Analyse syntaxique : grammaires, scanners, parsers

Sergiu IVANOV

sergiu.ivanov@u-pec.fr

Les diapos et le code disponibles en ligne :

http://lacl.fr/~sivanov/doku.php?id=fr:cours_de_theorie_des_langages

Hiérarchie de Chomsky

Noam Chomsky, un linguiste et philosophe américain, a proposé une classification des grammaires en 4 types, selon la forme des règles.

Hiérarchie de Chomsky

Noam Chomsky, un linguiste et philosophe américain, a proposé une classification des grammaires en 4 types, selon la forme des règles.

Soit la grammaire $G = (N, T, P, S)$

- N – symboles non-terminaux (*de travail*)
- T – symboles terminaux (*forment les mots*)
- P – règles de réécriture
- S – symbole de départ

Hiérarchie de Chomsky

Noam Chomsky, un linguiste et philosophe américain, a proposé une classification des grammaires en 4 types, selon la forme des règles.

Soit la grammaire $G = (N, T, P, S)$

- N – symboles non-terminaux (*de travail*)
- T – symboles terminaux (*forment les mots*)
- P – règles de réécriture
- S – symbole de départ

Les non-terminaux sont souvent écrits avec des majuscules (A), alors que les terminaux s'écrivent avec des minuscules (c).

Grammaires de type 0 : langages récursivement énumérables

Soit la grammaire $G = (N, T, P, S)$

Aucune restriction n'est imposée sur la forme des règles (presque).

Grammaires de type 0 : langages récursivement énumérables

Soit la grammaire $G = (N, T, P, S)$

Aucune restriction n'est imposée sur la forme des règles (presque).

Soit $u \to v \in P$ une règle

Grammaires de type 0 : langages récursivement énumérables

Soit la grammaire $G = (N, T, P, S)$

Aucune restriction n'est imposée sur la forme des règles (presque).

Soit $u \to v \in P$ une règle

- $v \in (N \cup T)^*$

  partie droite : aucune restriction

Grammaires de type 0 : langages récursivement énumérables

Soit la grammaire $G = (N, T, P, S)$

Aucune restriction n'est imposée sur la forme des règles (presque).

Soit $u \to v \in P$ une règle

- $v \in (N \cup T)^*$

  partie droite : aucune restriction
- $u \in (N \cup T)^* \; A \; (N \cup T)^*$

  partie gauche : doit contenir un non-terminal
  - pas drôle sinon

Grammaires de type 0 : langages récursivement énumérables

Soit la grammaire $G = (N, T, P, S)$

Aucune restriction n'est imposée sur la forme des règles (presque).

Soit $u \to v \in P$ une règle

- $v \in (N \cup T)^*$

  partie droite : aucune restriction
- $u \in (N \cup T)^* \; A \; (N \cup T)^*$

  partie gauche : doit contenir un non-terminal
  - pas drôle sinon

Engendrent les langages récursivement énumérables

- notation : RE

Grammaires de type 1 : langages à contexte sensible

Soit la grammaire $G = (N, T, P, S)$

Les règles doivent être croissantes

- quelle que soit $u \to v \in P$, $|u| \leq |v|$

Grammaires de type 1 : langages à contexte sensible

Soit la grammaire $G = (N, T, P, S)$

Les règles doivent être croissantes

- quelle que soit $u \to v \in P$, $|u| \leq |v|$

De façon équivalente, une règle doit réécrire un non-terminal dans un contexte, sans effacement :

$$\alpha A \beta \to \alpha w \beta,$$

$$A \in N,\ \alpha, \beta \in (N \cup T)^*,\ w \in (N \cup T)^+\ (w \neq \varepsilon)$$

