si $\varepsilon \in$ **Premier**$(\boldsymbol{Y_1})$ et $\varepsilon \in$ **Premier**$(\boldsymbol{Y_2})$ … $\varepsilon \in$ **Premier**$(\boldsymbol{Y_k})$

Ensemble Suivant

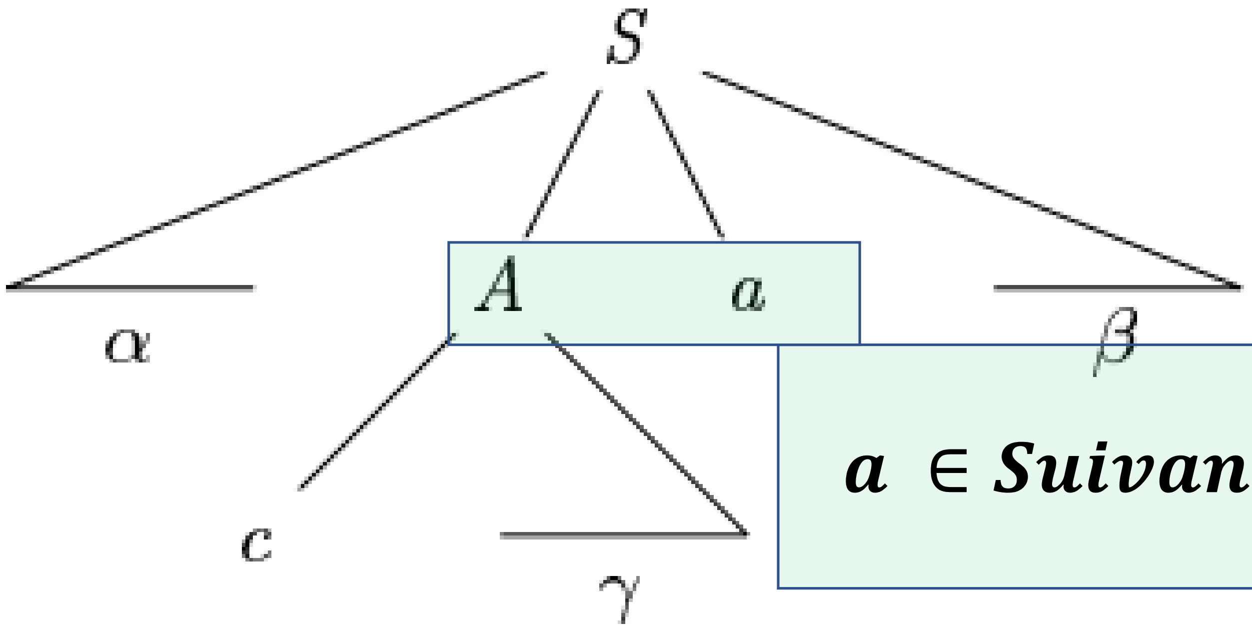
- $Suivant(\alpha) = \{a \in Premier(\gamma) |\ \alpha \overset{+}{\Rightarrow} \beta \alpha \boldsymbol{\gamma}\}$

- L'ensemble **Suivant** contient les **terminaux** qui peuvent apparaître **juste après un symbole** dans une suite de dérivations

$$\alpha \overset{+}{\Rightarrow} \beta \alpha \boldsymbol{\gamma}$$

Ensemble Suivant

- $Suivant\ (\alpha) = \{a \in Premier(\gamma) |\ \alpha \overset{+}{\Rightarrow} \beta\alpha\boldsymbol{\gamma}\}$



$\boldsymbol{a} \in \boldsymbol{Suivant(A)}$

Calcule Suivant

- Mettre **\$** dans **Suivant(S)**
  - Si S est un non-terminale de départ

- Mettre **Premier($\boldsymbol{\beta}$)-{$\boldsymbol{\varepsilon}$}** dans **Suivant(B)**
  - Si il existe une production $A \rightarrow \alpha \boldsymbol{B\beta}$

- Mettre **Suivant(A)** dans **Suivant(B)**
  - Si il existe une production $A \rightarrow \alpha \boldsymbol{B\beta}$
  - Si $\boldsymbol{\varepsilon}$ est dans **Premier($\boldsymbol{\beta}$)** ou $\boldsymbol{\beta}$= $\boldsymbol{\varepsilon}$

