Analyse syntaxique

Julien BERNARD

Université de Franche-Comté – UFR Sciences et Technique
Licence Informatique – 3è année

2016 – 2017

Première partie

Introduction – Compilation

Plan de ce cours

Plan

Votre enseignant
Qui suis-je ?

Qui suis-je ?

Julien BERNARD, Maître de Conférence (enseignant-chercheur)
`julien.bernard@univ-fcomte.fr`, Bureau 426C

Enseignement

- Responsable du semestre 1 (Starter) de la licence Informatique
- Cours : Bases de la programmation (L1), Publication web et scientifique (L1), Algorithmique (L2), Sécurité (L3), Théorie des Langages (L3), Analyse Syntaxique (L3)

Recherche

Optimisation dans les réseaux de capteurs

Plan

UE Analyse Syntaxique
Organisation

Équipe pédagogique

- Julien Bernard : CM, TD, TP (`julien.bernard@univ-fcomte.fr`)
- Guillaume Voiron : TP (`guillaume.voiron@univ-fcomte.fr`)

Volume

- Cours : 6 x 1h30
- TD : 6 x 1h30
- TP : 6 x 1h30

Évaluation

- 1 devoir surveillé
- un projet en TP

UE Analyse Syntaxique
Comment ça marche ?

Mode d'emploi

1. Prenez des notes ! Posez des questions !
2. Comprendre plutôt qu'apprendre
3. Le but de cette UE n'est pas d'avoir une note !

Niveau d'importance des transparents

	trivial	pour votre culture
⋆	intéressant	pour votre compréhension
⋆⋆	important	pour votre savoir
⋆⋆⋆	vital	pour votre survie

Note : les contrôles portent sur *tous* les transparents !

UE Analyse Syntaxique

Contenu pédagogique

Objectif

Comprendre les méthodes et outils pour l'analyse lexicale et l'analyse syntaxique en vue de la compilation et l'interprétation des langages de programmation

- Analyse syntaxique prédictive descendante
- Analyse syntaxique prédictive ascendante
- Transformation de grammaires algébriques pour l'analyse syntaxique
- Génération d'analyseurs syntaxiques (TP)

UE Analyse Syntaxique

Bibliographie

A. Aho, M. Lam, R. Sethi, J. Ullman.
Compilateurs : principes, techniques et outils.
2è édition, 2007, Pearson Education

D. Grun, C. Jacob.
Parsing techniques : A practical guide.
2è édition, 2010, Springer

Plan

Motivation

Pourquoi parler de compilateur ?

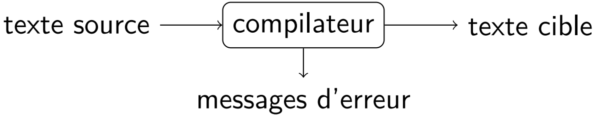
- Les concepts issus de la théorie des langages formels sont à la base des algorithmes pour le développement des compilateurs

$\rightarrow$ Le but du cours est de montrer en quoi cette théorie est une aide considérable pour la réalisation d'interpréteurs ou de compilateurs

Qu'est-ce qu'un compilateur ?

Définition (Compilateur)

Un **compilateur** est un programme qui lit un texte dans un premier langage, le *langage source*, et qui le traduit en un texte équivalent écrit dans un second langage, le *langage cible*.



Qu’est-ce qu’un compilateur ?

Exemples (Langages sources)

- C
- Java
- Scheme
- Ruby

Exemples (Langages cibles)

- Langage machine
- Bytecode
- C

Historique

Historique

- 1957, premier compilateur pour FORTRAN (25 KLOC)
- 1959, premier compilateur sur plusieurs architectures pour COBOL
- 1962, premier compilateur auto-hébergé pour LISP

→ Amélioration de la productivité
  - Techniques systématiques dérivées de la théorie des langages
  - Outils de développement

Domaines d'application

Domaines d'application

- Langages de programmation
  - Interpréteurs
  - Compilateurs
- Langages de description
  - XML, HTML, CSS
  - Fichiers de configuration
- Langages de requêtes
  - SQL

Plan

Structure d'un compilateur

Structure d'un compilateur

1. Partie avant *(front end)*
   1. Analyse lexicale
   2. Analyse syntaxique
   3. Analyse sémantique

$\rightarrow$ Représentation intermédiaire

2. Partie arrière *(back end)*
   1. Optimisations
   2. Génération de code

Analyse lexicale

Analyse lexicale

- Lexique : ensemble des mots d'une langue
- Décompose une chaîne de caractères en **lexèmes** *(token)*
  - Mots-clefs, identifiants, litéraux, opérateurs, . . .
- Utilise des *langages réguliers*
- Outils de génération : Lex, Flex

Analyse syntaxique

Analyse syntaxique

- Syntaxe : étude de l'arrangement des mots et des phrases
- Transforme une suite de lexèmes en une hiérarchie de **syntagmes**
    - Arbre de syntaxe abstrait
- Utilise des *langages algébriques*
- Outils de génération : Yacc, Bison, JavaCC, ANTLR, Menhir

Analyse sémantique

Analyse sémantique

- Sémantique : étude du sens, de la signification des mots
- Ajoute des informations à l'arbre de syntaxe abstrait
  - Résolution des noms (table des symboles)
  - Vérification des types

→ Voir cours de *Compilation* en M1

Terminologie

Terminologie

→ Terminologie différente en théorie des langages et en compilation !

Théorie des langages	Compilation
Lettre	Lexème
Mot	Phrase

Plan

1. Introduction
   - À propos de votre enseignant
   - À propos du cours d'Analyse Syntaxique
2. Compilation
   - Qu'est-ce qu'un compilateur ?
   - Structure d'un compilateur
3. Analyse syntaxique
   - Langages et analyseurs syntaxiques

Grammaire

Grammaire

Une grammaire G peut être utilisée pour trois tâches :

- **Reconnaissance** : pour un mot w donné, décider si $w \in \mathcal{L}(G)$
- **Analyse** *(Parsing)* : pour un mot w donné, construire tous les arbres de dérivation possibles pour le mot w
- **Génération** : générer tous les mots w tels que $w \in \mathcal{L}(G)$

Rappel : Dérivation la plus à gauche ⋆⋆

Définition (Dérivation la plus à gauche)

Soit $G = (N, T, S, R)$ une grammaire, et $w \in \mathcal{L}(G)$, la dérivation $S \rightarrow^* w$ est la **dérivation la plus à gauche** si, à chaque étape de la dérivation, c'est le symbole non-terminal le plus à gauche qui est dérivé.

Rappel : Arbre de dérivation ⋆⋆

Définition (Arbre de dérivation)

Soit $G = (N, T, S, R)$ une grammaire et $w \in \mathcal{L}(G)$.
L'**arbre de dérivation** du mot w est un arbre où :

- la racine est S
- les feuilles sont étiquetées par des éléments terminaux de T
- les nœuds sont étiquetés par des éléments non-terminaux de N
- si un nœud est étiqueté Y et ses fils sont étiquetés $Z_1, \ldots, Z_k$ dans cet ordre, alors il existe une règle $Y \to Z_1 \ldots Z_k$ dans R
- la lecture des feuilles de gauche à droite donne le mot w

Rappel : Grammaire ambiguë ⋆⋆

Définition (Grammaire ambiguë)

Une grammaire G est **ambiguë** s'il existe un mot de $\mathcal{L}(G)$ qui a au moins deux arbres de dérivation, c'est-à-dire deux dérivations la plus à gauche.

Proposition (Grammaire ambiguë)

Le problème de savoir si une grammaire G est ambiguë est indécidable, c'est-à-dire il n'existe pas d'algorithme qui permet de répondre à la question : est-ce que G est ambiguë ?

Grammaire ambiguë

Les méthodes d'analyse syntaxique permettent de répondre à cette question pour certaines classes de grammaires !

