Cours de Traductions

Florian Touraine

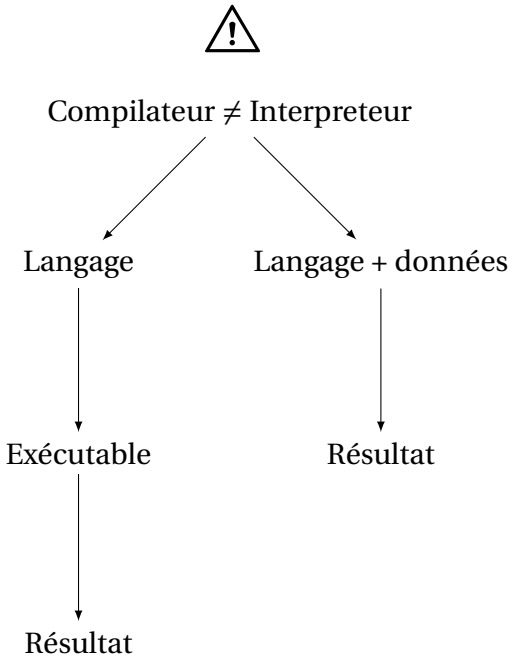
30 septembre 2024

Table des matières

1 Introduction

1.1 Qu'est ce qu'un compilateur ?

Compilateur : Logiciel qui traduit un programme écrit dans un langage de haut niveau (langage source) en *intructions exécutables* (langage cible). Un compilateur est composé d'algorithmes et de structures de données nécéssaire à la traduction.



Le compilateur : lit le code source fourni et renvoie un fichier résultat contenant le code machine (appelé aussi fichier cible) exécutable une fois pour toute.
Le C, le C++, le Fortran, le Pascal, le Cobol, l'Ada... sont des langages compilés.

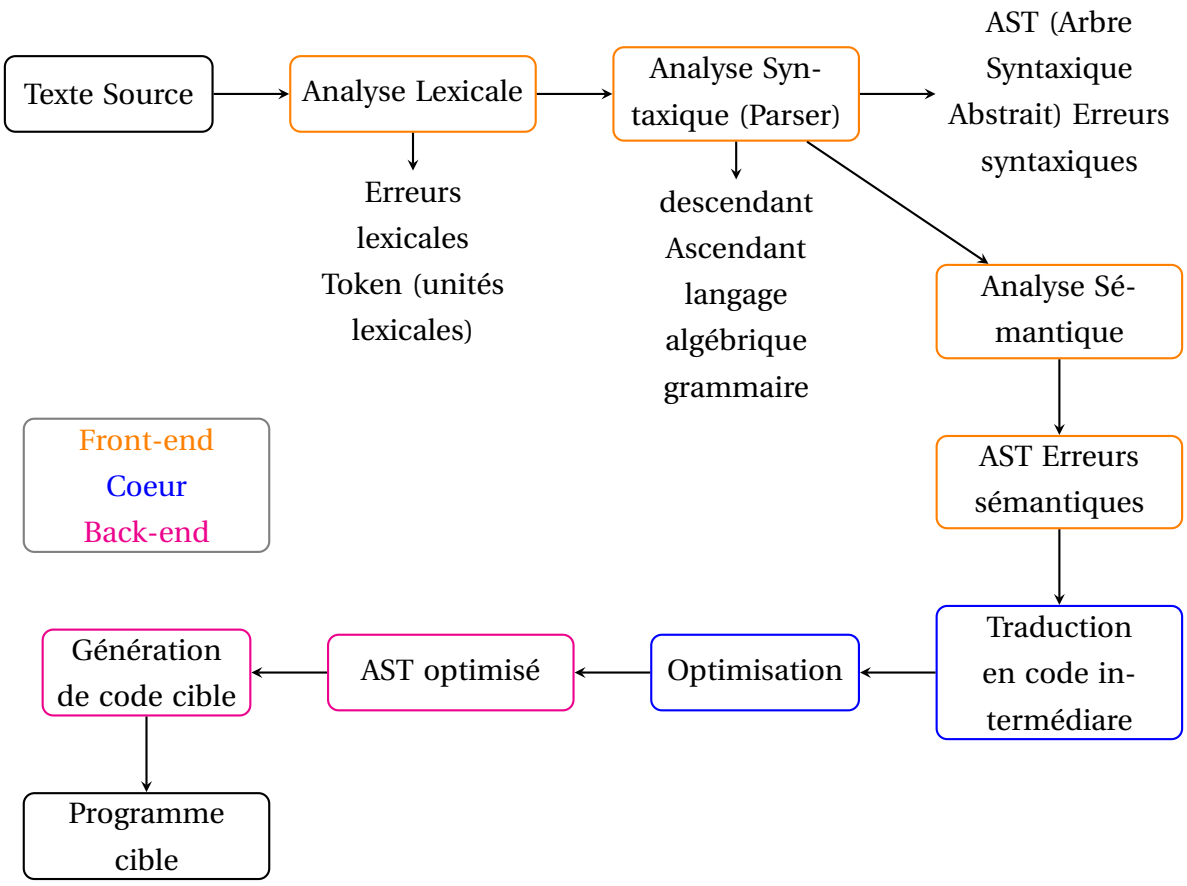
L'interpreteur : ne crée pas de fichier cible. Il interprète et exécute au fur et à mesure les instructions du programme source.
L'OCaml, le PHP, le Python, le sh, le bash... sont des langages interprétés.

