Traduction – Chapitre 3 : Analyse Syntaxique

# Objectif

Il a pour objectif de vérifier qu’une phrase est correcte syntaxiquement. C’est-à-dire que cette phrase est conforme à la syntaxe du langage définie par une grammaire.

Il a aussi pour objectif de reconnaitre les unités lexicales, décrites par une grammaire. (Déclaration, instructions, variables, expression, fonctions et procédure, …)

Il a aussi pour objectif de détecter les erreurs syntaxiques. (Numéro de ligne et message clair, …). Dans certains cas, il propose une correction. En général, il continu l’analyse pour détecter d’autres erreurs. Il ne ralentit pas l’analyse d’un programme correct.

# Les erreurs syntaxiques

Quelques exemples d’erreurs syntaxique :

* Erreur de parenthésage
* Absence de condition dans un if
* Absence d’un ; quand il y a plusieurs instructions sur une ligne
* Manque un type de retour

LA double déclaration de variable est à la fois une erreur sémantique et lexicale.

# Grande famille d’analyseurs syntaxiques

Il existe deux grandes familles d’analyseur syntaxique :

* Descendant : ANTLR
  + Rappels
    - G = (N, T , ->, X)
    - Le mot vide : ou ^
    - Fin de texte : $
    - A -> aABbA => Grammaire récursive droite (non terminal à droite)
    - A -> A => Grammaire récursive gauche (non terminal à gauche)
    - Pour un analyseur syntaxique, la grammaire ne doit pas être ambiguë.
    - G est ambiguë si on peut construire 2 arbres syntaxiques différents pour reconnaitre 1 même phrase
    - LL(1)
      * Premier L : Lefet to right scanning
      * Deuxième L : Left mont denvation
    - G est LL(k) si on utiles les k prochaines unités lexicales pour décider de la production à appliquer lors d’une expansion
  + La construction de l’arbre syntaxiques se fait à partir de l’axiome, en
  + Lecture du texte de gauche à droite
  + Suite d’expression, c’est-à-dire, on accroche deux parties droites de production.
  + Exemple
    - Grammaire G, LL(1) : Pas notre cas ici.
      * X -> cAd
      * A -> ab | a

|  |  |
| --- | --- |
| Axiome | Texte |
| X | c a d $ |
| A | a d $ |
|  | d $ |
|  | $ |

* + - * Si on avait une grammaire LL(1), on n’aurait pas besoin de faire un aller-retour en arrière
      * Une grammaire LL(k) n’est forcément pas ambiguë.
* Ascendant : YACC ou BISON
  + Arbre syntaxique ,Analyse du code source pour arriver sur un axiome si tout va bien.
  + Principe
    - Partir de la phrase à analyser
    - Remplacer itérativement des fragments de la phrase courante qui correspondent à des membres droits d’A production, par le membre gauche de cette production
    - Analyse correcte si le mot courant Final est l’axiome de G

x

Arbre

Premier (x)

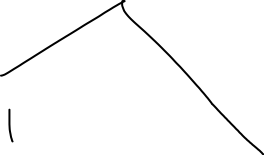
Suivant (x)

Analysée

Partie non analysée

Pile Déjà

Pile



Analyseur Ascendant prédictif LR(k)

* LR(K) : On utilise k unités lexicales du mot d’entrée pour faire la prédiction
  + Le L: Left to right scanning
  + Le R : Rigjt most derivation : on construit une dérivation droite à partir du mot à analyser

On a des familles d’analyse ascendantes utilisée par YACC :

* LR(0)
* SLR(1)
* LALR(1)

Ces algorithmes sont basés sur deux opérations :

* Le décalage (shift)
* L’opération de réduction (reduce)

Exemple :

S’ -> S

S -> Ac

A -> AaAb | d

On prend le mot dadbc$

|  |  |
| --- | --- |
| PILE | Texte |
| d  A | d a d b c $ |
| A d | a d b c $ |
| A a ~~d~~ A Ac[[1]](#footnote-1) | d b c $ |
| S | b c $ |



On remplace Ac par S puis par S’ (Réduction)

## Comment par algorithme, décider à chaque étape si on réduit ou si on lit ?

On va construire un automate déterministe. Il doit être LR(0) qui va dire quoi faire en fonction du contenu de la pile. Cette pile ne contient pas que des unités lexicales. Il va lire des choses et décider ce qu’il doit faire en fonction de la première unité lexicale de l’entrée.

