Definitions

AST (Abstract syntax tree) ou ASA (Arbre Syntaxique Abstrait)

Un AST est une structure permettant de représenter la construction d'un code a partir d'une grammaire, les noeuds représentent les non terminaux de la grammaire et les feuilles les symboles terminaux, dans le cadre d'un compilateur, il est produit par le parser

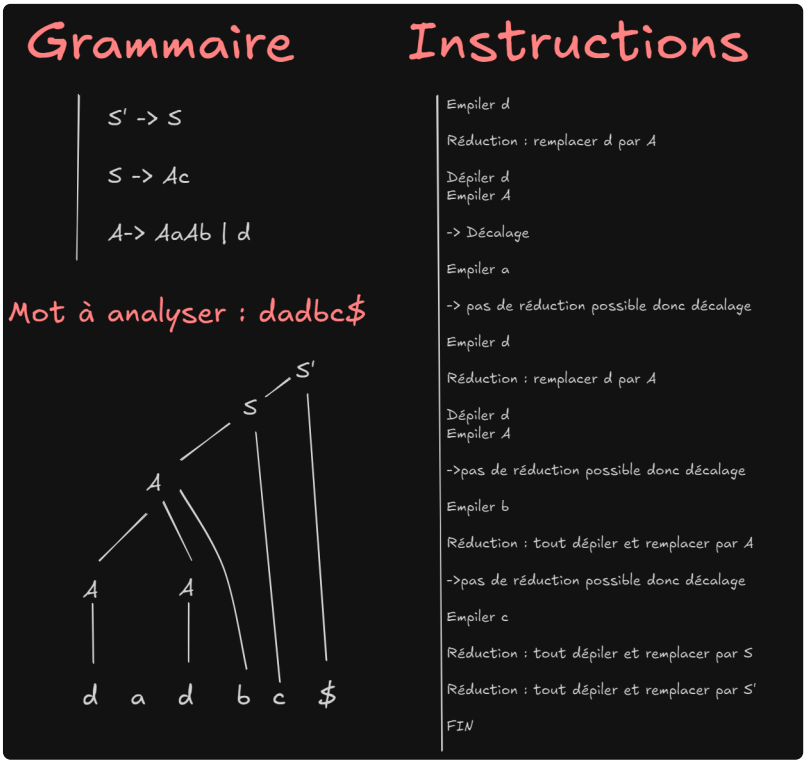
Arbre syntaxique abstrait

Ce qui différencie un arbre syntaxique abstrait d'un arbre syntaxique normal c'est la suppression de tout les noeuds "inutiles" dans le sens qu'ils ne donnent aucune information et ne servent uniquement qu'a construire l'arbre
Typiquement, un noeud avec un seul fils est inutile et peut être supprimé, le fils de ce noeud allant directement au père de ce noeud. ce processus en général s'appelle l'élagage

Analyse syntaxique ascendante et famille LR(k)

Le principe d'une analyse syntaxique ascendante (voir parser : voir AST) est de construire notre arbre syntaxique en partant du bas (donc des feuilles) , pour cela on construit un automate LR(k) (généralement 1 ou 0)
Un automate LR(k) signifie qu'on lit le texte de gauche à droite (Left to right scanning, Rightmost derivation) et le k signifie le nombre de caractères qui nous faut lire en avance du caractère qu'on est en train de traiter pour prendre une décision(soit décalage ou réduction), ainsi, dans un automate LR(0) , nous n'avons pas besoin de connaitre le caractère d'après pour construire notre arbre syntaxique

Analyse syntaxique ascendante exemple



Item

C'est la production de G avec un marqueur en partie droite.

Exemple de construction d'un automate LR(0)

$$G \begin{cases} A' \to A \quad r0 \\ A- > V|(A.\ A)|(AS) \quad r1, r2, r3 \\ S- >, AS|\varepsilon \quad r4, r5 \\ V- > entier|nil \quad r6, r7 \end{cases}$$

- On part de l'état 0, avec tout les items liés a la fermeture de l'axiome
- Pour chaque terminal ou non terminal qu'on peut lire à partir de ces items on construit un nouvel état avec l'item qu'on a utilisé, on avance notre marqueur d'un

Etat I_i : un ensemble d'items obtenus par fermeture (un ensemble d'items construits à partir de I selon l'algo suivant :)

- placer chaque item de I dans fermeture(I).
- si $[A \to \alpha.B\beta] \in$ Fermeture(I) et $[B \to \gamma]$, alors ajouter $[B \to \gamma]$ à Fermeture(I) sauf si déjà présent.
- Itérer jusqu'à ne plus trouver d'items à ajouter à Fermeture(I)

- ### Et en français ?