Certains langages ne sont ni compilés, ni interprétés totalement (c'est un « mélange » des deux) : c'est le cas du langage Java. En effet, le résultat de la compilation d'un code source java est un fichier non pas en langage machine, mais en un bytecode java qui n'est pas directement exécutable par la machine. Le bytecode généré par le compilateur javac est interprété et exécuté par la machine virtuelle java : la JVM.

Aperçu historique :

- **1954 :** Fortran
- **1958 :** LISP
- **1959 :** COBOL
- **1966 :** SINULA (Classes et Objets)
- **1971 :** PASCM
- **1972 :** langage C et PROLOG
- **1976 :** ADA (généricité et type abstrait) et Caml
- **1982 :** C++
- **1991 :** Java, Rust, Python, HTML, BD, Tex

1.2 Schéma général d'un compilateur



2 Analyseur lexical

Analyseur lexicale :

- le texte source caractère par caractère (seul module du compilateur qui lit le texte)
- reconnait les unités lexicales (token) qui sont les mots du langage et les présente à l'annalyseur syntaxique
- c'est un automate

Principales unités lexicales :

- caractères simples : + − * / ( ) { } [ ]
- Caractères à deux caractères : ++ −− := <= >=
- identificateurs : x g $factorielle$
- constantes numériques : 123 4,5
- mots cléfs : if $then$

Autres tâches du lexeur :

- Supprimer les tabulations, les caractères de fin de lignes
- Supprimer les commentaires
- Afficher les erreurs lexicales : caractères inconnus, idf trop long, constante trop grande

Le lexeur « code » les unités reconnues.

Lexique : Table dans laquelle on range chaque identificateur présent dans le texte source, avecv un indice de rang.

3 Analyse Syntaxique

Il exite deux types d'analyses syntaxiques, l'analyse syntaxique ***Descendante*** (cf cours 1A) et l'analyse syntaxique ***Ascendante***.

Le but :

- vérifier que la phrase en entrée est conforme à la syntaxe du langage (défini par la grammaire)
- reconnaitre les unités syntaxiques décrites par la grammaire (déclaration, instruction, variables, expressions)
- détecter les erreurs syntaxiques (parfois le parseur propose 1 correction)
- localise l'erreur : le numéro de la ligne, donne un message clair
- ne s'arrête pas à la 1ere erreur rencontrée

3.1 Analyse syntaxique descendante

Les actions principales en analyse ascendante sont le *décalage (ou lecture)* et la *réduction*.

