Langages et Compilation

Analyse syntaxique descendante

Grammaires attribuées

Analyse syntaxique descendante

Grammaires attribuées

Analyse syntaxique

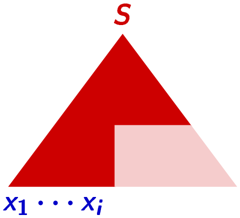
Étant donnée une grammaire :

- vérifier si un mot (un programme) est engendré par cette grammaire
- si oui, expliciter un arbre de dérivation de ce mot

Deux types d'analyse :

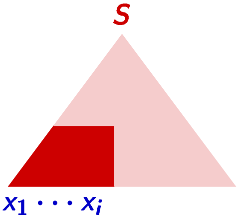
Analyse descendante

Construction de l'arbre de dérivation du haut vers le bas



Analyse ascendante

Construction de l'arbre de dérivation du bas vers le haut



Analyse descendante avec rebroussement

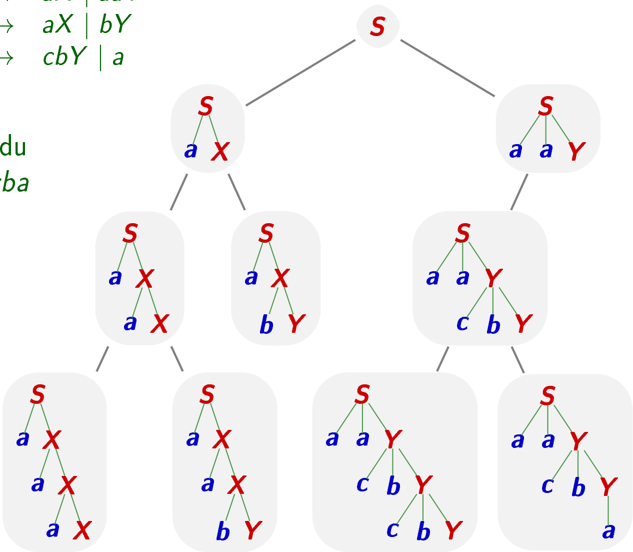
La méthode gros sabots : construction de l'arbre de dérivation avec backtracking

- On débute la construction de l'arbre à la racine.
- On dérive la variable la plus à gauche en choisissant les productions dans l'ordre où elles sont notées.
- Lorsque le fragment dérivé diffère du mot à analyser ou lorsque toutes les alternatives pour une variable ont été essayées, on retourne en arrière.

Analyse descendante avec rebroussement

$$\begin{cases} S & \rightarrow & aX \mid aaY \\ X & \rightarrow & aX \mid bY \\ Y & \rightarrow & cbY \mid a \end{cases}$$

analyse du mot *aacba*

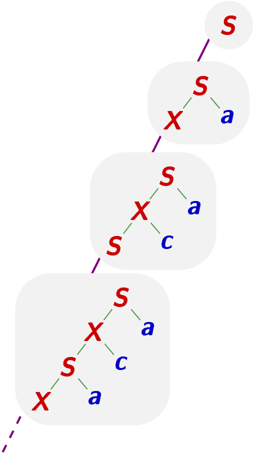


Analyse descendante et récursivité à gauche

Si la grammaire est **récursive à gauche**, i.e., si une de ses variables A est telle que $A \xrightarrow{*} A\alpha$ alors l'analyse descendante ne marche pas.

$$\begin{cases} S & \rightarrow & Xa \mid b \\ X & \rightarrow & Sc \mid Xd \mid \varepsilon \end{cases}$$

Analyse du mot b



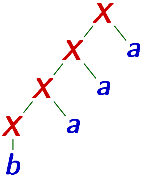
Mais on sait éliminer la récursivité gauche des grammaires.

Grammaire sans récursivité à gauche

Éliminer la récursivité à gauche immédiate

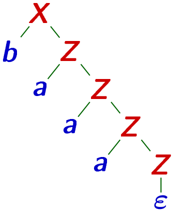
Une variable $\boldsymbol{A}$ d'une grammaire **G** est **récursive à gauche immédiate** si $\boldsymbol{A} \rightarrow \boldsymbol{A}\alpha$ est une production de **G**.