  Pour chaque état, pour chaque règle, si on à un non terminal sur le caractère a gauche de notre marqueur, on rajoute l'ensemble des règles produites par ce non terminal avec le marqueur partant du début
  Pour l'état initial on part de l'axiome

Sinon, on part des règles ou on peut avancer le marqueur en lisant un caractère spécifique

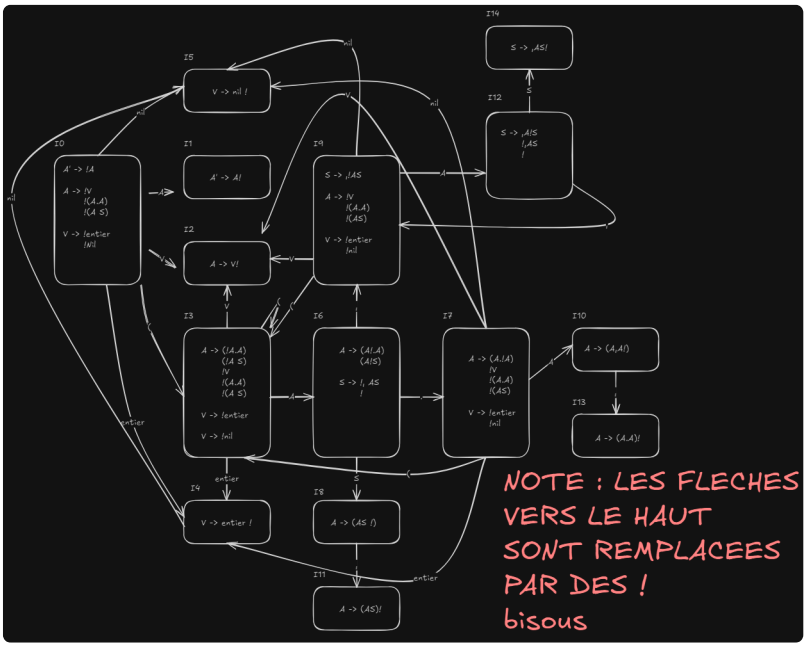


Table SLR(1)

La table SLR(1) (pour Short LR) reprend les informations de l'automate LR(0) dans une table divisée en deux sous tables, une table d'action pour les terminaux et une table de transition pour les non terminaux: elle se remplit de la manière suivante

- Pour les non terminaux
  - Si on va vers un état, mettre l'état
- Pour les terminaux
  - Si on va vers un état , mettre d suivi de état
- Quand il y'a un item avec ! a la fin , exemple A -> a!
  - mettre r : réduction avec le numéro de la réduction liée a la règle utilisée aux suivants de A (si A est en fin de règle on met dollar)
- Lorsqu'on a lu l'axiome dans l'état i ,
  - Mettre OK ligne i colonne dollar

	(	)	Entier	Nil	.	,	$	A'	A	S	V
I0	d3		d4	d5					1		2
1							OK				
2		r1			r1	r1	r1				
3	d3		d4	d5					6		2
4		r6		d5	r6	r6	r6				
5		r7			r7	r7	r7				
6		r5			d7	d9				8	

	(	)	Entier	Nil	.	,	$	A'	A	S	V
7	d3		d4	d5					10		2
8		d11									
9	d3		d4	d5					12		2
10		d13									
11		r3			r3	r3	r3				
12		r5				d9				14	
13		r2			r2	r2	r2				
14		r4									

Si il n'y a pas de conflits lecture/reduction, alors l'analyseur syntaxique est SLR(1)

Table LR(0)

Même principe que la table SLR(1) , se base sur un automate LR(0), la partie reduction est construite différement mais j'ai pas compris encore

Lecture d'un mot par une pile

- On regarde l'état dans lequel on est, si le caractère suivant est dans la table et lisible on le lit, on le met dans la pile ainsi que l'état dans lequel on va
- Si il y'a une réduction, on remplace tout ce qui est a remplacer dans la pile par ca, on revient dans l'état avant la réduction (donc celui a gauche de tout ce qu'on a enlevé , si c'est espilon on n'enleve rien)
- Le mot est lisible si on arrive a OK a la fin

Automate LR(1)

Contexte ?

Le contexte, ou symbole de précision, est rajouté a un item pour un analyseur syntaxique LR(1) , il permet de savoir un caractère en avance et donc de créer des items différents pour éviter les shift reduce conflicts
Lorsqu'on calcule un état, et qu'on ajoute item par fermeture, on prend le premier de ce qui suit le non terminal qu'on a ajouté, a défaut, on remet le contexte précédent

Etat de l'automate LR(1)

- Ensemble des items LR(1) obtenus par fermeture
- Calcul des fermetures pour un état I:
  - Si $[A \rightarrow \alpha . B\beta, a]$ est dans I
  - Si $[B \rightarrow \gamma] \in G$

- Alors ajouter $[B \rightarrow .\gamma, b] ou \quad b \in Premier(\beta a)$
Ensuite, la table LR(1) se construit de la meme manière

Table LALR(1)

On fusionne les états similaires a contexte près, si ils ne provoquent pas de conflits lors de la fusion

Trivia (Mais important quand même hein)

- Le mot vide n'est pas une unité lexicale, on ne la lit pas sous peine de se faire défoncer