Grammaires de type 1 : langages à contexte sensible

Soit la grammaire $G = (N, T, P, S)$

Les règles doivent être croissantes

- quelle que soit $u \to v \in P$, $|u| \leq |v|$

De façon équivalente, une règle doit réécrire un non-terminal dans un contexte, sans effacement :

$$\alpha A \beta \to \alpha w \beta,$$

$$A \in N,\ \alpha, \beta \in (N \cup T)^*,\ w \in (N \cup T)^+\ (w \neq \varepsilon)$$

Engendrent les langages à contexte sensible

- notation : CS (context sensitive)

Grammaires de type 2 : langages algébriques

Soit la grammaire $G = (N, T, P, S)$

Les règles réécrivent des non-terminaux, un par un :

$$A \longrightarrow u,$$

$$A \in N,\ u \in (N \cup T)^*$$

Grammaires de type 2 : langages algébriques

Soit la grammaire $G = (N, T, P, S)$

Les règles réécrivent des non-terminaux, un par un :

$$A \to u,$$

$$A \in N,\ u \in (N \cup T)^*$$

Engendrent les langages algébriques (hors contexte)

- notation : CF (context free)
- notation : ALG

Grammaires de type 2 : langages algébriques

Soit la grammaire $G = (N, T, P, S)$

Les règles réécrivent des non-terminaux, un par un :

$$A \to u,$$

$$A \in N,\ u \in (N \cup T)^*$$

Engendrent les langages algébriques (hors contexte)

- notation : CF (context free)
- notation : ALG

Langages de programmation $\subseteq$ Langages algébriques

Grammaires de type 3 : langages réguliers

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$, $c \in T$. Deux formes de règle sont admises :

$$A \rightarrow cB \qquad\qquad A \rightarrow c$$

Grammaires de type 3 : langages réguliers

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$, $c \in T$. Deux formes de règle sont admises :

$$A \rightarrow cB \qquad\qquad A \rightarrow c$$

Engendrent les langages réguliers (rationnels)

- notation : REG

Grammaires de type 3 : langages réguliers

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$, $c \in T$. Deux formes de règle sont admises :

$$A \rightarrow cB \qquad\qquad A \rightarrow c$$

Engendrent les langages réguliers (rationnels)

- notation : REG

Langages acceptés par les automates finis (clin d'œil au futur ;-))

La hiérarchie de Chomsky

Récursivement énumérables	$u \to v$	*RE*
$\bigcup$		
À contexte sensible	$\alpha A \beta \to \alpha w \beta$	*CS*
$\bigcup$		
Algébriques	$A \to u$	*CF*
$\bigcup$		
Réguliers	$A \to cB$, $A \to c$	*REG*

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

Types de règles linéaires

constante $A \rightarrow \alpha$

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

Types de règles linéaires

constante	$A \rightarrow \alpha$
linéaire	$A \rightarrow \alpha B \beta$

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

Types de règles linéaires

constante	$A \rightarrow \alpha$
linéaire	$A \rightarrow \alpha B \beta$
linéaire droite	$A \rightarrow \alpha \underline{B}$

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

Types de règles linéaires

constante	$A \rightarrow \alpha$
linéaire	$A \rightarrow \alpha B \beta$
linéaire droite	$A \rightarrow \alpha \underline{B}$
linéaire gauche	$A \rightarrow \underline{B} \beta$

D'autres types : règles linéaires

Soit la grammaire $G = (N, T, P, S)$

Soit $A, B \in N$ et $\alpha, \beta \in T^*$

Types de règles linéaires

constante	$A \rightarrow \alpha$
linéaire	$A \rightarrow \alpha B \beta$
linéaire droite	$A \rightarrow \alpha \underline{B}$
linéaire gauche	$A \rightarrow \underline{B} \beta$