Grammaire et langages de programmation

Grammaire et langages de programmation

Un langage de programmation doit avoir une grammaire non-ambiguë

$\rightarrow$ S'il existe plusieurs arbres de dérivation pour un même programme, il existe plusieurs façons d'analyser le programme avec des résultats qui peuvent être très différents

Exemple (Grammaire ambiguë)

L'analyse et l'évaluation avec une grammaire ambiguë de l'expression «$10 \times 10 + 10$» peut donner 110 ou 200

Grammaire et langages de programmation

Grammaire et langages de programmation

- Les langages de programmation sont des langages algébriques
- La reconnaissance et l'analyse sont possibles !
- Difficulté principale : le *non-déterminisme* des langages algébriques
  - Éviter de perdre son temps dans des impasses
  - Éviter de refaire plusieurs fois les mêmes calculs

→ Classes de grammaires algébriques avec de bonnes propriétés

Analyseur syntaxique

Analyseur syntaxique

Un analyseur syntaxique va permettre d'analyser une grammaire G :

- Il prend en entrée un mot w de G
- Il retourne l'unique arbre de dérivation du mot w

$\rightarrow$ Si G n'est pas dans une bonne classe de grammaire, on échoue

Type d'analyseurs syntaxiques

Type d'analyseurs syntaxiques

Il existe deux grands types d'analyseurs syntaxiques :

- **Analyseurs descendants** : construction de l'arbre syntaxique à partir de la racine, et donc de l'axiome de la grammaire
  $\rightarrow$ Analyse LL
- **Analyseurs ascendants** : construction de l'arbre syntaxique à partir des feuilles, et donc des symboles terminaux
  $\rightarrow$ Analyse LR

Deuxième partie

Analyse descendante

Plan de ce cours

Plan

Analyse descendante simple

Principe

- On lit le mot lettre après lettre
- On essaie de prédire quelle est la dérivation la plus à gauche
  - issue de l'axiome
  - correspondant à ce qu'on a déjà lu

→ Mécanisme d'essai-erreur

Analyse descendante simple

Analyse descendante simple

On considère :

- une grammaire $G = (N, T, S, R)$
- une procédure `read(a)` qui :
  - renvoie vrai et lit la lettre a si c'est la lettre courante
  - renvoie faux si la lettre a n'est pas la lettre courante
- pour chaque non-terminal $A \in T$, une procédure `A()` pour l'ensemble de règles $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_k$ de R, qui va tester successivement les règles en appelant :
  - les procédures correspondantes à chaque non-terminal
  - la procédure `read` pour un terminal

$\rightarrow$ Si l'appel à `S()` renvoie vrai alors le mot est reconnu.

Analyse descendante simple

Exemple (Analyse descendante simple)

On considère la grammaire $G = (\{S, A\}, \{a, b, c, d\}, S, R)$ avec R :

- $S \to cAd$
- $A \to ab \mid a$

```
S() {
  if read('c') and A() and read('d')
    return true
  return false
}

A() {
  if read('a') and read('b')
    return true
  if read('a')
    return true
  return false
}
```

Analyse descendante simple

Exemple (Analyse descendante simple)

On considère la grammaire $G = (\{S, A\}, \{a, b, c, d\}, S, R)$ avec R :

- $S \to cAd$
- $A \to ab \mid a$

On essaie de reconnaître le mot *cad* :

- On appelle `S()`
  - On choisit la première règle $S \to cAd$
  - On lit c
  - On appelle `A()`
    - On choisit la première règle $A \to ab$
    - On lit a
    - On lit b mais on échoue
    - On choisit la seconde règle $A \to a$
    - On lit a
  - On lit d : le mot a été lu en entier

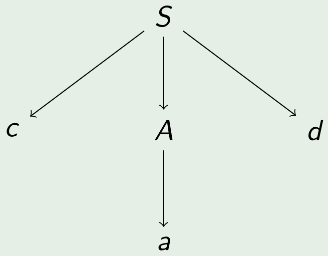
Analyse descendante simple

Exemple (Analyse descendante simple)

On considère la grammaire $G = (\{S, A\}, \{a, b, c, d\}, S, R)$ avec R :

- $S \rightarrow cAd$
- $A \rightarrow ab \mid a$

L'arbre de dérivation obtenu pour le mot *cad* est le suivant :



Analyse descendante simple

Remarques

- On peut avoir des retours arrières *(backtracking)* si on s'est trompé
  - $\rightarrow$ La complexité est exponentielle !
- Les procédures ainsi construites peuvent être mutuellement récursives
  - $\rightarrow$ Comment s'assurer qu'on ne boucle pas ?
- On fait parfois le même travail plusieurs fois (par exemple `read(a)`)
  - $\rightarrow$ Peut-on factoriser ces parties communes ?

Plan

Problèmes liés à la grammaire

Problèmes liés à la grammaire

L'analyse descendante se heurte à deux problèmes :

- l'analyse peut boucler s'il y a une récursivité à gauche, immédiate ou non, entraînant une récursivité infinie dans la procédure
  - $\rightarrow$ Élimination des récursivités à gauche
- l'analyse peut répéter un même traitement pour des membres droits d'un non-terminal qui commencent de la même manière
  - $\rightarrow$ Factorisation

Grammaire récursive à gauche ⋆⋆

Définition (Grammaire récursive à gauche)

Une grammaire algébrique est **récursive à gauche** si elle contient un symbole $A \in N$ tel qu'il existe une dérivation $A \rightarrow^* A\alpha, \alpha \in V^*$

Algorithme d'élimination des récursivités à gauche ⋆⋆

Algorithme d'élimination des récursivités à gauche immédiates

Soit une grammaire avec des dérivations à gauche *immédiates* pour le symbole non-terminal A :

- $A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

Alors, on transforme la règle précédente en ajoutant le non-terminal A' :

- $A \to \beta_1 A' \mid \ldots \mid \beta_m A'$
- $A' \to \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$

Algorithme d'élimination des récursivités à gauche ⋆⋆

Algorithme d'élimination des récursivités à gauche

1. Numéroter les non-terminaux : $N = \{A_1, \ldots, A_n\}$
2. Pour tous les non-terminaux A_i
   1. Pour toutes les règles $A_i \rightarrow A_j\alpha_i$ avec $j < i$
      1. Pour chaque règle $A_j \rightarrow \beta_j$, ajouter la règle $A_i \rightarrow \beta_j\alpha_i$
      2. Enlever la règle $A_i \rightarrow A_j\alpha_i$
   2. Éliminer les récursions à gauche immédiates pour A_i

Factorisation

Factorisation

La **factorisation** consiste à :

1. regrouper les parties initiales communes des membres droits des règles d'un non-terminal
2. modifier les règles en tenant compte de ces regroupements en introduisant de nouveaux non-terminaux

$\rightarrow$ permet de repousser les choix entre alternatives le plus tard possible

Factorisation

Algorithme de factorisation

1. Pour chaque non-terminal A :
   1. Trouver le plus long préfixe α commun à deux de ses règles ou plus
   2. Si $\alpha \neq \varepsilon$ (il y a un préfixe commun non-trivial), remplacer toutes les règles
      - $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_m$

      où les γ_i ne commencent pas par α, par :
      - $A \to \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m$
      - $A' \to \beta_1 \mid \beta_2 \mid \ldots \beta_n$
   3. Recommencer jusqu'à ce que le non-terminal n'ait plus de règle avec un préfixe commun

Factorisation

Exemple (Factorisation)

On considère la grammaire G_1 suivante :

- $S \rightarrow$ if B then $S \mid$ if B then S else $S \mid a$
- $B \rightarrow$ true $\mid$ false

On peut factoriser des membres droits de S, on obtient la grammaire G_2 :

- $S \rightarrow$ if B then $SS' \mid a$
- $S' \rightarrow$ else $S \mid \varepsilon$
- $B \rightarrow$ true $\mid$ false

La grammaire G_1 est ambiguë, est-ce le cas de la grammaire G_2 ?

Factorisation

Exemple (Factorisation)

On considère la grammaire suivante :

- $S \to abS \mid abA \mid aB$
- $A \to c$
- $B \to d$

On fait une première itération avec $\alpha = ab$:

- $S \to abS' \mid aB$
- $S' \to S \mid A$
- $A \to c$
- $B \to d$

On fait une seconde itération avec $\alpha = a$:

- $S \to aS''$
- $S'' \to bS' \mid B$
- $S' \to S \mid A$
- $A \to c$
- $B \to d$

Plan

4 Analyse descendante simple
- Principe
- Transformation de la grammaire
- Limite

5 Analyse LL(1)
- Principe
- Construction de la table d'analyse
- Grammaire LL(1)

Limite de l'analyse descendante simple

Limite de l'analyse descendante simple

- L'analyse descendante simple n'est pas efficace à cause des retours arrières et de la complexité exponentielle
- Or, il est souvent possible de connaître la règle à appliquer
  - lorsqu'on analyse un non-terminal donné
  - lorsqu'on connaît la première lettre d'un mot issu de ce non-terminal

→ On peut construire une analyse *prédictive* sans retour arrière

Plan

Analyse LL

Définition (Analyse LL)

Une **analyse LL(k)** :

- analyse un mot d'entrée de gauche à droite *(**L**eft to right)*
- en construit une dérivation à gauche *(**L**eftmost derivation)*
- prend en compte les k symboles suivants de la chaîne d'entrée

Généralement, $k = 1$.