$X \rightarrow Xa \mid b \quad \rightsquigarrow \quad \mathcal{L}_G(X) : ba^*$



Introduire une nouvelle variable :

$$\begin{cases} X & \rightarrow & bZ & \rightsquigarrow & \mathcal{L}_G(X) : ba^* \\ Z & \rightarrow & aZ \mid \varepsilon & \rightsquigarrow & \mathcal{L}_G(Z) : a^* \end{cases}$$



Grammaire sans récursivité à gauche

Éliminer la récursivité à gauche immédiate

Cas général

On remplace

$$A \rightarrow A\beta_1 \mid \cdots \mid A\beta_j \mid \gamma_1 \mid \cdots \mid \gamma_k$$

où $\beta_1, \cdots, \beta_j \neq \varepsilon$ *et* $\gamma_1, \cdots, \gamma_k$ *ne débutant pas par* A

par

$$\begin{cases} A & \rightarrow & \gamma_1 Z \mid \cdots \mid \gamma_k Z \\ Z & \rightarrow & \beta_1 Z \mid \cdots \mid \beta_j Z \mid \varepsilon \end{cases}$$

$$A \rightsquigarrow (\gamma_1 + \cdots + \gamma_k)(\beta_1 + \cdots + \beta_j)^*$$

Grammaire sans récursivité à gauche

Éliminer la récursivité à gauche indirecte

Algorithme de suppression de toutes les récursivités à gauche

0. La grammaire est supposée sans cycle : elle n'admet pas de dérivation $A \xrightarrow{+} A$. on peut toujours se ramener à ce cas
1. Donner un ordre sur les variables

   $A_1, \cdots, A_n$

2. Modifier la grammaire de sorte que $j > i$ si $A_i \to A_j\alpha \in P$

```
pour i de 1 à n faire
    pour j de 1 à i − 1 faire
        remplacer chaque productions Ai → Ajγ
            par Ai → δ1γ | ··· | δkγ
            où Aj → δ1 | ··· | δk
    éliminer les récursivités à gauche immédiates de Ai
```

Grammaire sans récursivité à gauche

Éliminer la récursivité à gauche indirecte

$$\begin{cases} S & \rightarrow & Xa \mid b \\ X & \rightarrow & Sd \mid Xc \mid \varepsilon \end{cases}$$

On fixe un ordre sur les variables : $S < X$

Pour S, la première variable :

rien à faire, S n'a pas de récursivité à gauche immédiate

Pour X, la deuxième variable :

on doit éliminer la production $X \rightarrow Sd$

comme $S \rightarrow Xa \mid b$, on la remplace par $X \rightarrow Xad \mid bd$

on obtient $\begin{cases} S & \rightarrow & Xa \mid b \\ X & \rightarrow & Xad \mid Xc \mid bd \mid \varepsilon \end{cases}$

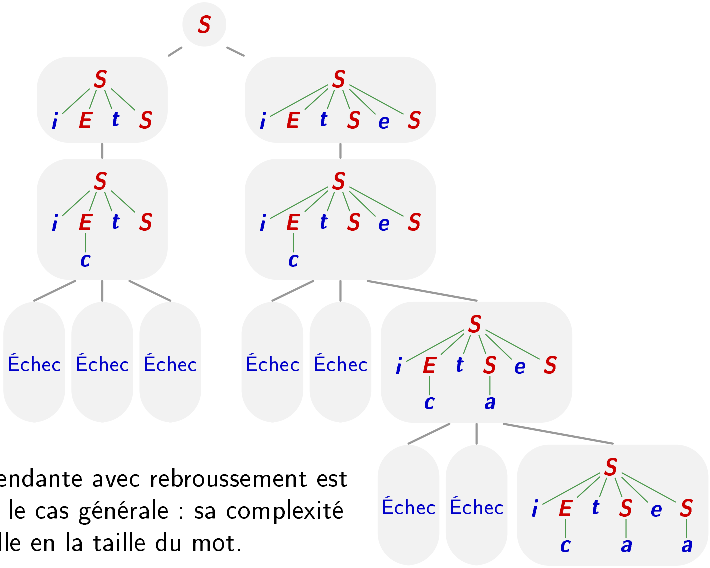
on élimine les récursivités à gauche immédiates de X

ce qui donne $\begin{cases} S & \rightarrow & Xa \mid b \\ X & \rightarrow & bdZ \mid Z \\ Z & \rightarrow & adZ \mid cZ \mid \varepsilon \end{cases}$

Coût de l'analyse descendante

$$\begin{cases} S & \rightarrow & i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a \\ E & \rightarrow & c \end{cases}$$

analyse
du mot
i c t a e a



L'analyse descendante avec rebroussement est inefficace dans le cas générale : sa complexité est exponentielle en la taille du mot.