Une règle est réduite

si $|\alpha| = 1$ et $|\beta| = 1$

$(A \rightarrow cBd,\ c, d \in T)$

Grammaires linéaires

Toute règle linéaire/linéaire droite/linéaire gauche

$\Longrightarrow$ grammaire linéaire/linéaire droite/linéaire gauche

Grammaires linéaires

Toute règle linéaire/linéaire droite/linéaire gauche

$\Longrightarrow$ grammaire linéaire/linéaire droite/linéaire gauche

Toute règle réduite $\Longrightarrow$ grammaire linéaire standard

Grammaires linéaires

Toute règle linéaire/linéaire droite/linéaire gauche

$\Longrightarrow$ grammaire linéaire/linéaire droite/linéaire gauche

Toute règle réduite $\Longrightarrow$ grammaire linéaire standard

Grammaires régulières = grammaires linéaires droites standard

- $A \to cB$, $A \to c$

Formes normales pour les grammaires algébriques

Soit une grammaire algébrique arbitraire $G = (N, T, P, S)$

- $A \to u$, $A \in N$, $u \in (N \cup T)^*$

Formes normales pour les grammaires algébriques

Soit une grammaire algébrique arbitraire $G = (N, T, P, S)$

- $A \to u$, $A \in N$, $u \in (N \cup T)^*$

Forme normale de Greibach

Pour toute G il existe une grammaire G' qui engendre le même langage et dont toutes les règles ont la forme suivante :

$$A \to c\alpha,\ A \in N,\ c \in T,\ \alpha \in (N \cup T)^*$$

Formes normales pour les grammaires algébriques

Soit une grammaire algébrique arbitraire $G = (N, T, P, S)$

- $A \to u$, $A \in N$, $u \in (N \cup T)^*$

Forme normale de Greibach

Pour toute G il existe une grammaire G' qui engendre le même langage et dont toutes les règles ont la forme suivante :

$$A \to c\alpha,\ A \in N,\ c \in T,\ \alpha \in (N \cup T)^*$$

Forme normale de Chomsky

Pour toute G il existe une grammaire G' qui engendre le même langage et dont toutes les règles ont l'une des formes suivantes :

$$A \to BC \qquad A \to d \qquad A \to \varepsilon$$

$$A, B, C \in N,\ d \in T$$

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

E

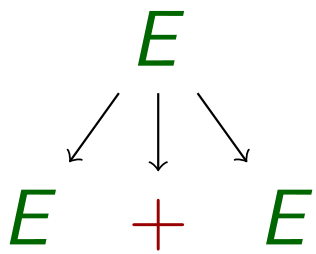
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



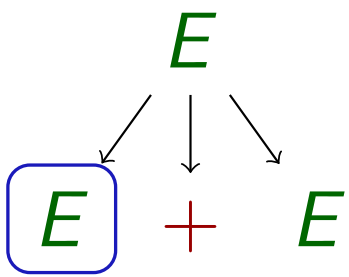
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



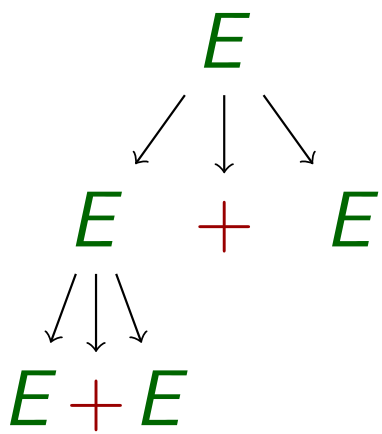
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



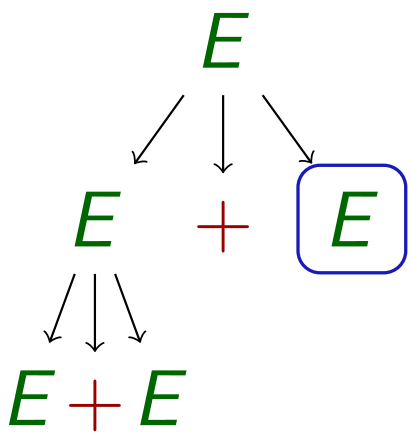
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