Remarques

- Une analyse LL ne fait jamais de retour arrière, si elle ne parvient pas à continuer, c'est que le mot n'appartient pas à la grammaire
- Un analyse LL est une analyse descendante particulière, il est donc nécessaire que la grammaire soit sans récursivité à gauche et factorisée

Analyse LL

Analyseur LL

Un analyseur LL est composé de :

- un *mot à analyser* suivi d'un marqueur de fin $\#$, ainsi que la lettre courante
- une *pile* contenant des terminaux et des non-terminaux en attente d'analyse, initialisée à S où S est l'axiome
- une *table d'analyse* T qui indique la règle à utiliser (s'il y en a une) en fonction de la lettre courante et du sommet de pile

Analyse LL

Fonctionnement d'un analyseur LL

On examine le sommet de pile et la lettre courante c

1. Si la pile est vide, on s'arrête ; si $c = \#$, alors le mot est reconnu
2. Si le sommet de pile est un terminal a et que $a = c$ alors, on dépile a et on consomme c ; sinon on échoue
3. Si le sommet de pile est un non-terminal X, alors on remplace X par le mot $\beta = T(X, c)$, en empilant les lettres de β en partant de la fin ; sinon on échoue

Analyse LL

Exemple (Analyse LL)

On considère la grammaire suivante :

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid \text{id}$

La table d'analyse pour cette grammaire est :

	id	$+$	$\times$	(	)	#
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to \times FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to \text{id}$			$F \to (E)$		

Analyse LL

Exemple (Analyse du mot $\mathrm{id} + \mathrm{id} \times \mathrm{id}$)

Pile	Mot	Règle
$E \vdash$	$\mathrm{id} + \mathrm{id} \times \mathrm{id}\#$	$E \to TE'$
$TE' \vdash$	$\mathrm{id} + \mathrm{id} \times \mathrm{id}\#$	$T \to FT'$
$FT'E' \vdash$	$\mathrm{id} + \mathrm{id} \times \mathrm{id}\#$	$F \to \mathrm{id}$
$\mathrm{id}T'E' \vdash$	$\mathrm{id} + \mathrm{id} \times \mathrm{id}\#$	
$T'E' \vdash$	$+\mathrm{id} \times \mathrm{id}\#$	$T' \to \varepsilon$
$E' \vdash$	$+\mathrm{id} \times \mathrm{id}\#$	$E' \to +TE'$
$+TE' \vdash$	$+\mathrm{id} \times \mathrm{id}\#$	
$TE' \vdash$	$\mathrm{id} \times \mathrm{id}\#$	$T \to FT'$
$FT'E' \vdash$	$\mathrm{id} \times \mathrm{id}\#$	$F \to \mathrm{id}$
$\mathrm{id}T'E' \vdash$	$\mathrm{id} \times \mathrm{id}\#$	
$T'E' \vdash$	$\times\mathrm{id}\#$	$T' \to \times FT'$
$\times FT'E' \vdash$	$\times\mathrm{id}\#$	
$FT'E' \vdash$	$\mathrm{id}\#$	$F \to \mathrm{id}$
$\mathrm{id}T'E' \vdash$	$\mathrm{id}\#$	
$T'E' \vdash$	$\#$	$T' \to \varepsilon$
$E' \vdash$	$\#$	$E' \to \varepsilon$
$\vdash$	$\#$	

Analyse LL

En pratique

- L'analyse LL présentée ici n'est pas récursive mais itérative
  - la pile est explicite
  - la table est explicite
- Il est possible de créer des procédures récursives
  - la pile est implicite : $\sim$ pile d'appels
  - la table est implicite : suite de conditions

Analyse LL

Exemple (Retour sur l'exemple introductif)

On considère la grammaire $G = (\{S, A\}, \{a, b, c, d\}, S, R)$ avec R :

- $S \to cAd$
- $A \to ab \mid a$

On la factorise :

- $S \to cAd$
- $A \to aA'$
- $A' \to b \mid \varepsilon$

On crée la table d'analyse :

	a	b	c	d	$\#$
S			$S \to cAd$		
A	$A \to aA'$				
A'		$A' \to b$		$A' \to \varepsilon$	

Analyse LL

Exemple (Retour sur l'exemple introductif)

Avec la table d'analyse, on peut créer des procédures sans retour arrière.

```
S() {
  if (current = 'c')
    read('c'); A(); read('d'); return
  error()
}

A() {
  if (current = 'a')
    read('a'); Ap(); return
  error()
}

Ap() {
  if (current = 'b')
    read('b'); return
  if (current = 'd')
    /* epsilon */ return
  error()
}
```

Que se passe-t-il à la lecture du mot *cad* ?

Plan

Table d'analyse

Table d'analyse

- Intuitivement, si on a une règle $A \to \alpha \mid \beta$, la table d'analyse se construit en sachant par quelle lettre commencent les mots dont dérivent α et β, ce qui permettra de choisir entre la règle $A \to \alpha$ et la règle $A \to \beta$.
- Pour cela, on a besoin de trois fonctions : $\textsc{Null}$, $\textsc{First}$, $\textsc{Follow}$.

Définition ($\textsc{Null}$)

Soit $\alpha \in (N \cup T)^+$, alors $\textsc{Null}(\alpha)$ vaut `true` si α est annulable, c'est-à-dire $\alpha \to^* \varepsilon$, et `false` sinon.

Table d'analyse

Définition (First)

Soit $\alpha \in (N \cup T)^+$, alors $\textsc{First}(\alpha)$ est l'ensemble des terminaux par lesquels la chaîne α peut commencer.

$$\textsc{First}(\alpha) = \{a \in T \mid \exists \beta \in (N \cup T)^*, \alpha \to^* a\beta\}$$

Définition (Follow)

Soit $\alpha \in (N \cup T)^+$, alors $\textsc{Follow}(\alpha)$ est l'ensemble des terminaux (ou le marqueur de fin $\#$) qui peuvent apparaître à droite de α au cours d'une dérivation. On a toujours $\# \in \textsc{Follow}(S)$.

$$\textsc{Follow}(\alpha) = \{a \in \textsc{First}(\gamma) \mid \exists \beta, \gamma \in (N \cup T)^*, S \to^* \beta\alpha\gamma\}$$

$\textsc{Null}$

Algorithme de calcul de $\textsc{Null}(\alpha)$

1. Si α contient un terminal, alors $\textsc{Null}(\alpha)$ vaut `false`
2. Si $\alpha = A_1 \dots A_n$, $\textsc{Null}(\alpha)$ vaut `true` si $\textsc{Null}(A_i)$ vaut `true` pour tout i
3. Si $\alpha = A$, avec $A \in N$, ...
   $\rightarrow$ Voir cours de Théorie des Langages, chapitre *Grammaires algébriques*

NULL

Exemple (NULL)

Soit la grammaire :

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid \text{id}$

NULL :

E	E'	T	T'	F
false	true	false	true	false

First

Équations pour First

Soient $X \in N, a \in T, \beta \in (N \cup T)^*$, First est défini par une série d'équations mutuellement récursives :

$$\textsc{First}(X) = \bigcup_{X \to \beta} \textsc{First}(\beta)$$

ainsi que :

$$\begin{aligned}
\textsc{First}(\varepsilon) &= \varnothing \\
\textsc{First}(a\beta) &= \{a\} \\
\textsc{First}(X\beta) &= \textsc{First}(X), \text{ si } \neg\textsc{Null}(X) \\
\textsc{First}(X\beta) &= \textsc{First}(X) \cup \textsc{First}(\beta), \text{ si } \textsc{Null}(X)
\end{aligned}$$

First

Algorithme de construction de First

1. Pour tous les non-terminaux A, poser $\text{First}_0(A) = \varnothing$
2. Calculer $\text{First}_{n+1}(A)$ à l'aide des équations et des $\text{First}_n(A_i)$
3. Arrêter lorsque $\text{First}_{n+1}(A_i) = \text{First}_n(A_i)$ pour tout i

→ Ces ensembles forment First

First

Exemple (First)

Soit la grammaire :

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid \text{id}$

$\textsc{First}(E) = \textsc{First}(T)$

$\textsc{First}(E') = \{+\}$

$\textsc{First}(T) = \textsc{First}(F)$

$\textsc{First}(T') = \{\times\}$

$\textsc{First}(F) = \{(, \text{id}\}$

Null :