L'analyse des mots de taille ***n*** nécessitent l'exploration d'un arbre de hauteur ***n***

Coût de l'analyse descendante

Côté pratique, une complexité exponentielle est rédhibitoire.

Comment rendre l'analyse plus efficace ?

- Éviter des retours en arrière inutiles en factorisant la grammaire à gauche.
- Et mieux : prédire si possible la bonne règle.

Factorisation à gauche des grammaires

La grammaire

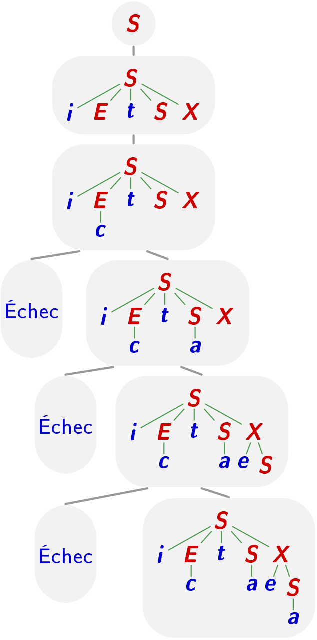
$$\begin{cases} S & \rightarrow & i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a \\ E & \rightarrow & c \end{cases}$$

n'est pas factorisée à gauche : deux dérivations de la variable S commencent par la même chaîne $iEtS$

On la remplace par une grammaire équivalente factorisée à gauche

$$\begin{cases} S & \rightarrow & i\,E\,t\,S\,X \mid a \\ X & \rightarrow & \varepsilon \mid e\,S \\ E & \rightarrow & c \end{cases}$$

En retardant ainsi les choix, on évite certains rebroussements lors de l'analyse du mot $i\;c\;t\;a\;e\;a$



Factorisation à gauche des grammaires

Cas général

- Trouver, pour chaque variable A, le plus long préfixe α commun à deux de ses alternatives.
- Si $\alpha \neq \varepsilon$, remplacer toutes les productions avec ce préfixe

$$A \rightarrow \alpha\beta_1 \mid \cdots \mid \alpha\beta_n$$

par

$$\begin{cases} A & \rightarrow \quad \alpha A' \\ A' & \rightarrow \quad \beta_1 \mid \cdots \mid \beta_n \end{cases}$$

- Recommencer tant que deux alternatives d'une même variable ont un préfixe commun.

Analyse descendante prédictive

Deviner à chaque étape la bonne alternative.

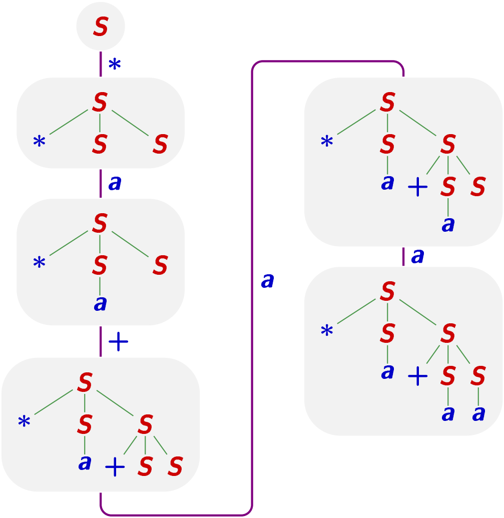
$S \;\rightarrow\; +SS \mid *SS \mid a$

analyse du mot

$*\,a + a\,a$

On détermine la dérivation à appliquer en prévisualisant une lettre

Ceci n'est possible que pour une certaine famille de grammaires, dites ***LL***, qu'on reverra.