## Automate LR(0) : Table ASA LR(0)

Définition

Un item d’une grammaire G est une production de G avec un « point » repérant une position de sa partie droite.

Intuitivement : C’est la « quantité » de partie droite qui a été reconnu par l’analyseur à un moment donné.

Exemple :

A ------> Xa fournit 3 items :

* A -------> .Xa
* A -------> X.a
* A -------> Xa.

Définition Fermeture :

Soit I un ensemble d’items de G.

Fermeture(I) = ensemble d’item construit à partir de I tel que :

* On place chaque item I dans Fermeture(I)
* Si A ------> . B Fermeture (I) et si B ------> alors ajouter B -------> à Fermeture(I) sauf s’il est déjà dedans.

On itère jusqu’à ne plus trouver d’item à ajouter à Fermture(I)

Définition  :

Les états de l’automate LR(0) sont les ensembles I d’items obtenus par fermeture.

L’état initial est obtenu par la fermeture de S’ ---------> .S

Définition :

Soit I un ensemble d’items.

X un symbole de G (X (NT))

On définit la transition (I, X) comme la la fermeture de l’ensemble des items A ----> X . tel que (sachant que) A ---> .X I

Exemple : Automate LR(0)

I11

S’ ----> S.

Pour fermeture (aucune)

I4

S

I2

S ----> Ac.

I.

c

I5

S’ -----> .S

S -------> .Ac

A ------>. AaAb

.d

S ----> A.C

A -----> A.aAb

Pour fermeture (aucune)

S’ -----> S A

G S -----> Ac

a

A -----> Aa.Ab

(fermeture A)

A ------> .AaAb

.d

A -----> AaAb

d

Pas de fermeture

I3

d

A

A -----> AaAb.

A -----> Aa.Ab

A ------> A.aAb

Pas de fermeture

A ---> d.

Pas de fermeture

a

d

I6

I7

b

Remarque : La construction de l’automate est indépendante du mot d’entrée.

Table d’ASA : Analyseur LR(0) :

Automate (LR(0)) + une PILE va permettre de dire ce qu’il faut faire, c’est-à-dire s’il faut « réduire » ou « lire » qui sont les 2 actions principales d’un ASA

L4automate LR(0) est codé dans 2 tables :

Non -Term

Term

* Table ACTION

TRANSITION

ACTION

* + Lecture

Ii

* + Réduction
  + Erreurs
  + Accepter les erreurs
* Table TRANSITION
  + C’est la fonction de transition restreinte aux non-terminaux

🡪 On remplit les tables à partir de l’automate LR(0)

$ Terminaux

(Mot vide) Terminaux (Le mot vide n’est pas une unité lexicale)

L’utilisation de la table :

* Lecture d’une unité lexicale ai

TABLE

Sorite

Im

ah

In

amam+1 … an$

ACTION

TRANSITION

* Consultation du sommet de la PILE Sm  
  Puis consultation de Table(Sm, ai)

Remplissage de la TABLE d’ASA LR(0) :

* Table ACTION
  + SI [ S’ ---------> S ]  Ii[[2]](#footnote-2) alors remplir ACTION(Ii, $) = « OK » (texte accepté)
  + Si [ X ----------> . A II alors ACTION (Ii, a) = « Lecture et aller à Ij »
  + Si [ X ----------> . ]  II alors
    - Ii est un état de réduction
    - On remplit ACTION(Ii, a[[3]](#footnote-3)) = « réduire par X ------->  »
  + Erreur dans les autres cas
* Table TRANSACTION
  + Transaction(Ii, X) = Ij

Table d’ASA LR(0) :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I | A | b | c | d | $ | S’ | S | A |
| 0 |  |  |  | d3 |  |  | 1 | 2 |
| 1 |  |  |  |  |  |  |  |  |
| 2 | d5 | d4 |  |  |  |  |  |  |
| 3 | Reduire  a 🡪d | r3 | r3 | r3 | r3 |  |  |  |
| 4 | r1 | r1 | r1 | r1 | r1 |  |  |  |
| 5 |  |  |  | d3 |  |  |  | 6 |
| 6 | D5 | D6 |  |  |  |  |  |  |
| 7 | R2 | R2 | R2 | R2 | R2 |  |  |  |

Les cases ou rien n’est indiqué, cela correspond à des erreurs.