E	E'	T	T'	F
false	true	false	true	false

First :

E	E'	T	T'	F
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\varnothing$	$\{+\}$	$\varnothing$	$\{\times\}$	$\{(, \text{id}\}$
$\varnothing$	$\{+\}$	$\{(, \text{id}\}$	$\{\times\}$	$\{(, \text{id}\}$
$\{(, \text{id}\}$	$\{+\}$	$\{(, \text{id}\}$	$\{\times\}$	$\{(, \text{id}\}$
$\{(, \text{id}\}$	$\{+\}$	$\{(, \text{id}\}$	$\{\times\}$	$\{(, \text{id}\}$

Follow

Équations pour Follow

Soient $X, Y \in N, \alpha, \beta \in (N \cup T)^*$, Follow est défini par une série d'équations mutuellement récursives :

$$\textsc{Follow}(X) = \left(\bigcup_{Y \to \alpha X \beta} \textsc{First}(\beta) \right) \cup \left(\bigcup_{Y \to \alpha X \beta \text{ et } \textsc{Null}(\beta)} \textsc{Follow}(Y) \right)$$

Follow

Algorithme de construction de Follow

- Pour tous les non-terminaux A, poser $\textsc{Follow}_0(A) = \varnothing$ sauf pour l'axiome S où $\textsc{Follow}(S) = \{\#\}$
- Calculer $\textsc{Follow}_{n+1}(A)$ à l'aide des équations des $\textsc{Follow}_n(A_i)$
- Arrêter lorsque $\textsc{Follow}_{n+1}(A_i) = \textsc{Follow}_n(A_i)$ pour tout i

$\rightarrow$ Ces ensembles forment Follow

Follow

Exemple (Follow)

Soit la grammaire :

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid \mathrm{id}$

$\textsc{Follow}(E) = \{\#,)\}$

$\textsc{Follow}(E') = \textsc{Follow}(E)$

$\textsc{Follow}(T) = \textsc{First}(E') \cup \textsc{Follow}(E') \cup \textsc{Follow}(E)$

$\textsc{Follow}(T') = \textsc{Follow}(T)$

$\textsc{Follow}(F) = \textsc{First}(T') \cup \textsc{Follow}(T') \cup \textsc{Follow}(T)$

Null :

E	E'	T	T'	F
`false`	`true`	`false`	`true`	`false`

First :

E	E'	T	T'	F
$\{(, \mathrm{id}\}$	$\{+\}$	$\{(, \mathrm{id}\}$	$\{\times\}$	$\{(, \mathrm{id}\}$

Follow :

E	E'	T	T'	F
$\{\#\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\{\#,)\}$	$\{\#\}$	$\{+, \#\}$	$\varnothing$	$\{\times\}$
$\{\#,)\}$	$\{\#,)\}$	$\{+, \#,)\}$	$\{+, \#\}$	$\{\times, +, \#\}$
$\{\#,)\}$	$\{\#,)\}$	$\{+, \#,)\}$	$\{+, \#,)\}$	$\{\times, +, \#,)\}$
$\{\#,)\}$	$\{\#,)\}$	$\{+, \#,)\}$	$\{+, \#,)\}$	$\{\times, +, \#,)\}$

Table d'analyse

Algorithme de construction de la table d'analyse

1. Pour toutes les règles $X \to \alpha$
   1. Pour tout $a \in \textsc{First}(\alpha)$, ajouter $X \to \alpha$ à $T(X, a)$
   2. Si $\textsc{Null}(\alpha)$, pour tout $a \in \textsc{Follow}(X)$, ajouter $X \to \alpha$ à $T(X, a)$

Table d'analyse

Exemple (Table d'analyse)

- $E \to TE'$
- $E' \to +TE' \mid \varepsilon$
- $T \to FT'$
- $T' \to \times FT' \mid \varepsilon$
- $F \to (E) \mid \mathrm{id}$

FIRST :

E	E'	T	T'	F
$\{(, \mathrm{id}\}$	$\{+\}$	$\{(, \mathrm{id}\}$	$\{\times\}$	$\{(, \mathrm{id}\}$

FOLLOW :

E	E'	T	T'	F
$\{\#,)\}$	$\{\#,)\}$	$\{+, \#,)\}$	$\{+, \#,)\}$	$\{\times, +, \#,)\}$

	id	$+$	$\times$	(	)	#
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to \times FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to \mathrm{id}$			$F \to (E)$		

Plan

Grammaire LL(1)

Définition (Grammaire LL(1))

Une **grammaire LL(1)** est une grammaire telle que la table d'analyse construite par l'analyse LL(1) est déterministe

Définition (Grammaire LL(1))

Une **grammaire LL(1)** est une grammaire telle qu'il est possible lors de l'analyse syntaxique de déterminer quelle règle de production doit être appliquée au non-terminal le plus à gauche dans une dérivation gauche par la connaissance de ce non-terminal et de la première lettre de la chaîne d'entrée restant à lire, si elle existe

Grammaire LL(1)

Proposition (Grammaire LL(1))

Si une grammaire est LL(1), alors :

- *elle est non-ambiguë*
- *l'algorithme d'analyse LL(1) donne une façon de construire l'unique arbre de dérivation d'un mot*

Remarques

- Si une grammaire n'est pas LL(1), il est parfois possible de faire une analyse déterministe en lisant $k > 1$ lettres, on parle alors de grammaires LL(2), LL(3), . . . , LL(k)
- Certaines grammaires algébriques ne peuvent pas être analysées de manière descendante quel que soit le nombre de lettres lues à l'avance

Troisième partie

Analyse LR

Plan de ce cours

Plan

6 Analyse LR(0)

- Principe de l'analyse ascendante
- Analyseur LR
- Tables d'analyse LR(0)
- Tables d'analyse SLR(1)

Analyse ascendante

Principe

L'analyse LR est une analyse ascendante *(bottom-up parsing)*, c'est-à-dire qu'on part du mot à analyser et on essaie de reconstruire l'arbre de dérivation en remontant jusqu'à l'axiome de la grammaire.

Rappel

Dans un arbre de dérivation, les lettres du mot sont les feuilles.

Analyse ascendante

Fonctionnement intuitif

On dispose de :

- un mot w à reconnaître, suivi d'un marqueur de fin $\#$
- une pile contenant des terminaux et des non-terminaux en attente d'analyse, initialement vide

À chaque étape, deux actions sont possibles :

- un **décalage** *(shift)* :
  - on lit un terminal de la chaîne d'entrée
  - on l'empile
- une **réduction** *(reduce)* :
  - on reconnaît en sommet de pile le membre droit α d'une règle $X \to \alpha$
  - on remplace α par X en sommet de pile

Analyse ascendante

Fonctionnement intuitif

- Le mot w est accepté si on termine la lecture du mot avec la pile réduite à l'axiome S
- Si l'analyse n'est pas possible, alors le mot n'appartient pas au langage engendré par la grammaire

$\rightarrow$ Mécanisme d'essai-erreur : il peut y avoir plusieurs actions possibles (décalage ou réduction)

Analyse ascendante

Exemple (Analyse du mot id $+$ id $\times$ id)

On considère la grammaire suivante :

- $E \to E + T \mid T$
- $T \to T \times F \mid F$
- $F \to (E) \mid \text{id}$

Pile	Mot	Action
$\dashv$	$\text{id} + \text{id} \times \text{id}\#$	décalage
$\dashv \text{id}$	$+\text{id} \times \text{id}\#$	réduction $F \to \text{id}$
$\dashv F$	$+\text{id} \times \text{id}\#$	réduction $T \to F$
$\dashv T$	$+\text{id} \times \text{id}\#$	réduction $E \to T$
$\dashv E$	$+\text{id} \times \text{id}\#$	décalage
$\dashv E+$	$\text{id} \times \text{id}\#$	décalage
$\dashv E + \text{id}$	$\times \text{id}\#$	réduction $F \to \text{id}$
$\dashv E + F$	$\times \text{id}\#$	réduction $T \to F$
$\dashv E + T$	$\times \text{id}\#$	décalage
$\dashv E + T\times$	$\text{id}\#$	décalage
$\dashv E + T \times \text{id}$	$\#$	réduction $F \to \text{id}$
$\dashv E + T \times F$	$\#$	réduction $T \to T \times F$
$\dashv E + T$	$\#$	réduction $E \to E + T$
$\dashv E$	$\#$	accepté

Analyse ascendante

Questions

- Comment décider entre un décalage et une réduction ?
- Comment rendre déterministe ce choix et ne pas retourner en arrière ?

