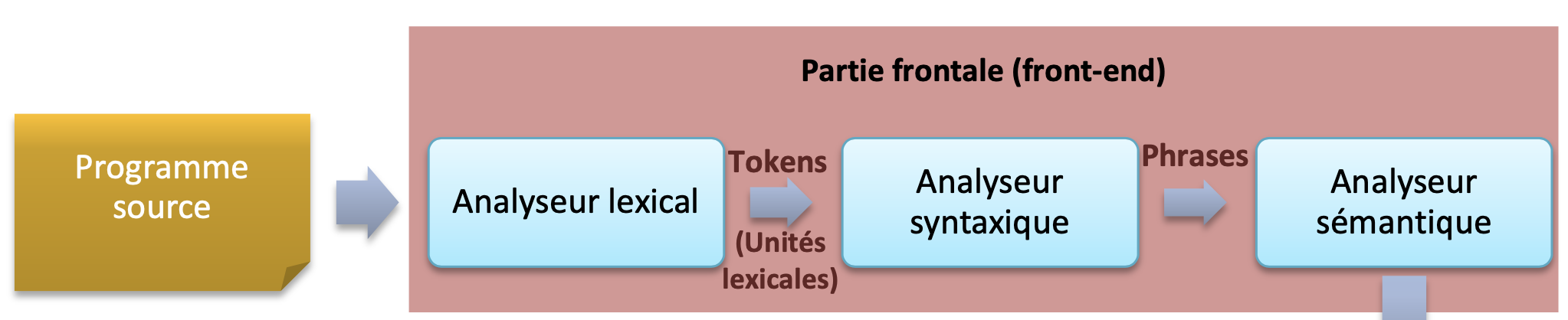
# TRAD

**Analyse sémantique** : La phrase a-t-elle du sens ? Par exemple : une chaîne de caractère dans un entier

Au minimum 2 passes de lecture du texte

Différentes étapes à partir du texte source :

* **Analyse lexicale** (s’intéresse à la **construction des mots**) : recherche des mots clés, les identifiants (noms de variables, de fonctions), les valeurs (nombres, chiffres)
* **Analyse syntaxique** (s’intéresse à la **construction des phrases**)
* **Analyse sémantique** : (ensemble de règle autorisées où non) vérifie par exemple si une variable n’est pas déclarée deux fois ou si on met bien un entier dans un int.



(Cf. Schéma du cours partie 1)

Token : un élément : un identifiant, une valeur, un mot clé… → Unité lexicale

Phrases : Arbre syntaxique : tous les éléments

Arbre abstrait est créé à la sortie de l’analyseur syntaxique et on en a besoin à l’entrée dans l’analyse sémantique : dans cet arbre on retire tous les éléments superflus.

**Table Des Symboles** (**TDS**) : table dans laquelle on y retrouve tous les symboles, c’est-à-dire :

* les identifiants (regroupe les autres)
  + les variables
  + les types
  + les fonctions
  + les classes
  + les paramètres et arguments
  + les noms de tableaux (différent d’une variable)

La TDS va permettre de savoir si une variable a déjà été déclarée, de définir les naming spaces et le scope des variables en créant une arborescence de différentes TDS.

Partie 1 : Analyseur Lexical

Va remplacer tous les éléments communs (+, -, ...) par des codes entiers afin d’accélérer le processus. A la sortie, on a plus que des codes de chiffres.

Partie 2 : Analyseur syntaxique

Descendante : partir de l'axiome et descendre jusqu’à obtenir le résultat

Ascendante : on part du mot que l’on est en train de lire est on essaye de trouver où le mot intervient dans les règles, on remonte dans toutes les règles, on cherche la partie gauche dans les autres parties droites et ainsi de suite.

Les points représentent des arbres binaires (2.3) et la virgule représente une liste (2,3).