Algorithme :

- on part du mot à analyser
- on remplace itérativement des fragments du mot courant qui correspondent exactement à des membres droits de la règle de production par le membre gauche de cette règle
- succès si le mot final est l'axiome de la grammaire

3.2 Exemple

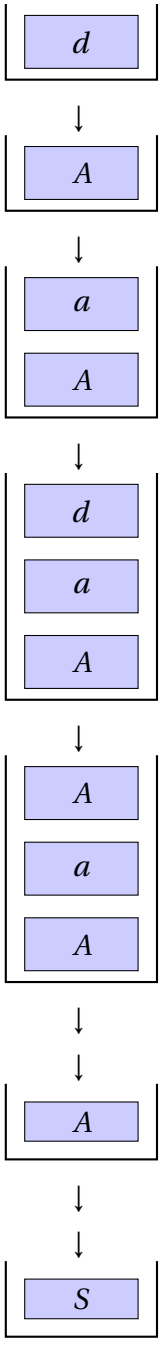
Soit une grammaire :

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow Ac \\ A &\rightarrow AaAb \mid d \end{aligned}$$

Le mot à analyser : $dadbc\$$

Analyse :

- On place d dans la pile.
- On remarque que d est donné par A.
- On remplace d par A.
- On obtient donc $\boldsymbol{Aadbc\$}$.
  Réduction : remplacer d par A revient à dépiler d, empiler A.
- Aucun membre gauche des règles n'est seulement A, on empile donc a le deuxième caractère.
- On regarde si a peut être réduit (non).
  On regarde alors si Aa peut être réduit (non).
  Donc on continue d'empiler.
- On peut réduire d donc on le remplace par A.
- Ainsi de suite on arrive à $\boldsymbol{AaAbc\$}$.
- Puis à $\boldsymbol{Ac\$}$.
- Puis à $\boldsymbol{S\$}$ l'axiome.
  Le mot est donc valide.



Dénomination des ascendants :
Famille LR(k) left to right scanning (Right most scanning) On utilise les k unités lexicales du mot d'entrée pour faire la prédiction.

3.3 Analyseur ascendant LR(0)

Définition : C'est une production de grammaire avec un marqueur en partie droite.
$[A \rightarrow aAbB]EG$

Remarque : Le mot vide n'est pas une unité lexicale. **On ne lit pas le mot vide.**
On ne supprime pas le caractère vide dans la grammaire on le remplace par .
Exemple : $A \rightarrow \epsilon$ devient $A \rightarrow .$

On définit :
I_i : un ensemble d'items obtenus par fermeture.
Fermeture(I) : un ensemble d'items construit à partir de I selon l'algorithme suivant.

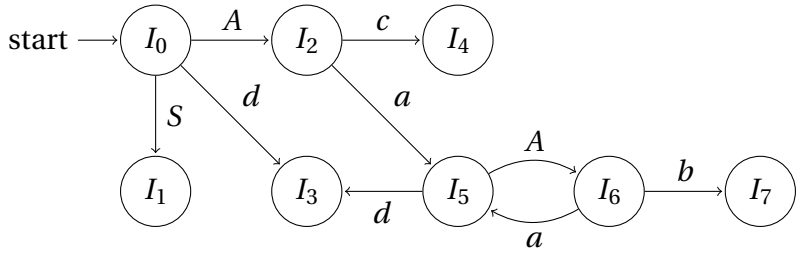
- Placer chaque item de I dans $Fermeture(I)$.
- Si $A \rightarrow \alpha.B\beta$ appartient à *Fermeture(I)* et $B \rightarrow \gamma$
  alors ajouter $\beta \rightarrow .\gamma$ à *Feremture(I)*
- Itérer jusqu'à ne plus trouver d'items à ajouter à *Fermeture(I)*

3.4 Exemple

Soit la grammaire :

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow Ac \\ A &\rightarrow AaAb \mid d \end{aligned}$$