Plan

Analyse LR

Définition (Analyse LR)

Une **analyse LR(k)** :

- analyse un mot d'entrée de gauche à droite *(**L**eft to right)*
- en construit une dérivation à droite *(**R**ightmost derivation)*
- prend en compte les k symboles suivants de la chaîne d'entrée

Généralement, $k = 1$.

Remarques

- La grammaire peut contenir des récursivités à gauche, et même des récursivités à droite tant que la grammaire n'est pas ambiguë !
- La plupart des grammaires de langage de programmation peut s'analyser de cette façon

Analyse LR

Analyseur LR

Un analyseur LR est composé de :

- un *mot à analyser* suivi d'un marqueur de fin #, ainsi que la lettre courante
- une *pile* de la forme $\dashv s_0 X_1 s_1 X_2 s_2 \ldots X_n s_n$ où :
  - X_i est un *symbole de la grammaire* (terminal ou non-terminal)
  - s_i est un *état* qui résume l'information contenue en dessous dans la pile
- deux *tables d'analyse* :
  - `Action` qui détermine l'action à effectuer en fonction de l'état courant et d'un terminal
  - `Goto` qui indique l'état résultat en fonction d'un état et d'un non-terminal

Analyse LR

Table d'analyse `Action`

La table d'analyse `Action` peut contenir :

- «`décaler` s_i» où s_i est l'état suivant
  - parfois abrégé «Si»
- «`réduire` $A \to \alpha$»
  - parfois abrégé «Rj» où j est le numéro de la règle $A \to \alpha$
- «`accepter`»
  - parfois abrégé «Acc»
- «`refuser`», pour indiquer une erreur
  - abrégé sans aucune indication

Analyse LR

Fonctionnement d'un analyseur LR

On examine l'état s_n du sommet de la pile, la lettre courante a et $\texttt{Action}(s_n, a)$:

- «$\texttt{décaler}$ s» :
  - on lit a
  - on empile a et s : $\dashv \ldots X_n s_n a s$
- «$\texttt{réduire}$ $A \to \alpha$» avec $\alpha = \alpha_1 \ldots \alpha_p$
  $\to$ la pile est $\dashv s_0 X_1 s_1 \ldots X_{n-p} s_{n-p} | \alpha_1 s_{n-p+1} \ldots \alpha_p s_n$
  - on dépile α
  - on empile A et s avec $s = \texttt{Goto}(s_{n-p}, A)$: $\dashv \ldots X_{n-p} s_{n-p} X s$
- «$\texttt{accepter}$» :
  - on arrête, le mot est reconnu
- «$\texttt{refuser}$» :
  - on arrête, le mot n'appartient pas au langage

Analyse LR

Exemple (Analyse LR : tables d'analyse Action et Goto)

1. $E \to E + T$
2. $E \to T$
3. $T \to T \times F$
4. $T \to F$
5. $F \to (E)$
6. $F \to \text{id}$

État	Action						Goto		
	id	$+$	$\times$	(	)	#	E	T	F
0	S5			S4			1	2	3
1		S6				Acc			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Analyse LR

Exemple (Analyse LR du mot id + id × id)

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T \times F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \text{id}$

Pile	Mot	Action
$\dashv 0$	$\text{id} + \text{id} \times \text{id}\#$	S5
$\dashv 0\ \text{id}\ 5$	$+\text{id} \times \text{id}\#$	R6
$\dashv 0\ F\ 3$	$+\text{id} \times \text{id}\#$	R4
$\dashv 0\ T\ 2$	$+\text{id} \times \text{id}\#$	R2
$\dashv 0\ E\ 1$	$+\text{id} \times \text{id}\#$	S6
$\dashv 0\ E\ 1 + 6$	$\text{id} \times \text{id}\#$	S5
$\dashv 0\ E\ 1 + 6\ \text{id}\ 5$	$\times\text{id}\#$	R6
$\dashv 0\ E\ 1 + 6\ F\ 3$	$\times\text{id}\#$	R4
$\dashv 0\ E\ 1 + 6\ T\ 9$	$\times\text{id}\#$	S7
$\dashv 0\ E\ 1 + 6\ T\ 9 \times 7$	$\text{id}\#$	S5
$\dashv 0\ E\ 1 + 6\ T\ 9 \times 7\ \text{id}\ 5$	$\#$	R6
$\dashv 0\ E\ 1 + 6\ T\ 9 \times 7\ F\ 10$	$\#$	R3
$\dashv 0\ E\ 1 + 6\ T\ 9$	$\#$	R1
$\dashv 0\ E\ 1$	$\#$	Acc

Analyseur LR

Remarque

Tous les analyseurs LR fonctionnent de cette manière, la seule différence est dans la construction des tables d'analyse

Plan

Tables d'analyse LR(0)

Tables d'analyse LR(0)

Pour construire les tables d'analyse LR(0), on a besoin des outils suivants :

- une grammaire augmentée
- des items LR(0)
- de la fermeture d'un ensemble d'items LR(0)
- de la transition entre deux ensembles d'items LR(0)

Exemple (Grammaire d'exemple)

Soit $G = (\{E, B\}, \{0, 1\}, E, R)$ avec R :

- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

Grammaire augmentée

Définition (Grammaire augmentée)

Soit la grammaire $G = (N, T, S, R)$, la **grammaire augmentée** G' est la grammaire G à laquelle on a ajouté un nouvel axiome S' qui mène vers l'ancien axiome.

$$G' = (N \cup \{S'\}, T, S', R \cup \{S' \to S\})$$

Intérêt

- L'intérêt d'une grammaire augmentée est qu'on sait que la dernière règle qui servira pour une réduction sera $S' \to S$
- S' ne se trouve jamais en partie droite d'une règle

Grammaire augmentée

Exemple (Grammaire augmentée)

$G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

Item LR(0)

Définition (Item LR(0))

Un **item LR(0)** (ou item) d'une grammaire G est une règle de G avec un point ($\bullet$) repérant une position de sa partie droite :

$$A \to \alpha \bullet \beta$$

Item LR(0)

Le point fait une séparation entre ce qui a déjà été lu et ce qui reste à lire. L'item $A \to \alpha \bullet \beta$ signifie qu'on a déjà lu une chaîne dérivée de α et qu'on attend maintenant une chaîne dérivée de β pour pouvoir faire la réduction de $\alpha\beta$ vers A.

Item LR(0)

Exemples (Item LR(0))

La règle $E \to E \times B$ fournit quatre items LR(0) :

- $E \to \bullet E \times B$
- $E \to E \bullet \times B$
- $E \to E \times \bullet B$
- $E \to E \times B \bullet$

Cas particulier

Une règle $A \to \varepsilon$ fournit uniquement l'item $A \to \bullet$

Fermeture d'un ensemble d'items

Définition (Fermeture d'un ensemble d'items)

La **fermeture d'un ensemble d'items** I, notée $Closure(I)$, est un ensemble d'items qui se calcule par l'algorithme suivant :

1. Initialement, ajouter chaque item de I dans $Closure(I)$
2. Si $A \rightarrow \alpha \bullet B\beta$ est dans $Closure(I)$ et $B \rightarrow \gamma$ est une règle, alors ajouter $B \rightarrow \bullet\gamma$ à $Closure(I)$. Cette étape est appliquée jusqu'à stabilisation de l'ensemble.