Exemples :

(1,(2.2), 3) nil 23 ((2.3),(2,(3.4),nil)) (1) (2,3,4,5,6) ((2,3).4)

A’ → A

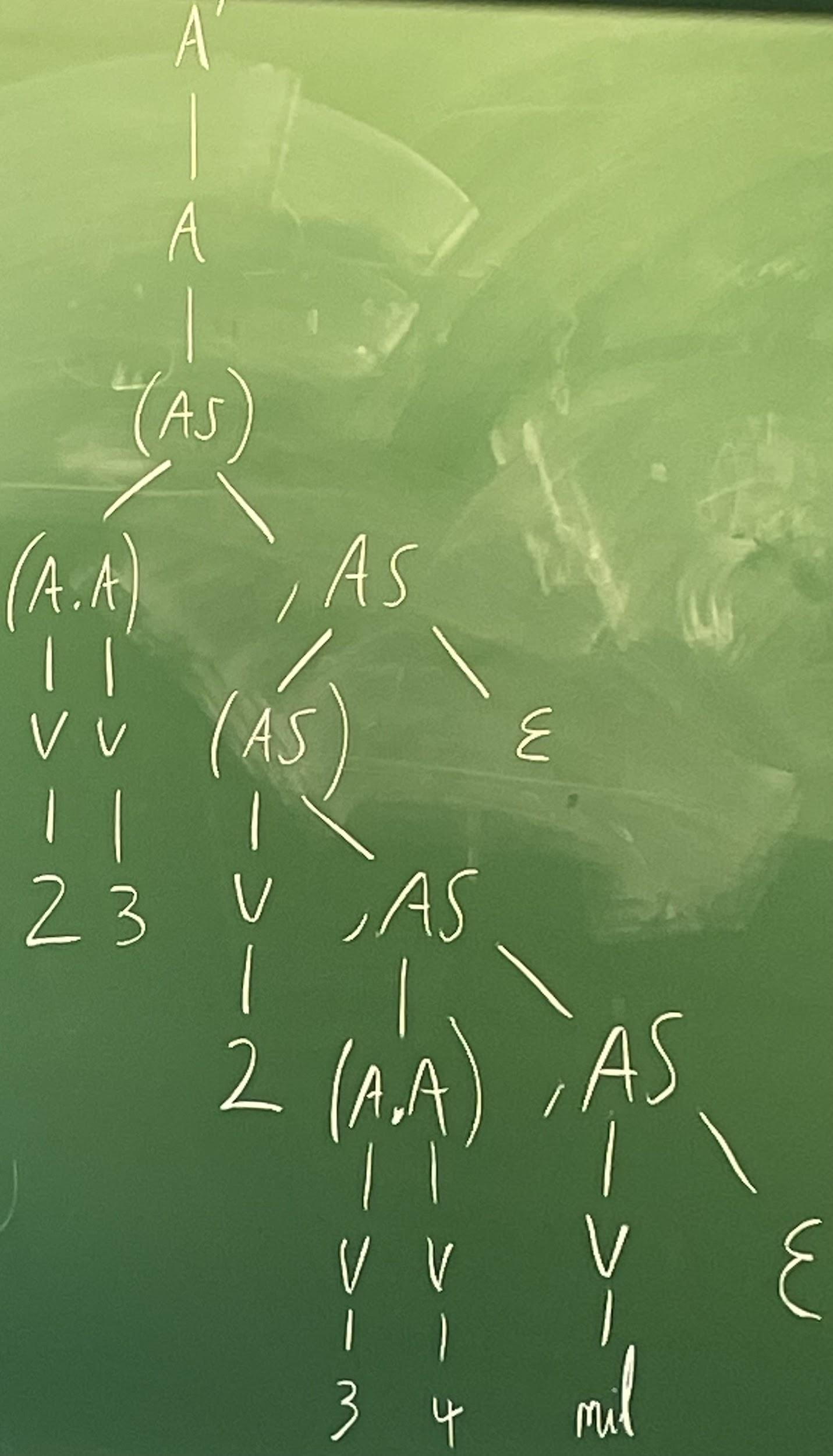
A → ( AS ) | ( A . A ) | V

S → , AS | Ɛ

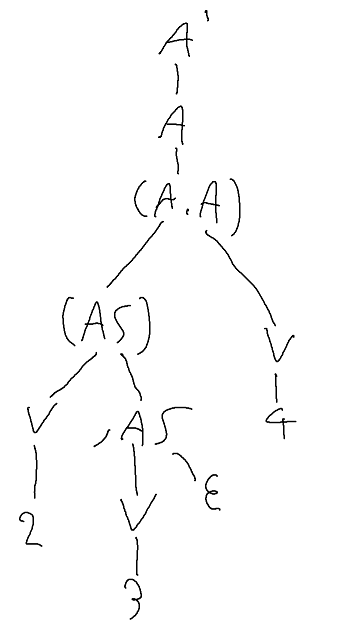
V → INT | ‘nil’

Avec la grammaire citée dans l’exemple, il est impossible de construire un analyseur LL1 (car en regardant le premier caractère on ne sait pas quoi choisir (à cause de la parenthèse par exemple)). Il est cependant possible de construire un analyseur LL3 (3 caractères d’avance).