Étape 1 : On construit l'automate *LR(0)*



$$\underline{I_0}: \begin{array}{lcl} S' & \to & .S \\ S & \to & .Ac \\ A & \to & .AaAb \mid .d \end{array} \qquad \underline{I_1}: S' \to S. \qquad \underline{I_2}: \begin{array}{lcl} S & \to & A.c \\ A & \to & A.aAb \end{array} \qquad \underline{I_3}: A \to d.$$

$$\underline{I_4}: S \to Ac. \qquad \underline{I_5}: \begin{array}{lcl} A & \to & Aa.Ab \\ A & \to & .AaAb \mid .d \end{array} \qquad \underline{I_6}: \begin{array}{lcl} A & \to & AaA.b \\ & & A.aAb \end{array} \qquad \underline{I_7}: A \to AaAb.$$

I_0 est obtenu par fermeture de l'axiome
La deuxieme règle de I_5 est obtenu par fermeture la première

L'automate est bien $LR(0)$
L'automate $LR(0)$ est en fait « codé » dans une table partagée en :

- table « ACTION »
- table « TRANSITION »

Table d'ASA $LR(0)$

états	ACTION	TRANSITION
	T	N

Table ACTION :

- Si $[S' \rightarrow S.]$ appartient à I_i alors $ACTION(I_i, \$) =$ « $accpeter$ »

- Si $[X \rightarrow \beta.]$ appartient à I_i avec $X \neq S'(axiome)$ alors
  $\forall a \in \{T, \$\}$, $ACTION(I_i, a) =$ « réduit par $X \rightarrow \beta$ »

- Si $[X \rightarrow \alpha.a\beta]$ appartient à I_i alors $ACTION(I_i, a) =$ « aller à l'état I_j »
- « erreur » dans les autres cas

Table TRANSITION :

- Si dans l'automate, on a $TRANSITION(I_i) = I_j$ alors $TRANSITION(I_i) = j$

On va donc construire cette table sur l'exemple précedant.

	a	b	c	d	\$	S	A	S'
I_0				d_3		1	2	S'
I_1					OK			
I_2	d_5		d_4					
I_3	r_3	r_3	r_3	r_3	r_3			
I_4	r_1	r_1	r_1	r_1	r_1			
I_5				d_3			6	
I_6	d_5	d_7						
I_7	r_2	r_2	r_2	r_2	r_2			

r_x : Réduit par la règle x

d_x : décaler à l'état I_x

1. $S \rightarrow Ac$
2. $A \rightarrow AaAb$
3. $A \rightarrow d$

Pas de conflit ⇒ L'analyseur est LR(0)

Fonctionnement :

États	**Pile**	**Texte source**
$O \rightarrow$	d_3 + lecture	$\leftarrow dadbc\$$
$Od3 \rightarrow$	r_3	$\leftarrow adbc\$$
$OA2$		$adbc\$$
$OA2 \rightarrow$	d_5 + lecture	$\leftarrow adbc\$$
$OA2a5 \rightarrow$	d_3 + lecture	$\leftarrow dbc\$$
$OA2a5d3 \rightarrow$	r_3	$\leftarrow bc\$$
$OA2a5A6$		$bc\$$
$OA2a5A6 \rightarrow$	d_7 + lecture	$\leftarrow bc\$$
$OA2a5A6b7 \rightarrow$	r_2	$\leftarrow c\$$
$OA2$		$c\$$
$OA2 \rightarrow$	d_4 + lecture	$\leftarrow c\$$
$OA2c4 \rightarrow$	d_4 + lecture	$\leftarrow \$$
$OA2c4 \rightarrow$	r_1	$\leftarrow \$$
$OS1$		$\$$
$OS1$	OK	$\leftarrow \$$

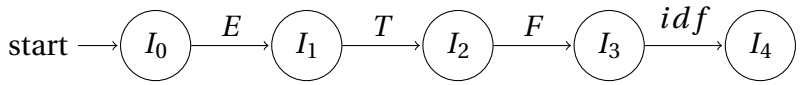
Cependant il peut y avoir des conflits entre differents règles à appliquer. Nous allons donc voir un exemple de conflit.