Fermeture d'un ensemble d'items

Intuitivement, si $A \rightarrow \alpha \bullet B\beta$ est dans $Closure(I)$, on s'attend à dériver une chaîne depuis $B\beta$. Si $B \rightarrow \gamma$ est une règle, la chaîne à dériver pourrait être dérivée depuis γ, d'où l'ajout de $B \rightarrow \bullet\gamma$ à $Closure(I)$

Fermeture d'un ensemble d'items

Exemple (Fermeture d'un ensemble d'items)

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

La fermeture de $E' \to \bullet E$ se calcule de la manière suivante :

1. Étape 1 :
   - $E' \to \bullet E$
2. Étape 2 :
   1. Itération 1 :
      - $E \to \bullet E \times B$; $E \to \bullet E + B$; $E \to \bullet B$
   2. Itération 2 :
      - $B \to \bullet 0$; $B \to \bullet 1$

Automate des items LR(0)

Automate des items LR(0)

Les ensembles d'items LR(0) fermés vont former un automate dont les transitions sont étiquetées par des symboles terminaux ou non-terminaux de la grammaire : $\mathcal{A} = (\{s_0, \dots, s_n\}, s_0, \delta, \varnothing)$ avec :

- $s_0 = Closure(\{S' \to \bullet S\})$
- δ est définie de la manière suivante :
  - soit s_i un ensemble d'items et $X \in (N \cup T)$, $\delta(s_i, X)$ est défini comme la fermeture de l'ensemble des items $A \to \alpha X \bullet \beta$ où $A \to \alpha \bullet X\beta$ se trouve dans s_i

$$\delta(s_i, X) = \{Closure(\{A \to \alpha X \bullet \beta\}) \mid A \to \alpha \bullet X\beta \in s_i\}$$

Automate des items LR(0)

Algorithme de calcul de l'automate des items LR(0)

1. Ajouter $s_0 = Closure(\{S' \rightarrow \bullet S\})$ aux états non-traités
2. Tant qu'il existe un état non-traité s_i
    1. Calculer $\mathcal{X}$ l'ensemble des symboles qui suivent directement le point $\bullet$
    2. Pour tous les symboles $X \in \mathcal{X} \subset (N \cup T)$
        1. Prendre l'ensemble I de tous les items où un point précède le symbole X
        2. Pour tous les items dans I, déplacer le point à droite de X
        3. Compléter par fermeture ce nouvel ensemble des items
        4. Ajouter le nouvel ensemble s_j obtenu aux états non-traités
        5. Ajouter (s_i, X, s_j) à δ

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet E \times B, E \to \bullet E + B, E \to \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet E \times B, E \to \bullet E + B, E \to \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 0) = \{B \to 0\bullet\}$
- $s_2 = \delta(s_0, 1) = \{B \to 1\bullet\}$
- $s_3 = \delta(s_0, E) = \{E' \to E\bullet, E \to E\bullet \times B, E \to E\bullet + B\}$
- $s_4 = \delta(s_0, B) = \{E \to B\bullet\}$

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet E \times B, E \to \bullet E + B, E \to \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 0) = \{B \to 0 \bullet\}$
- $s_2 = \delta(s_0, 1) = \{B \to 1 \bullet\}$
- $s_3 = \delta(s_0, E) = \{E' \to E \bullet, E \to E \bullet \times B, E \to E \bullet + B\}$
- $s_4 = \delta(s_0, B) = \{E \to B \bullet\}$
- $s_5 = \delta(s_3, \times) = \{E \to E \times \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_6 = \delta(s_3, +) = \{E \to E + \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet E \times B, E \to \bullet E + B, E \to \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 0) = \{B \to 0 \bullet\}$
- $s_2 = \delta(s_0, 1) = \{B \to 1 \bullet\}$
- $s_3 = \delta(s_0, E) = \{E' \to E \bullet, E \to E \bullet \times B, E \to E \bullet + B\}$
- $s_4 = \delta(s_0, B) = \{E \to B \bullet\}$
- $s_5 = \delta(s_3, \times) = \{E \to E \times \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_6 = \delta(s_3, +) = \{E \to E + \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_7 = \delta(s_5, B) = \{E \to E \times B \bullet\}$ et aussi $\delta(s_5, 0) = s_1$ et $\delta(s_5, 1) = s_2$

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet E \times B, E \to \bullet E + B, E \to \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 0) = \{B \to 0 \bullet\}$
- $s_2 = \delta(s_0, 1) = \{B \to 1 \bullet\}$
- $s_3 = \delta(s_0, E) = \{E' \to E \bullet, E \to E \bullet \times B, E \to E \bullet + B\}$
- $s_4 = \delta(s_0, B) = \{E \to B \bullet\}$
- $s_5 = \delta(s_3, \times) = \{E \to E \times \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_6 = \delta(s_3, +) = \{E \to E + \bullet B\} \cup \{B \to \bullet 0, B \to \bullet 1\}$
- $s_7 = \delta(s_5, B) = \{E \to E \times B \bullet\}$ et aussi $\delta(s_5, 0) = s_1$ et $\delta(s_5, 1) = s_2$
- $s_8 = \delta(s_6, B) = \{E \to E + B \bullet\}$ et aussi $\delta(s_6, 0) = s_1$ et $\delta(s_6, 1) = s_2$

Automate des items LR(0)

Exemple (Automate des items LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

Table de transition :

États	$\times$	$+$	0	1	E	B
0			1	2	3	4
1						
2						
3	5	6				
4						
5			1	2		7
6			1	2		8
7						
8						

Tables d'analyse LR(0)

Tables d'analyse LR(0)

Pour la table `Action` :

1. Si $A \rightarrow \alpha \bullet a\beta \in s_i$ et $\delta(s_i, a) = s_j$
   - $\texttt{Action}(s_i, a)$ vaut «`décaler` s_j»
2. Si $A \rightarrow \alpha \bullet \in s_i$ et $A \neq S'$
   - $\texttt{Action}(s_i, x)$ vaut «`réduire` $A \rightarrow \alpha$» pour tout $x \in T \cup \{\#\}$
3. Si $S' \rightarrow S \bullet \in s_i$
   - $\texttt{Action}(s_i, \#)$ vaut «`accepter`»

Pour la table `Goto` :

1. Si $\delta(s_i, A) = s_j$
   - $\texttt{Goto}(s_i, A) = s_j$

Tables d'analyse LR(0)

Exemple (Tables d'analyse LR(0))

Soit $G' = (\{E', E, B\}, \{0, 1\}, E', R')$ avec R' :

- $E' \to E$
- $E \to E \times B \mid E + B \mid B$
- $B \to 0 \mid 1$

Tables d'analyse :

État	Action					Goto	
	$\times$	$+$	0	1	#	E	B
0			S1	S2		3	4
1	R4	R4	R4	R4	R4		
2	R5	R5	R5	R5	R5		
3	S5	S6			Acc		
4	R3	R3	R3	R3	R3		
5			S1	S2			7
6			S1	S2			8
7	R1	R1	R1	R1	R1		
8	R2	R2	R2	R2	R2		

Grammaire LR(0)

Définition (Grammaire LR(0))

Une **grammaire LR(0)** est une grammaire telle que les tables d'analyse construites par l'analyse LR(0) sont déterministes

Remarque

Les tables d'analyse LR(0) sont souvent non-déterministes, des conflits apparaissent !

Conflits

Conflits

La table `Action` peut contenir deux sortes de conflits :

- un conflit décalage/réduction *(shift/reduce)*, si dans un état s on peut effectuer un décalage ou une réduction
- un conflit réduction/réduction *(reduce/reduce)*, si dans un état s deux réductions différentes sont possibles

Remarque

Il ne peut pas y avoir de conflit décalage/décalage

Conflits

Exemple (Conflit décalage/réduction)

Soit la grammaire G :

- $E' \to E$
- $E \to 1E \mid 1$

On a alors :

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet 1E, E \to \bullet 1\}$
- $s_1 = \delta(s_0, 1) = \{E \to 1 \bullet E, E \to 1 \bullet\} \cup \{E \to \bullet 1E, E \to \bullet 1\}$
- $s_2 = \delta(s_0, E) = \{E' \to E \bullet\}$
- $s_3 = \delta(s_1, E) = \{E \to 1E \bullet\}$ et aussi $\delta(s_1, 1) = s_1$

Un conflit intervient pour l'état s_1 puisqu'on a :

- un décalage avec le terminal 1
- une réduction avec la règle $E \to 1$

Conflits

Exemple (Conflit réduction/réduction)

Soit la grammaire G :

- $E' \to E$
- $E \to A1 \mid B2$
- $A \to 1$
- $B \to 1$

On a alors :

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet A1, E \to \bullet B2\} \cup \{A \to \bullet 1, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 1) = \{A \to 1\bullet, B \to 1\bullet\}$
- ...