Analyse syntaxique descendante

Grammaires attribuées

Grammaires attribuées

Les grammaires attribuées sont une extension des grammaires hors contexte qui permet de décorer les arbres syntaxiques d'informations sémantiques.

Ces informations sont ensuite exploitées lors de la phase d'analyse sémantique et à la génération de code.

Partant d'une grammaire, on associe

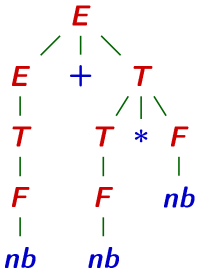
- un ou plusieurs **attributs** à chaque variable
- une ou plusieurs **règles sémantiques** à chaque production qui spécifient le calcul des valeurs des attributs

La valeur d'un attribut en un nœud de l'arbre est alors définie par la règle sémantique associée à la production utilisée en ce nœud.

Grammaires attribuées

Une grammaire des expressions arithmétiques

$$\begin{cases} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid nb \end{cases}$$



En construisant l'arbre de dérivation d'une expression, on voudrait également calculer sa valeur.

Grammaires attribuées

Une grammaire attribuée des expressions arithmétiques

On associe à chaque variable A un attribut $A.val$
On associe à chaque production une règle sémantique

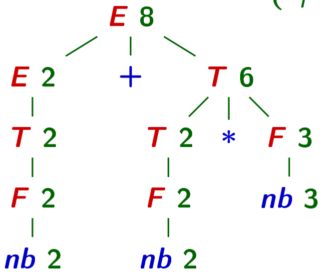
$$
\begin{array}{ll}
\text{Productions} & \text{Règles sémantiques} \\
\left\{\begin{array}{lcl}
E & \rightarrow & E_1 + T \\
E & \rightarrow & T \\
T & \rightarrow & T_1 * F \\
T & \rightarrow & F \\
F & \rightarrow & (E) \\
F & \rightarrow & nb
\end{array}\right.
&
\begin{array}{l}
\{E.val \leftarrow E_1.val + T.val\} \\
\{E.val \leftarrow T.val\} \\
\{T.val \leftarrow T_1.val * F.val\} \\
\{T.val \leftarrow F.val\} \\
\{F.val \leftarrow E.val\} \\
\{F.val \leftarrow getValeur(nb)\}
\end{array}
\end{array}
$$

NB. Dans chaque règle, on distingue les différentes occurrences d'une même variable en les numérotant.

Grammaires attribuées

L'arbre de dérivation décoré de l'expression $nb + nb * nb$

$$\begin{cases} E & \rightarrow & E_1 + T & \{E.val \leftarrow E_1.val + T.val\} \\ E & \rightarrow & T & \{E.val \leftarrow T.val\} \\ T & \rightarrow & T_1 * F & \{T.val \leftarrow T_1.val * F.val\} \\ T & \rightarrow & F & \{T.val \leftarrow F.val\} \\ F & \rightarrow & (E) & \{F.val \leftarrow E.val\} \\ F & \rightarrow & nb & \{F.val \leftarrow getValeur(nb)\} \end{cases}$$



Les valeurs des attributs aux feuilles sont données par l'analyseur lexical.

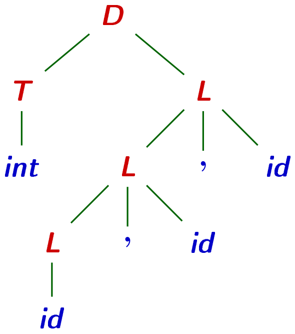
Le calcul s'effectue en remontant l'arbre.

Ici, l'attribut d'un nœud se calcule à l'aide de ceux de ses fils. L'attribut est dit **synthétisé**.

Grammaires attribuées

Une grammaire de déclaration de types de base

$$\left\{\begin{array}{lcl} D & \rightarrow & T\ L \\ T & \rightarrow & int \mid float \\ L & \rightarrow & L, id \mid id \end{array}\right.$$



Pour une déclaration

int id , id , id

on souhaite propager le type

int aux trois instances *id*.