3.5 Exemple de conflit

Soit une grammaire :

$$\begin{aligned} E' &\rightarrow E \\ E &\rightarrow E+T|T \\ T &\rightarrow T*F|F \\ F &\rightarrow idf \end{aligned}$$

Avec l'arbre syntaxique :



$$\underline{I_0:}\ \begin{array}{lcl} E & \to & .E \\ E & \to & .E+T \mid .T \\ T & \to & .T*F \mid .F \\ F & \to & .idf \end{array} \qquad \underline{I_1:}\ \begin{array}{lcl} E' & \to & E. \\ E & \to & E.+T \end{array} \qquad \underline{I_2:}\ \begin{array}{lcl} E & \to & T. \\ T & \to & T.*F \end{array} \qquad \underline{I_3:}\ T \to F.$$

$$\underline{I_4:}\ F \to idf.$$

Avec sa pile :

États	**Pile**	**Texte source**
$O \to$	d_3 + lecture	$\leftarrow a+a*a\$$
$Oa4 \to$	réduction	$\leftarrow +a*a\$$
$OF3$	reduction	$\leftarrow +a*a\$$
$OT2$	reduction	$\leftarrow +a*a\$$
$OE1 \to$	reduction/lecture	$\leftarrow +a*a\$$

On a bien un conflit à la derniere ligne de la pile car on peut choisir de faire une reduction ou une lecture.

De manière générale on a :

$$\boxed{\exists\ \ E' \overset{*}{\mapsto} E'\beta\$\ ,\ \ \beta \neq \epsilon,\ \ suivant(E') = \{\$\}\ \ \Rightarrow \text{pas } LR(0)}$$

3.6 Analyseur syntaxique SLR(1)

Analyseur Syntaxique SLR(1) est :

- Basé sur un automate LR(0)
- Basé sur une table syntaxique LR(0) à une différence près pour la réduction

Table ACTION :

- Si $[S' \rightarrow S.]$ appartient à I_i alors $ACTION(I_i, \$) =$ « $accpeter$ »

- Si $[X \rightarrow \beta.]$ appartient à I_i avec $X \neq S'(axiome)$ alors
  $\forall a \in$ **Suivant(X)**, $ACTION(I_i, a) =$ « réduit par $X \rightarrow \beta$ »

- Si $[X \rightarrow \alpha.a\beta]$ appartient à I_i alors $ACTION(I_i, a) =$ « aller à l'état I_j »
- « erreur » dans les autres cas

Table TRANSITION :

- Si dans l'automate, on a $TRANSITION(I_i) = I_j$ alors $TRANSITION(I_i) = j$

La seule différence est donc que l'on réduit non plus pour tous les caractères terminaux mais seulement pour les suivants de la règle.
Un exemple à été vu en TD donc je ne le traite pas ici.

Puisque les suivants sont un sous ensemble de T (les Terminaux),
il est facile de montrer que :

$$\boxed{LR(0) \Rightarrow SLR(1)}$$

3.7 Exemple de conflit

Soit deux états :

$$\underline{I_i :} \begin{array}{lcl} E & \rightarrow & B.aDa \\ E & \rightarrow & B. \end{array} \qquad \underline{I_j :} \; E' \rightarrow Ba.d$$

Si on lit le caractère a, deux solutions s'offrent à nous :

- $ACTION(I_i, a)$ = « décaler à I_j »
- $ACTION(I_i, a)$ = « réduit par $D \rightarrow B$ », car $a \in$ suivant(D)

On a donc bien un conflit dû à la construction. On va donc palier le problème avec un analyseur amélioré par sa construction.