Un conflit intervient pour l'état s_1 puisqu'on a :

- une réduction avec la règle $A \to 1$
- une réduction avec la règle $B \to 1$

Plan

6 Analyse LR(0)

- Principe de l'analyse ascendante
- Analyseur LR
- Tables d'analyse LR(0)
- Tables d'analyse SLR(1)

Motivation

Motivation

- Les tables d'analyse LR(0) ne font aucune anticipation
  - $\rightarrow$ les actions de réduction occupent des lignes entières de la table `Action`
  - $\rightarrow$ des conflits apparaissent
- Idée : regarder les terminaux suivant le non-terminal réduit pour restreindre les actions de réduction et donc éviter les conflits

Tables d'analyse SLR(1)

Tables d'analyse SLR(1)

Les tables d'analyse SLR(1) *(Simple LR(1))* se construisent de la même manière que les tables d'analyse LR(0) à une exception :

2. Si $A \to \alpha \bullet \in s_i$ et $A \neq S'$
   - $\mathtt{Action}(s_i, x)$ vaut «$\mathtt{réduire}$ $A \to \alpha$» pour tout $x \in \textsc{Follow}(A)$

L'idée est de regarder la lettre suivante x du mot d'entrée et de faire la réduction $A \to \alpha$ uniquement si x peut suivre A.

Grammaire SLR(1)

Définition (Grammaire SLR(1))

Une **grammaire SLR(1)** est une grammaire telle que les tables d'analyse construites par l'analyse SLR(1) sont déterministes

Proposition (Grammaire SLR(1))

Si une grammaire est SLR(1), alors :

- *elle est non-ambiguë*
- *l'algorithme d'analyse SLR(1) donne une façon de construire l'unique arbre de dérivation d'un mot*

Grammaire SLR(1)

Exemple (Grammaire SLR(1))

Soit la grammaire G :

- $E' \to E$
- $E \to 1E \mid 1$

On a alors :

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet 1E, E \to \bullet 1\}$
- $s_1 = \delta(s_0, 1) = \{E \to 1 \bullet E, E \to 1 \bullet\} \cup \{E \to \bullet 1E, E \to \bullet 1\}$
- $s_2 = \delta(s_0, E) = \{E' \to E \bullet\}$
- $s_3 = \delta(s_1, E) = \{E \to 1E \bullet\}$ et aussi $\delta(s_1, 1) = s_1$

Le conflit décalage/réduction pour l'état s_1 n'existe plus puisque :

$$1 \notin \textsc{Follow}(E) = \{\#\}$$

Grammaire SLR(1)

Exemple (Grammaire SLR(1))

Soit la grammaire G :

- $E' \to E$
- $E \to A1 \mid B2$
- $A \to 1$
- $B \to 1$

On a alors :

- $s_0 = \{E' \to \bullet E\} \cup \{E \to \bullet A1, E \to \bullet B2\} \cup \{A \to \bullet 1, B \to \bullet 1\}$
- $s_1 = \delta(s_0, 1) = \{A \to 1 \bullet, B \to 1 \bullet\}$
- ...

Le conflit réduction/réduction pour l'état s_1 n'existe plus puisque :

$$\textsc{Follow}(A) = \{1\} \text{ et } \textsc{Follow}(B) = \{2\}$$

Quatrième partie

Analyse LR

Plan de ce cours

7 Analyse LR(1)
- Tables d'analyse LR(1)
- Tables d'analyse LALR(1)
- Conclusion

Plan

Motivation

Motivation

- Dans les tables d'analyse SLR(1), on indique une réduction par $A \rightarrow \alpha$ avec l'item $A \rightarrow \alpha \bullet$ pour tous les $a \in \textsc{Follow}(A)$
- Or, on ne sait pas si a peut suivre βA dans la pile !

Exemple (Grammaire d'exemple)

Soit la grammaire $G = (\{S', S, C\}, \{a, b\}, S', R)$ avec R :

- $S' \rightarrow S$
- $S \rightarrow CC$
- $C \rightarrow aC|b$

Item LR(1)

Définition (Item LR(1))

Un **item LR(1)** d'une grammaire G est un couple formé d'un item LR(0) et d'un symbole de prévision (non-terminal ou $\#$) :

$$(A \rightarrow \alpha \bullet \beta, a)$$

Item LR(1)

L'item $(A \rightarrow \alpha \bullet \beta, a)$ signifie qu'on a déjà lu une chaîne dérivée de α et qu'on attend maintenant une chaîne dérivée de β pour pouvoir faire la réduction de $\alpha\beta$ vers A et le caractère suivant doit être a.

Exemple (Item LR(1))

$(S' \rightarrow \bullet S, \#)$ est un item LR(1).

Fermeture d'un ensemble d'items

Définition (Fermeture d'un ensemble d'items)

La **fermeture d'un ensemble d'items** I, notée $Closure(I)$, est un ensemble d'items qui se calcule par l'algorithme suivant :

1. Initialement, ajouter chaque item de I dans $Closure(I)$
2. Si...
   - $(A \to \alpha \bullet B\beta, a)$ est dans $Closure(I)$
   - $B \to \gamma$ est une règle
   - b est dans $\textsc{First}(\beta a)$

   ...alors ajouter $(B \to \bullet\gamma, b)$ à $Closure(I)$.
   Cette étape est appliquée jusqu'à stabilisation de l'ensemble.

Fermeture d'un ensemble d'items

Exemple (Fermeture d'un ensemble d'items)

Soit la grammaire $G = (\{S', S, C\}, \{a, b\}, S', R)$ avec R :

- $S' \to S$
- $S \to CC$
- $C \to aC|b$

La fermeture de $(S' \to \bullet S, \#)$ est :

1. Étape 1 :
   - $(S' \to \bullet S, \#)$
2. Étape 2 :
   1. Itération1 :
      - $(S \to \bullet CC, \#)$ car $\# \in \textsc{First}(\#)$
   2. Itération2 :
      - $(C \to \bullet aC, a)$ et $(C \to \bullet aC, b)$ car $\textsc{First}(C\#) = \{a, b\}$
      - $(C \to \bullet b, a)$ et $(C \to \bullet b, b)$ car $\textsc{First}(C\#) = \{a, b\}$

Automate des items LR(1)

Automate des items LR(1)

Les ensembles d'items LR(1) fermés vont former un automate dont les transitions sont étiquetées par des symboles terminaux ou non-terminaux de la grammaire : $\mathcal{A} = (\{s_0, \dots, s_n\}, s_0, \delta, \varnothing)$ avec :

- $s_0 = Closure(\{(S' \to \bullet S, \#)\})$
- δ est définie de la manière suivante :
  - soit s_i un ensemble d'items et $X \in (N \cup T)$, $\delta(s_i, X)$ est défini comme la fermeture de l'ensemble des items $(A \to \alpha X \bullet \beta, a)$ où $(A \to \alpha \bullet X\beta, a)$ se trouve dans s_i

$$\delta(s_i, X) = \{Closure(\{(A \to \alpha X \bullet \beta, a)\}) \mid (A \to \alpha \bullet X\beta, a) \in s_i\}$$

Automate des items LR(1)

Algorithme de calcul de l'automate des items LR(1)

1. Ajouter $s_0 = Closure(\{(S' \rightarrow \bullet S, \#)\})$ aux états non-traités
2. Tant qu'il existe un état non-traité s_i
    1. Calculer $\mathcal{X}$ l'ensemble des symboles qui suivent directement le point $\bullet$
    2. Pour tous les symboles $X \in \mathcal{X} \subset (N \cup T)$
        1. Prendre l'ensemble I de tous les items où un point précède le symbole X
        2. Pour tous les items dans I, déplacer le point à droite de X
        3. Compléter par fermeture ce nouvel ensemble des items
        4. Ajouter le nouvel ensemble s_j obtenu aux états non-traités
        5. Ajouter (s_i, X, s_j) à δ

Automate des items LR(1)

Exemple (Automate des items LR(1))

Soit la grammaire $G = (\{S', S, C\}, \{a, b\}, S', R)$ avec R :

- $S' \to S$
- $S \to CC$
- $C \to aC|b$

- $s_0 = \{(S' \to \bullet S, \#)\} \cup \{(S \to \bullet CC, \#)\}$
$\cup \{(C \to \bullet aC, a), (C \to \bullet aC, b), (C \to \bullet b, a), (C \to \bullet b, b)\}$
- $s_1 = \delta(s_0, S) = \{(S' \to S \bullet, \#)\}$
- $s_2 = \delta(s_0, C) = \{(S \to C \bullet C, \#)\} \cup \{(C \to \bullet aC, \#), (C \to \bullet b, \#)\}$
- $s_3 = \delta(s_0, a) = \{(C \to a \bullet C, a), (C \to a \bullet C, b)\}$
$\cup \{(C \to \bullet aC, a), (C \to \bullet b, a), (C \to \bullet aC, b), (C \to \bullet b, b)\}$
- $s_4 = \delta(s_0, b) = \{(C \to b \bullet, a), (C \to b \bullet, b)\}$
- $s_5 = \delta(s_2, C) = \{(S \to CC \bullet, \#)\}$
- $s_6 = \delta(s_2, a) = \{(C \to a \bullet C, \#)\} \cup \{(C \to \bullet aC, \#), (C\,\dot{\bullet} b, \#)\}$
- $s_7 = \delta(s_2, b) = \{(C \to b \bullet, \#)\}$
- $s_8 = \delta(s_3, C) = \{(C \to aC \bullet, a), (C \to aC \bullet, b)\}$ et aussi $\delta(s_3, a) = s_3$ et $\delta(s_3, b) = s_4$
- $s_9 = \delta(s_6, C) = \{(C \to aC \bullet, \#)\}$ et aussi $\delta(s_6, a) = s_6$ et $\delta(s_6, b) = s_7$