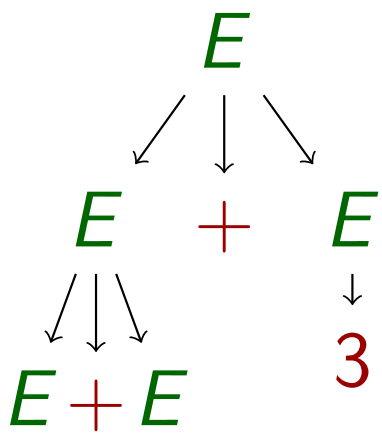
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



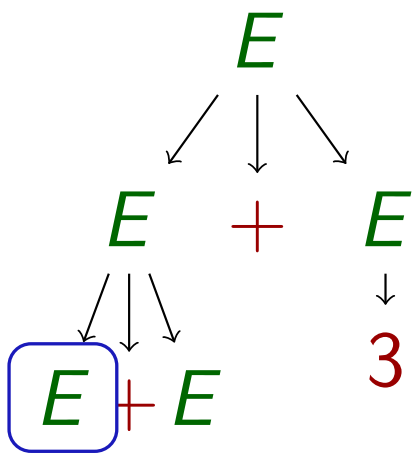
Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre



Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?

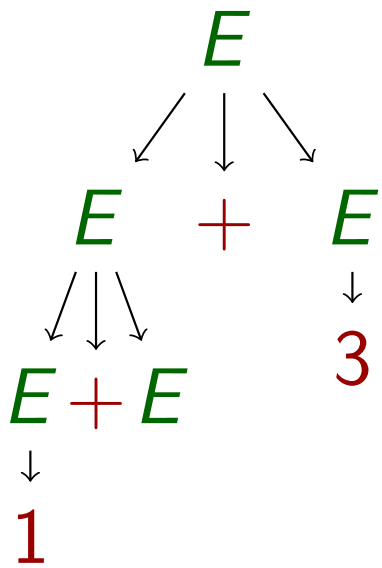
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



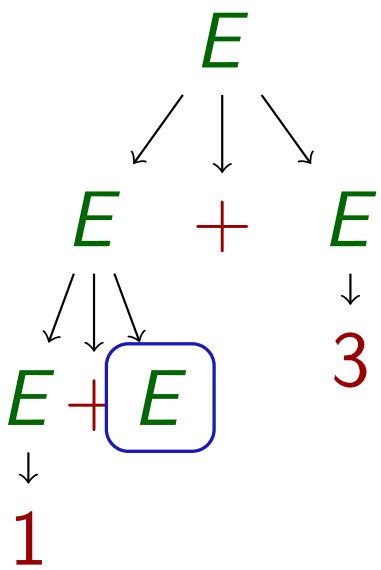
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



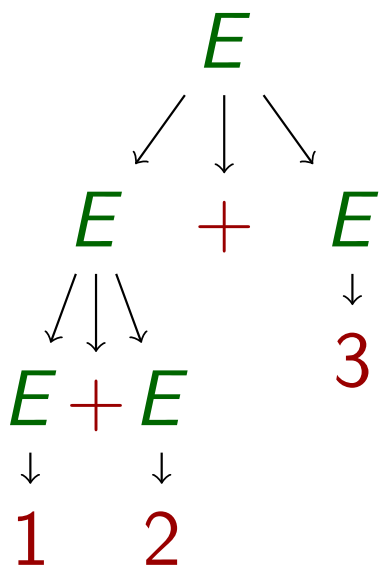
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



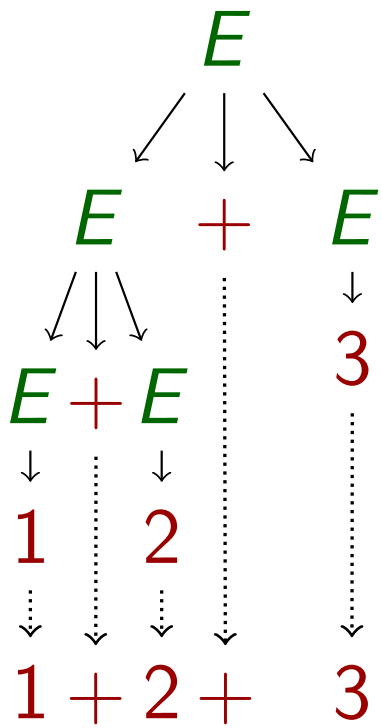
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