Une grammaire attribuée de déclaration de types

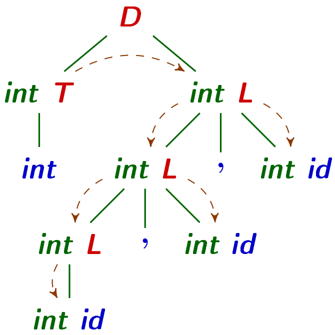
On introduit un attribut *type* et on ajoute des règles sémantiques.

$$\left\{\begin{array}{lcll} D & \rightarrow & T\ L & \{L.type \leftarrow T.type\} \\ T & \rightarrow & int & \{T.type \leftarrow int\} \\ T & \rightarrow & float & \{T.type \leftarrow float\} \\ L & \rightarrow & L_1, id & \{L_1.type \leftarrow L.type, \\ & & & \ id.type \leftarrow L.type\} \\ L & \rightarrow & id & \{id.type \leftarrow L.type\} \end{array}\right.$$

Grammaires attribuées

L'arbre de dérivation décoré de l'expression *int id*, *id*, *id*

$$\left\{\begin{array}{llll} D & \rightarrow & T\ L & \{L.type \leftarrow T.type\} \\ T & \rightarrow & int & \{T.type \leftarrow int\} \\ T & \rightarrow & float & \{T.type \leftarrow float\} \\ L & \rightarrow & L_1, id & \{L_1.type \leftarrow L.type, \\ & & & \ id.type \leftarrow L.type\} \\ L & \rightarrow & id & \{id.type \leftarrow L.type\} \end{array}\right.$$



Le calcul s'effectue en descendant l'arbre.

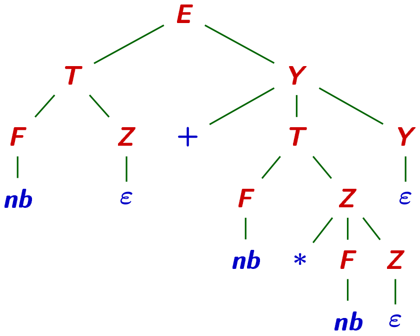
Ici, l'attribut d'un nœud se calcule à l'aide de son père ou de ses frères. L'attribut est dit **hérité**.

Grammaires attribuées

Une grammaire des expressions arithmétiques sans récursivité gauche

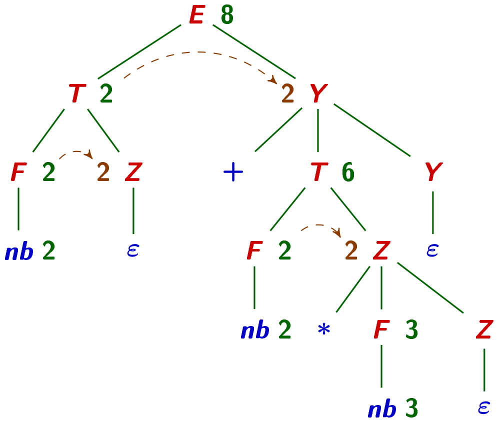
$$\begin{cases} E & \rightarrow & E+T \mid T \\ T & \rightarrow & T*F \mid F \\ F & \rightarrow & (E) \mid nb \end{cases} \xrightarrow{\text{suppression de la récursivité gauche}} \begin{cases} E & \rightarrow & TY \\ Y & \rightarrow & +TY \mid \varepsilon \\ T & \rightarrow & FZ \\ Z & \rightarrow & *FZ \mid \varepsilon \\ F & \rightarrow & (E) \mid nb \end{cases}$$

Comment décorer l'arbre pour calculer la valeur de l'expression ?



Grammaires attribuées

On va retrouver le même attribut synthétisé ***val*** que dans la version récursive mais on a besoin également d'introduire un attribut hérité ***st*** pour propager les calculs intermédiaires.