Automate des items LR(1)

Exemple (Automate des items LR(1))

Soit la grammaire $G = (\{S', S, C\}, \{a, b\}, S', R)$ avec R :

- $S' \to S$
- $S \to CC$
- $C \to aC|b$

Table de transition :

États	a	b	S	C
0	3	4	1	2
1				
2	6	7		5
3	3	4		8
4				
5				
6	6	7		9
7				
8				
9				

Tables d'analyse LR(1)

Tables d'analyse LR(1)

Pour la table `Action` :

1. Si $(A \to \alpha \bullet a\beta, b) \in s_i$ et $\delta(s_i, a) = s_j$
   - $\texttt{Action}(s_i, a)$ vaut «`décaler` s_j»
2. Si $(A \to \alpha \bullet, b) \in s_i$ et $A \neq S'$ et $b \in \textsc{Follow}(A)$
   - $\texttt{Action}(s_i, b)$ vaut «`réduire` $A \to \alpha$»
3. Si $(S' \to S \bullet, \#) \in s_i$
   - $\texttt{Action}(s_i, \#)$ vaut «`accepter`»

Pour la table `Goto` :

1. Si $\delta(s_i, A) = s_j$
   - $\texttt{Goto}(s_i, A) = s_j$

Tables d'analyse LR(1)

Exemple (Tables d'analyse LR(1))

Soit la grammaire $G = (\{S', S, C\}, \{a, b\}, S', R)$ avec R :

- $S' \to S$
- $S \to CC$
- $C \to aC|b$

Tables d'analyse :

États	Action			Goto	
	a	b	$\#$	S	C
0	S3	S4		1	2
1			Acc		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

Grammaire LR(1)

Définition (Grammaire LR(1))

Une **grammaire LR(1)** est une grammaire telle que les tables d'analyse construites par l'analyse LR(1) sont déterministes

Proposition (Grammaire LR(1))

Si une grammaire est LR(1), alors :

- *elle est non-ambiguë*
- *l'algorithme d'analyse LR(1) donne une façon de construire l'unique arbre de dérivation d'un mot*

Proposition (Grammaire LR(1) et SLR(1))

Si une grammaire n'est pas LR(1) alors, elle n'est pas SLR(1)

Analyseurs LR et LL

Analyseurs LR et LL

La famille des analyseurs LR(k) permet d'analyser tous les langages LL(k) et bien d'autres. Les analyseurs LR(k) font des choix plus éclairés :

- LL(k) doit pouvoir sélectionner sans erreur une règle $A \to \alpha$ sur la seule base des k symboles terminaux dont tout ou une partie peut être dérivée de A
- LR(k) fonde sa décision d'appliquer une réduction $A \to \alpha$ sur la connaissance de l'intégralité de la partie droite α et la connaissance des k symboles terminaux à droite de A

Ainsi, l'analyse LR fait porter moins de contraintes sur la forme de la grammaire que l'analyse LL.

Limites

Limites

- L'analyse LR(1) fonctionne bien mais engendre des tables très grandes, y compris pour de petites grammaires.

→ On écrit rarement des analyseurs LR(1) à la main, on fait plutôt appel à des générateurs d'analyseurs syntaxiques tels que Bison (C, C++) ou Menhir (OCaml).

→ On utilise plutôt des tables d'analyses LALR(1) !

Plan

7 Analyse LR(1)

- Tables d'analyse LR(1)
- Tables d'analyse LALR(1)
- Conclusion

Analyse LALR(1)

Analyse LALR(1)

Une **analyse LALR(1)** *(LookAhead LR)* est une version simplifiée de l'analyse LR(1) qui produit des tables d'analyse plus petites.

- L'analyse LALR(1) est moins puissante que l'analyse LR(1)
- L'analyse LALR(1) sont plus puissante que l'analyse SLR(1)

$\rightarrow$ Bon compromis entre les deux !

Analyse LALR(1)

Transformation des tables d'analyse LR(1)

1. On fusionne les états qui correspondent aux mêmes items LR(0), c'est-à-dire qu'on efface les symboles de prévision
2. Deux cas peuvent se présenter :
   - La table avec les états fusionnés reste déterministe
     $\rightarrow$ La grammaire est LALR(1)
   - La table avec les états fusionnés contient des conflits
     $\rightarrow$ La grammaire n'est pas LALR(1)

Remarque

Seuls des conflits réduction/réduction peuvent apparaître

Analyse LALR(1)

Exemple (Analyse LALR(1))

- $s_0 = \{(S' \to \bullet S, \#)\} \cup \{(S \to \bullet CC, \#)\}$
  $\cup \{(C \to \bullet aC, a), (C \to \bullet aC, b), (C \to \bullet b, a), (C \to \bullet b, b)\}$
- $s_1 = \delta(s_0, S) = \{(S' \to S \bullet, \#)\}$
- $s_2 = \delta(s_0, C) = \{(S \to C \bullet C, \#)\} \cup \{(C \to \bullet aC, \#), (C \to \bullet b, \#)\}$
- $s_3 = \delta(s_0, a) = \{(C \to a \bullet C, a), (C \to a \bullet C, b)\}$
  $\cup \{(C \to \bullet aC, a), (C \to \bullet b, a), (C \to \bullet aC, b), (C \to \bullet b, b)\}$
- $s_4 = \delta(s_0, b) = \{(C \to b \bullet, a), (C \to b \bullet, b)\}$
- $s_5 = \delta(s_2, C) = \{(S \to CC \bullet, \#)\}$
- $s_6 = \delta(s_2, a) = \{(C \to a \bullet C, \#)\} \cup \{(C \to \bullet aC, \#), (C \dot{\bullet} b, \#)\}$
- $s_7 = \delta(s_2, b) = \{(C \to b \bullet, \#)\}$
- $s_8 = \delta(s_3, C) = \{(C \to aC \bullet, a), (C \to aC \bullet, b)\}$ et aussi $\delta(s_3, a) = s_3$ et $\delta(s_3, b) = s_4$
- $s_9 = \delta(s_6, C) = \{(C \to aC \bullet, \#)\}$ et aussi $\delta(s_6, a) = s_6$ et $\delta(s_6, b) = s_7$

On peut fusionner : l'état s_3 et l'état s_6 ; l'état s_4 et l'état s_7 ; l'état s_8 et l'état s_9.

Tables d'analyse LR(1)

Exemple (Tables d'analyse LR(1))

On peut fusionner :

- l'état s_3 et l'état s_6
- l'état s_4 et l'état s_7
- l'état s_8 et l'état s_9

Tables d'analyse LR(1) :

États	Action			Goto	
	a	b	$\#$	S	C
0	S3	S4		1	2
1			Acc		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

Tables d'analyse LALR(1) :

États	Action			Goto	
	a	b	$\#$	S	C
0	S3-6	S4-7		1	2
1			Acc		
2	S6	S4-7			5
3-6	S3-6	S4-7			8-9
4-7	R3	R3	R3		
5			R1		
8-9	R2	R2	R2		

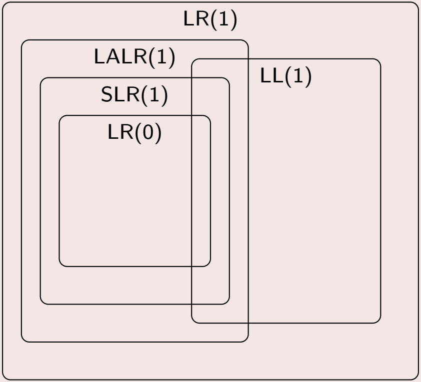
Plan

7 Analyse LR(1)

- Tables d'analyse LR(1)
- Tables d'analyse LALR(1)
- Conclusion

Conclusion

Conclusion



Puissance d'expression des grammaires

C'est tout pour le moment. . .

Des questions ?