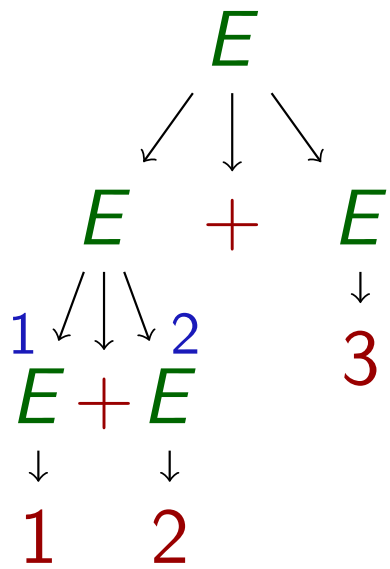
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



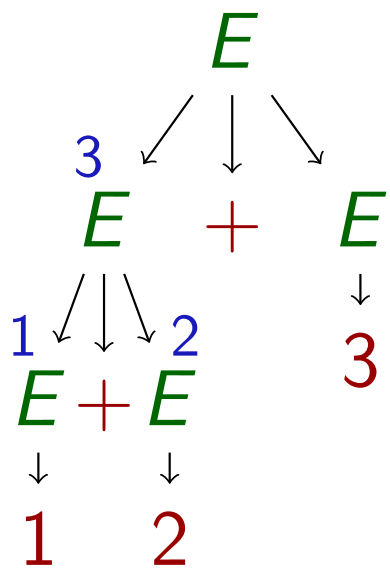
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



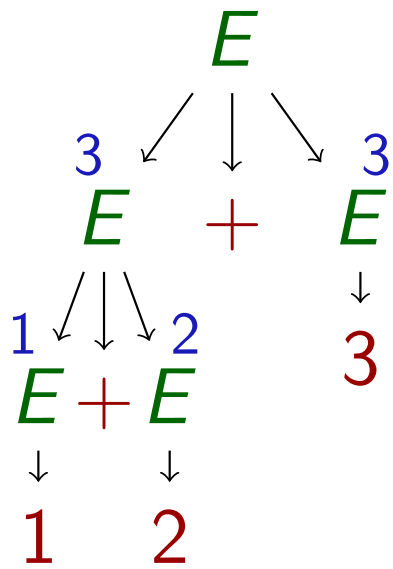
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



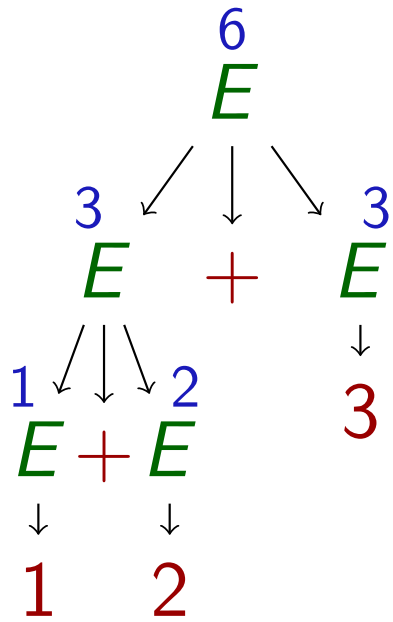
Arbres syntaxiques abstraits

Soit la grammaire G :

1. $E \rightarrow E+E$
2. $E \rightarrow E-E$
3. $E \rightarrow$ nombre

Exercices

1. Trouver N, T et S
2. Quel langage engendre G ?



Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to$ nombre

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3$$

$$1 \quad + \quad 2 \quad + \quad 3$$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$1 + 2 + \boxed{3}$