Analyseur syntaxique descendant prédictif

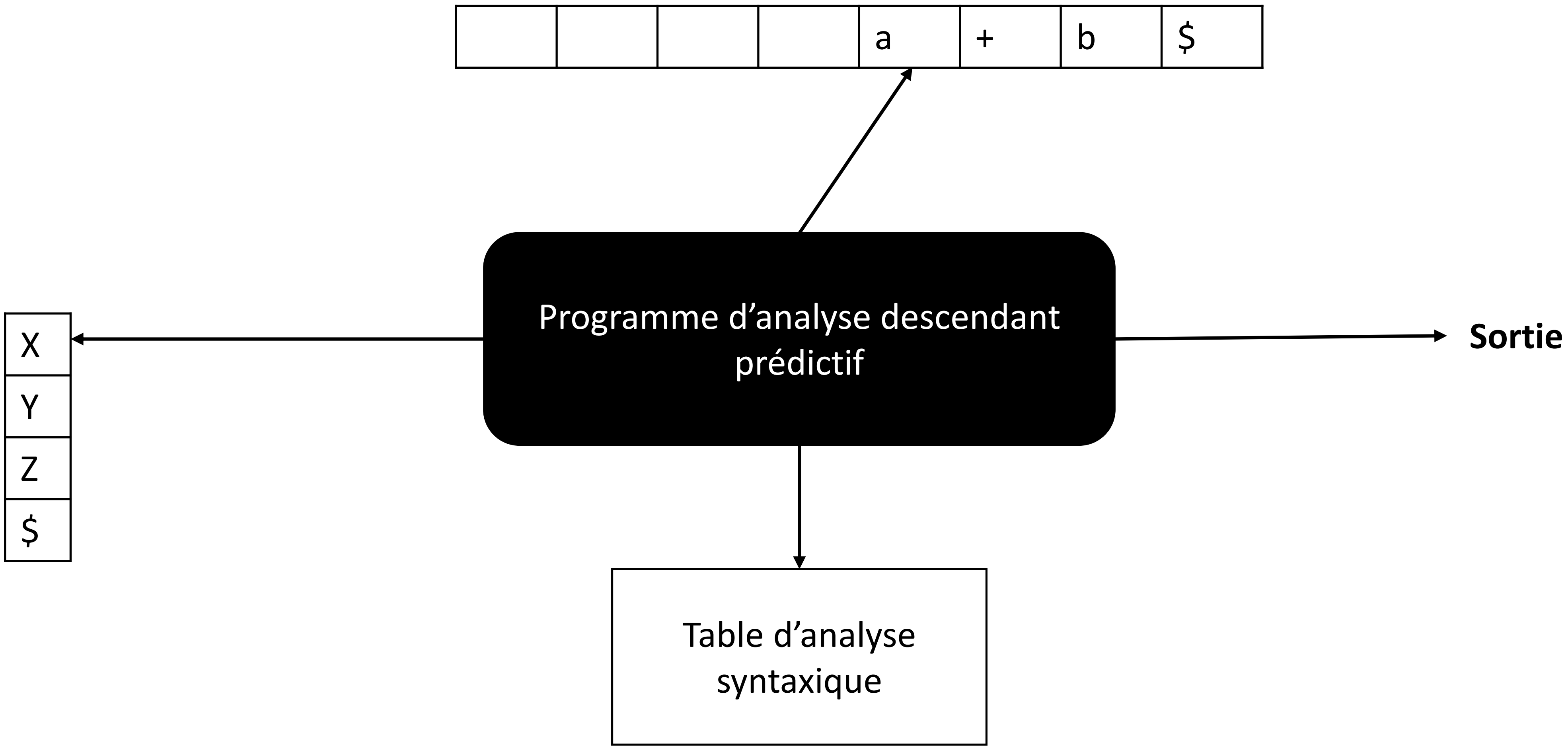


Table d'analyse LL(a)

Si $\boldsymbol{a_i} \in \boldsymbol{Premier(\alpha)}$

	$\boldsymbol{a_0}$	……	$\boldsymbol{a_i}$		$
$\boldsymbol{A_0}$					
$\boldsymbol{A_i}$			$\boxed{\boldsymbol{A_i \rightarrow \alpha}}$		
$\boldsymbol{A_n}$					

Si $\boldsymbol{\varepsilon} \in \boldsymbol{Premier(\alpha)}$ et $\boldsymbol{a_i} \in \boldsymbol{Suivant(A_i)}$

Table d’analyse LL(a)

Si $\boldsymbol{a_i} \in \boldsymbol{Premier}(\boldsymbol{Premier}(\boldsymbol{\alpha})$
$\boldsymbol{Suivant}(\boldsymbol{A_i}))$

	$\boldsymbol{a_0}$	……	$\boldsymbol{a_i}$		\$
$\boldsymbol{A_0}$					
$\boldsymbol{A_i}$			$\boxed{\boldsymbol{A_i \rightarrow \alpha}}$		
$\boldsymbol{A_n}$					

Table d'analyse LL(a)

Accéder à une Case vide → ERREUR

Deux dérivations possibles → Grammaire n'est pas LL(1)

	a_0	……	a_i		$
A_0					
A_i			$A_i \rightarrow \alpha$ $A_i \rightarrow \beta$		
A_n					

Reconnaissance d’un mot

- Dans le TD