3.8 Analyseur syntaxique LR(1)

On va ajouter une information supplémentaire dans les items.
Un item devient :
$[A \rightarrow \alpha.\beta, a]$ où $A \rightarrow \alpha\beta \in G$ et a un Terminal ou \$

Ce symbole a est appelé un *contexte (symbole de prévision)*. Ce contexte n'aura effet que sur les items de la forme $A \rightarrow \alpha.,\ a$

Table ACTION :

- Si $[X \rightarrow \beta.,\ a]$ appartient à I_i avec $X \neq S'(axiome)$ alors $ACTION(I_i, a) =$ « réduit par $X \rightarrow \beta$ », **uniquement pour a**

- Le reste est similaire.

Définition et construction des items LR(1)

Items LR(1) : $[A \rightarrow \alpha.\beta,\ a]$ sont obtenus par *fermeture*.

Fermeture(I) :

- Si $[A \rightarrow \alpha.B\beta,\ a] \in I$ avec $[B \rightarrow \gamma] \in G$ alors ajouter $[B \rightarrow .\gamma,\ b]$ avec $b \in Premier(\beta a)$

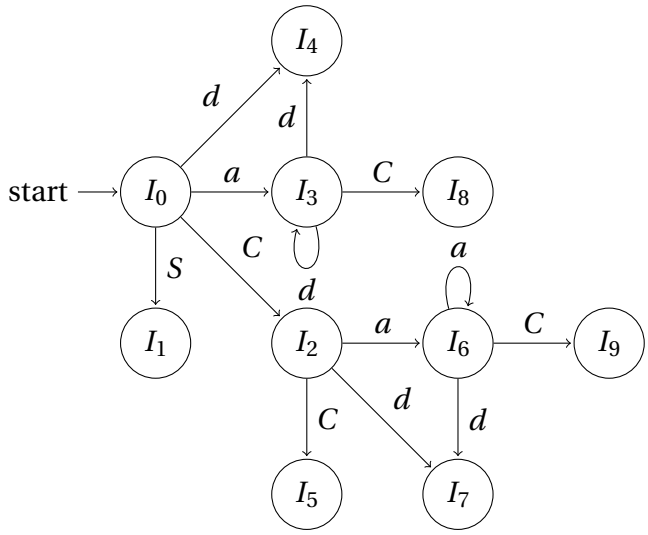
Le calcule des transitions et de la table ne changent pas.

3.9 Exemple

Soit une grammaire :

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow CC \\ C &\rightarrow aC \mid d \end{aligned}$$

Avec l'arbre syntaxique :



$$\underline{I_0 :} \begin{array}{lcl} S' & \to & .S, [\$] \\ S & \to & .CC, [\$] \\ C & \to & .aC, [a,d] \mid .d, [a,d] \end{array} \qquad \underline{I_1 :}\ S' \to S., [\$] \qquad \underline{I_2 :} \begin{array}{lcl} S & \to & C.C, [\$] \\ C & \to & .aC, [\$] \mid .d, [\$] \end{array}$$

$$\underline{I_3 :} \begin{array}{lcl} C & \to & a.C, [a,d] \\ C & \to & .aC, [a,d] \mid .d, [a,d] \end{array} \qquad \underline{I_4 :}\ C \to d., [a,d] \qquad \underline{I_5 :}\ S \to CC., [\$]$$

$$\underline{I_6 :} \begin{array}{lcl} C & \to & a.C, [\$] \\ C & & .aC, [\$] \mid .d, [\$] \end{array} \qquad \underline{I_7 :}\ C \to d., [\$] \qquad \underline{I_8 :}\ C \to aC., [a,d]$$

$$\underline{I_9 :}\ C \to aC., [\$]$$

La table LR(1) associée :