$1 \quad + \quad 2 \quad + \quad \boxed{3}$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E$$

$$\begin{array}{ccccc} & & & & E \\ & & & & \downarrow \\ 1 & + & 2 & + & 3 \end{array}$$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+\boxed{2}+E$$

$$1 \quad + \quad \boxed{2} \quad + \quad \begin{matrix} E \\ \downarrow \\ 3 \end{matrix}$$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E$$

$$\begin{array}{ccccc} & & E & & E \\ & & \downarrow & & \downarrow \\ 1 & + & 2 & + & 3 \end{array}$$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+\boxed{E+E}$$

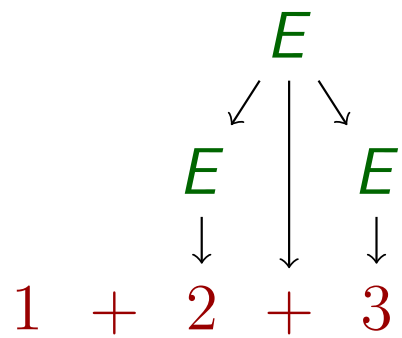
$$\begin{array}{ccccc} & & E & & E \\ & & \downarrow & & \downarrow \\ 1 & + & 2 & + & 3 \end{array}$$

Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E \overset{1}{\Rightarrow} 1+E$$

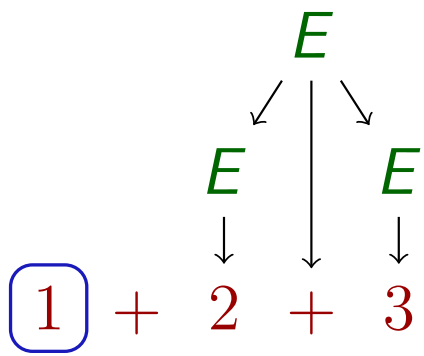


Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E \overset{1}{\Rightarrow} \boxed{1}+E$$

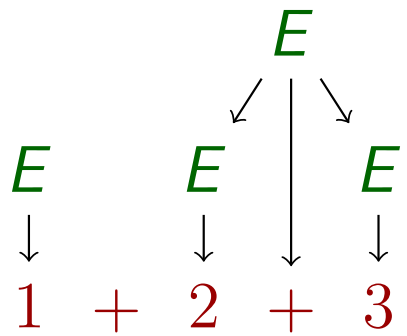


Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E \overset{1}{\Rightarrow} 1+E \overset{3}{\Rightarrow} E+E$$

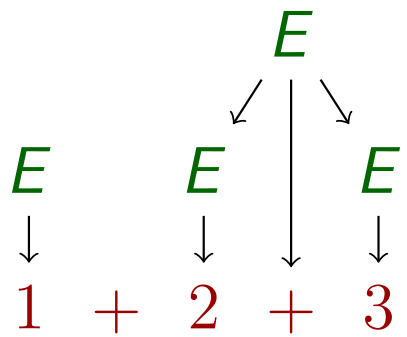


Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E \overset{1}{\Rightarrow} 1+E \overset{3}{\Rightarrow} \boxed{E+E}$$

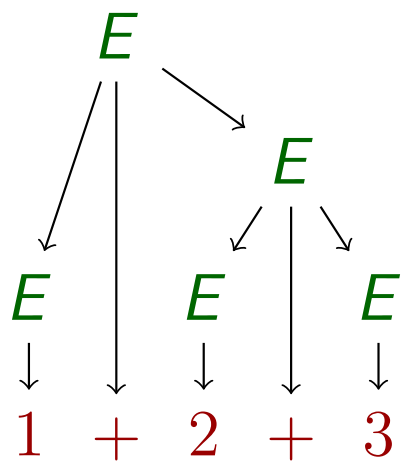


Analyse syntaxique automatique (vite fait)

On va de droite à gauche, en appliquant les règles à l'envers.