Pas de conflit Lecture/réduction ou réduction / réduction => Analyse est LR(0)

R1 correspond aux règles de l’exemple d’automate LR(0) vu précédemment.

|  |  |  |
| --- | --- | --- |
| PILE | Action | Mot |
| 0 | D3[[4]](#footnote-4) | dadbc$ |
| 0d3 | C3  A 🡪 d | adbc$ |
| 0A  2 | d5 |  |
| 0A2a5 | D5 | dbc$ |
| 0A2a5d3 | R3 | bc$ |
| 0A2a5A  6 | D7 |  |
| 0A2aA6d7 | R2 | C$ |
| 0A2 |  | C |
| 0A2c4 | R1 | $ |
| 0S1 | OK | $ |

## Analyseur SLR(1)

Soit G’ :

E’ --------> E

E --------> E + T

T

T --------> T \* F

F

F --------> idf

I4

Automate LR(0)

+

I1

idf

F ----> idf

I4

T ----> F.

I3

F

T

I2

E ----> T.

T ----> T. + F

E

E’ ----> E.

E ----> E. + T

E’ ----> .E

E ----> .E + T

.T

T ----> .T \* F

.F

F ----> .idf

+

I5

|  |  |  |
| --- | --- | --- |
| PILE |  | Texte source |
| 0 | D4 | I+i\*i$ |
| 0i4 | Reduction  F ---> idf | +i\*i$ |
| 0F3 | Red  T ---> F | + |
| 0T2 | Red  E ---> T | + |
| 0E1 |  | +i\*i$ |
| 0E1 |  |  |

On résout le conflit

* Réduction par E’ ----> E
* Lecture et changement d’état

En regardant si l’unité lexicale courante (ici, un « + ») (=celle présente en tête du mot restant à analyser) était dans l’ensemble des suivants du non terminal membre gauche[[5]](#footnote-5) de la production[[6]](#footnote-6) utilisée pour la réduction

🡺 Formaliser par l’analyseur SLR(1) qui se base (toujours) sur l’automate LR(0)

Ici, + Suiv(E’)

Donc ce n’est pas une action de réduction qu’il faut appliquer mais l’action de lecture avec changement d’état.

Table d’analyse SLR(1) :

* Partie transition
  + Comme table LR(0)
* Partie Action
  + Ne change pas
    - Si [S’ ----> S]
    - Action(i, $) = « OK »
    - Si [ A ------> .a] et si Transition (Ii, a) = Ij alors ACTION(i, a) = dj
  + Si [ A -----> d. ]  : A + S’ alors ACTION(i, a) = « réduire par A ----> pour tout a Suivant(A)
  + Le reste : ERREUR

S’il n’y a pas de conflit, l’analyseur est SLR(1)

## Analyseur LR(1) (et LALR(1))

Soit Ii

A ----> B.aDa

D ----> B.

Et Ij :

A -----> Ba.D

Si on lit a : Avancer et changement d’état et aussi réduire par D ---> B car a Suiv(D) ==> Conflit

Solution :

* On ajoute une information supplémentaire dans les items
* Un item devient A ----> ., a[[7]](#footnote-7) ou A --🡪
  + Ceci est un automate LR(1)

Définition : Construction des items LR(1)

* G’ augmentés
  + Item LR(1) : une production avec un . (un point) et un contexte
  + Fermeture : État I, I’ = Fermeture(I)
    - Placer les items de I dans I’
    - Si [ A ----> .B[
    - Si B --->
    - Si b Premier(a)
* Alors ajoute
  + « B ----> , b » à fermeture(I) (sauf si déjà présent dedans)
  + Boucler le processus

S’ ----> S

S -----> C C

C -----> a C

d

Construire l’automate LR(1), (à partir d’items LR(0)) :

Contexte, les terminaux qui sont derrière S.