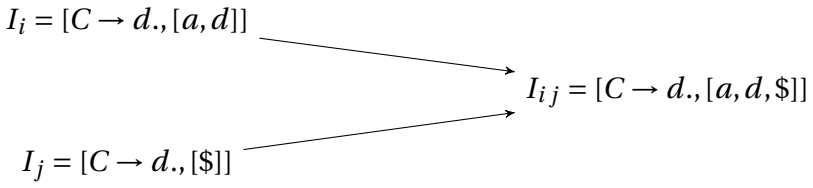
	a	d	\$	C	S	S'
I_0	d_3	d_4		2	1	
I_1			OK			
I_2	d_6	d_7		5		
I_3	d_3	d_4		8		
I_4	r_3	r_3				
I_5			r_1			
I_6	d_6	d_7		9		
I_7			r_3			
I_8	r_2	r_2				
I_9			r_2			

Cepedant on a des etats identiques aux contextes près.
$[I_4 - I_7]$, $[I_8 - I_9]$, $[I_3 - I_6]$

3.10 Analyseur syntaxique LALR(1)

- À partir de l'automate LR(1) et de ses items
- Principe : fusionner les états ayant même noyau

Le principe de la fusion est :



S'il n'y a pas de complit dans dans la table résultant de la fusion, alors ta table est LALR(1) et on puet donc construire un analyseur LALR(1).

3.11 Exemple

Pour la grammaire précédante, la table devient :

	a	d	\$	C	S	S'
I_0	d_{36}	d_{47}		2	1	
I_1			OK			
I_2	d_{36}	d_{47}		5		
I_{36}	d_{36}	d_{47}		89		
I_{47}	r_3	r_3	r_3			
I_5			r_1			
I_{89}	r_2	r_2	r_2			

3.12 Analyseur LR(1) et grammaire ambigüe

Soit la grammaire :

$$\begin{array}{lllllll} S & \rightarrow & if & E & then & S & else \quad S \\ S & \rightarrow & if & E & then & S & \end{array}$$

Soit la phrase : if a then if b then S1 else S2

Il peut y avoir deux interprétations différentes :

- if a then if b then S1 else S2
- if a then if b then S1 else S2

On a donc une grammaire qui est ambigüe.

Comment régler le problème ?

- Modifier la grammaire
- Garder la grammaire et utiliser les *précédences*

3.13 Utiliser les précédences des tokens

Par exemple : $*$ a une précédence plus haute / forte que +
De cette maniere, on va pouvoir faire un choix.

Soit :

$$\begin{array}{lll} P & \rightarrow & \alpha.\underline{t}\beta, [..] \quad decale\ sur\ t \\ Q & \rightarrow & \gamma\underline{u}R., [\underline{t}] \quad reduit\ pour\ t \end{array}$$

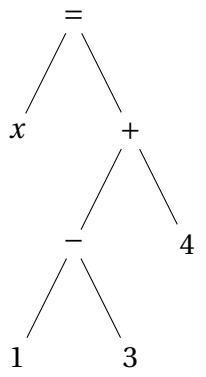
Dans l'item γuR, u est déjà empilé.

Si le symbole de prévison est t, on a trois possibilités. On va comparer u et t :

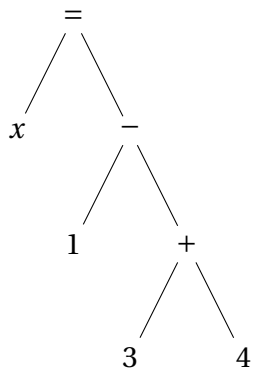
- $prec(u) > prec(t) \Rightarrow REDUCE : (3 * 4) + 2$
- $prec(u) < prec(t) \Rightarrow SHIFT(lecture) : 3 + 4 * 2$
- $prec(u) = prec(t) \Rightarrow SHIFT/REDUCE : 1 + 2 - 3$

Exemple :

Pour $x = 1 - 3 + 4$ par associativité gauche, on a :



et surtout pas



Remarque : Plus un oppérateur est prioritaire, plus il sera bas dans l'arbre syntaxique.

4 Analyse sémantique

Le but de l'annalyse sémantique est de vérifier que le programme satisfait un ensemble de règles de constructions définies par le langage.