1. $E \to E+E$
2. $E \to E-E$
3. $E \to \text{nombre}$

$$1+2+3 \overset{3}{\Rightarrow} 1+2+E \overset{3}{\Rightarrow} 1+E+E \overset{1}{\Rightarrow} 1+E \overset{3}{\Rightarrow} E+E \overset{3}{\Rightarrow} E$$



Analyse syntaxique dans la vraie vie

Souvent les symboles consistent en plusieurs caractères

- `if`, `return`, `class`, etc.

Analyse syntaxique dans la vraie vie

Souvent les symboles consistent en plusieurs caractères

- `if`, `return`, `class`, etc.

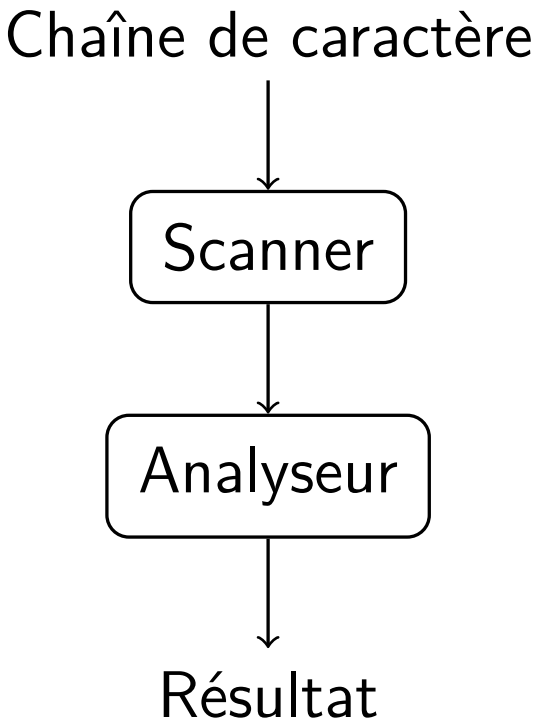
Un découpage de la chaîne est nécessaire avant que l'analyseur puisse commencer.

Analyse syntaxique dans la vraie vie

Souvent les symboles consistent en plusieurs caractères

- `if`, `return`, `class`, etc.

Un découpage de la chaîne est nécessaire avant que l'analyseur puisse commencer.

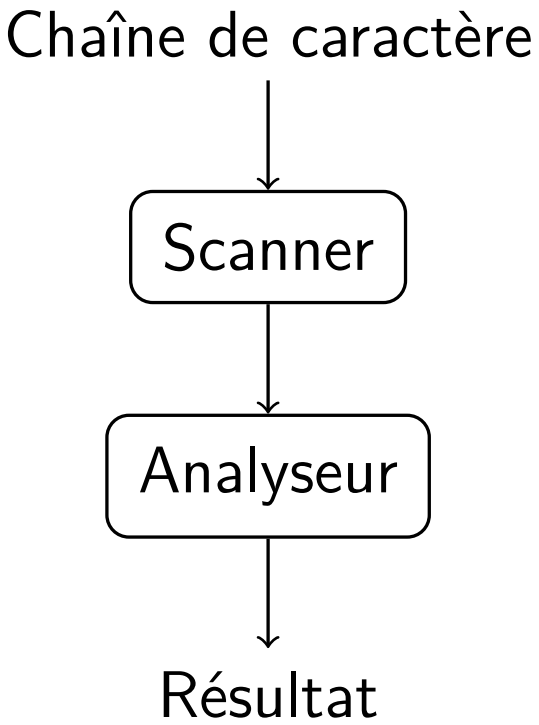


Analyse syntaxique dans la vraie vie

Souvent les symboles consistent en plusieurs caractères

- `if`, `return`, `class`, etc.

Un découpage de la chaîne est nécessaire avant que l'analyseur puisse commencer.



Nous allons jouer avec

- `JFlex` – générateur de scanners/lexers/bouchers
- `Jacc` – générateur d'analyseurs syntaxiques