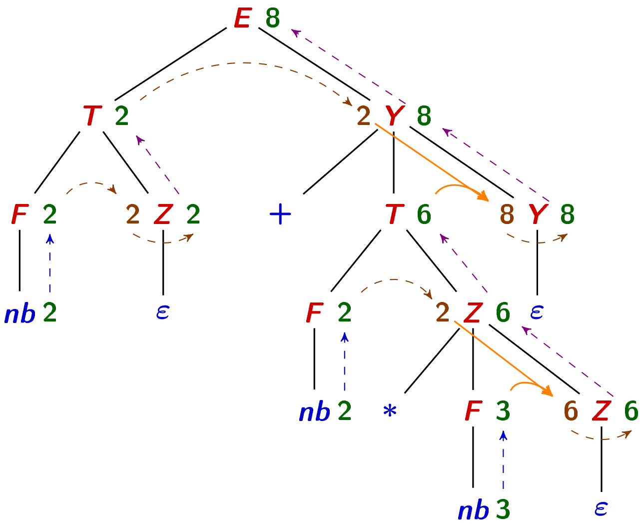
Grammaires attribuées

Une grammaire attribuée pour les expressions arithmétiques sans récursivité gauche

$$
\begin{cases}
E \rightarrow TY & \{Y.st \leftarrow T.val, E.val \leftarrow Y.val\} \\
Y \rightarrow +TY_1 & \{Y_1.st \leftarrow Y.st + T.val, Y.val \leftarrow Y_1.val\} \\
Y \rightarrow \varepsilon & \{Y.val \leftarrow Y.st\} \\
T \rightarrow FZ & \{Z.st \leftarrow F.val, T.val \leftarrow Z.val\} \\
Z \rightarrow *FZ_1 & \{Z_1.st \leftarrow Z.st * F.val, Z.val \leftarrow Z_1.val\} \\
Z \rightarrow \varepsilon & \{Z.val \leftarrow Z.st\} \\
F \rightarrow (E) & \{F.val \leftarrow E.val\} \\
F \rightarrow nb & \{F.val \leftarrow getValeur(nb)\}
\end{cases}
$$

Grammaires attribuées

L'arbre de dérivation décoré



Grammaires attribuées

Glossaire

Le calcul des attributs est calqué sur la structure syntaxique de l'arbre. On parle de **définition dirigée par la syntaxe**.

Les règles de calcul induisent des **contraintes de précédence** entre les instances d'attributs.

Une grammaire attribuée est **bien formée** s'il existe un schéma d'évaluation qui respecte les dépendances pour tout arbre de la grammaire.

Les trois grammaires attribuées vues en exemple sont bien formées.

Et autant que possible, on souhaite effectuer le calcul de ces attributs en parallèle de la construction de l'arbre de dérivation.

Grammaires attribuées

Glossaire

Parmi les grammaires bien formées, on distingue notamment les grammaires S-attribuées et les grammaires L-attribuées.

Une grammaire est **S-attribuée** si elle ne contient que des attributs synthétisés.

Le calcul des attributs est alors automatique avec une analyse ascendante.
La grammaire récursive gauche des expressions arithmétiques est S-attribuée.

Une grammaire est **L-attribuée** si on peut évaluer les attributs lors d'un parcours en profondeur préfixé de l'arbre de dérivation. Elle peut avoir des attributs hérités mais chacun ne dépend que de ceux du père ou des nœuds à gauche.
Une grammaire S-attribuée est L-attribuée.

Le calcul des attributs peut ainsi s'effectuer en parallèle d'une analyse descendante LL.
Les trois grammaires attribuées vues en exemple sont L-attribuées.

Construction de l'arbre de syntaxe abstraite d'une expression, version compacte de l'arbre de dérivation.

L'arbre de syntaxe abstraite de $\mathbf{2} + \mathbf{2} * \mathbf{3}$:

```
    +
   / \
  2   *
     / \
    2   3
```

Grammaires attribuées

$$\begin{cases} E & \rightarrow & E_1 + T & \{E.ast \leftarrow newArbre(+, E_1.ast, T.ast\} \\ E & \rightarrow & T & \{E.ast \leftarrow T.ast\} \\ T & \rightarrow & T_1 * F & \{T.ast \leftarrow newArbre(*, T_1.ast, F.ast)\} \\ T & \rightarrow & F & \{T.ast \leftarrow F.ast\} \\ F & \rightarrow & (E) & \{F.ast \leftarrow E.ast\} \\ F & \rightarrow & nb & \{F.ast \leftarrow newArbre(getValeur(nb), null, null)\} \end{cases}$$