I0 : S’ -----> S, $



S -----> .CC, $

C -----> .C,a

.d, a C ----> .aC [a,d]

.aC,d .d, [a,d]



.d, d

I1 : Avance (I0, S)

S’ -----> S., $ (Pas de calcul de nouveaux contextes)

I2 : Avance(I0, C)

S -----> C . C, $ (on garde le $)

Fermeture de C, et calcul des contextes



C ----> .aC , $

.d, $

I3 : Avance (I3, a)

C ----> a.C [a,d]

.aC, [a d]

.d, [a d]

I4 : AV(I0, d)

C --🡪d.,[a d]

I5 : AV(I2, C)

S 🡪 C C . ,$

I6 : AV(I2, C)

C 🡪 a.C,$

.aC,$

.d,$

I7 : AV(I2 , d)

C 🡪d.,$

I8 : AV(I3, C)

C 🡪 aC.,[a d]

I3 = AV(I3, a)

I4 = AV(I3, d)

I9 = AV(I6 , C)

C 🡪 aC.,$

I6 = AV(I6, a)

I7 = AV(I0, d)

Analyseur LR(1) : Construction de la table comme SLR(1) sauf les états de réduction si [ A -🡪 ; a ] : I i ; A ≠ S’

Alors Action (Ii, a) = réduire par A -----> ’

🡪

B🡪

I4 ~ I7 ) Même noyaux, contexte ≠

I3 ~ I6

I3 ~ I9

Table LR(1)



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | d | $ |  | S | C | S’ |
| I0 | d3 | d4 |  |  | 1 | 2 |  |
| I1 |  |  | OK |  |  |  |  |
| I2 | d6 | d7 |  |  |  | 5 |  |
| I3 | d5 | d4 |  |  |  | 8 |  |
| I4 | r3 | r3 | r3 |  |  |  |  |
| I5 |  |  | r1 |  |  |  |  |
| ~~I~~~~6~~ | ~~d6d3~~ | ~~d74~~ |  |  |  | ~~98~~ |  |
| ~~I~~~~7~~ | ~~-----~~ | ~~-----~~ | ~~r3~~ |  |  |  |  |
| I8 | r2 | r2 | r2 |  |  |  |  |
| ~~I~~~~9~~ | ~~------~~ | ~~-----~~ | ~~r2~~ |  |  |  |  |

La fusion est possible sans conflit donc l’analyseur est LALR(1)

K dans LR(k) et SLR(k) correspond au nombre d’unité lexical nécessaire pour prendre une décision

# Analyseur ascendant et grammaire ambiguë

Soit S ---> if E then S else S

-->if E then S

La phrase if a then if b then S1 else S2 pose un problème

🡪 (1) if a then {if b then S1 else S2}

(2) if a then { if b then S1} else S2

1. Est bon mais pas le (2)

En général, le else se rapporte au else le + récent

S -------> if E then S.[[8]](#footnote-8)

-----> if E then S. then S[[9]](#footnote-9) 🡺 Conflit lecture (Premier S)/reduction

Pour lever le conflit, il y a deux solutions :

* Transformer la grammaire
* Résoudre les conflits en utilisant les précédents (⬄ priorités)

Pourquoi le fait d’utiliser les précédents supprime certains conflits en ASA ?

Soit P ----> . t { … } Shift sur t

A ----> u R. { … t …} Réduction pour t

Si le symbole de prévision (contexte) est t

3 cas :

* u à une précédence supérieur à t :
  + prec(u) > prec(t) => REDUIRE
* prec(u) < prec(t) => LECTURE
* prec(u) = prec(t) => peu importe || mais utiliser les associativités

1. On a d’abord ajouté le d, mais vu que cela forme AdAb, on l’a remplacé par le A lui-même réduction. [↑](#footnote-ref-1)
2. État [↑](#footnote-ref-2)
3. Pour tous a (T $) [↑](#footnote-ref-3)
4. Lecture et aller à état I3 [↑](#footnote-ref-4)
5. Ici, E’ [↑](#footnote-ref-5)
6. E’ ----> E [↑](#footnote-ref-6)
7. Symbole de prévision, ou symbole de contexte (la même chose, juste deux dénomination différentes) [↑](#footnote-ref-7)
8. Réduction [↑](#footnote-ref-8)
9. Lecture [↑](#footnote-ref-9)