**Arbres SYNTAXIQUES :**

((2.3),(2,(3.4),nil)) → 

((2,3).4) →



LR(0) machine à état qui va nous permettre de savoir où nous en sommes dans l’analyse.

Une règle comportant un index de lecture devient un **item**. (Page 10 du cours)

Exemple : A’ → A (c’est une règle) et A’ → .A (ici on a le point en indice de lecture, c’est donc un item)

B → \*A. : ici on a “lu” \* et on a “reconnu” A, car A est un non terminal et ne peut pas être lu, simplement reconnu.

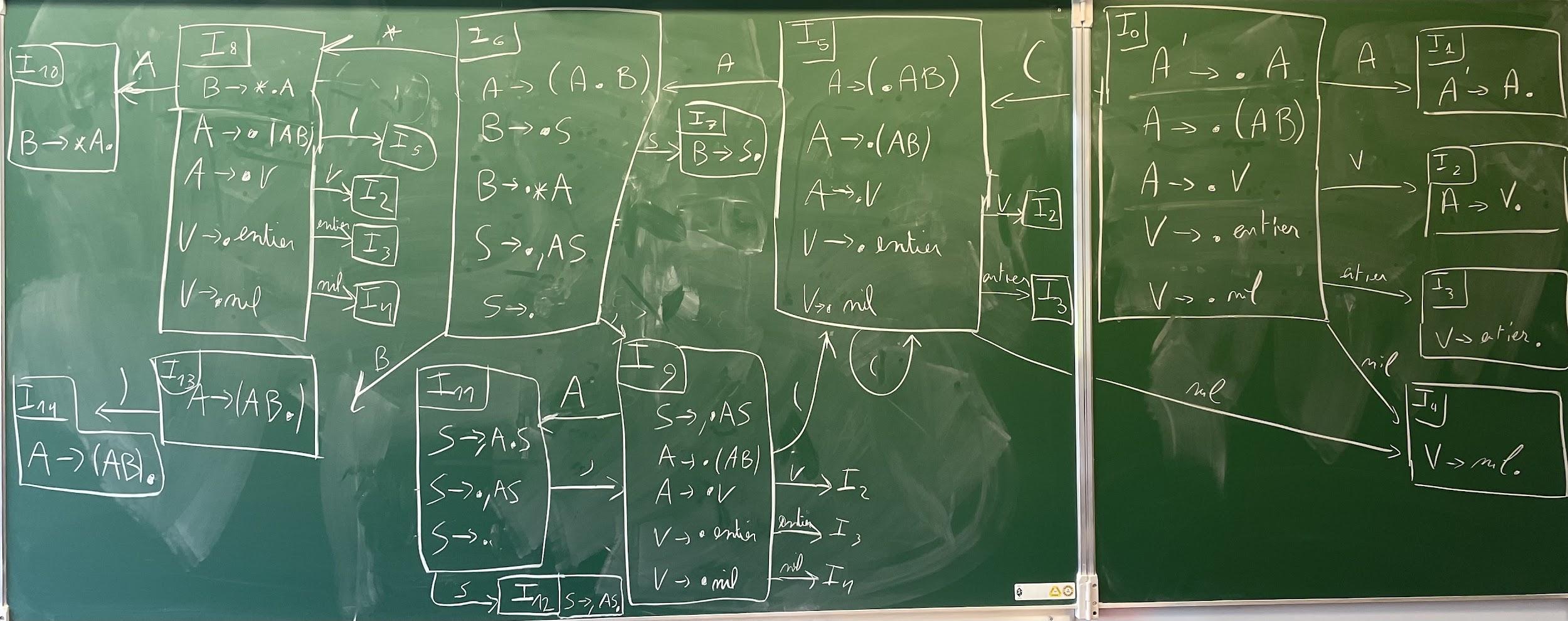
C’est un item terminal car on a reconnu tous les éléments à droite de la règle.

A’ → .A : être devant “A”, c’est être devant A → .(AB) et aussi A → .V et aussi V → .INT et V → .nil

Maintenant devant un INT ou nil, il sait le lire, car c’est des terminaux. (les terminaux sont tous ceux qui peuvent être lus).

C’est la fermeture de l’item dans l’état.

Automate à états LR0



Grammaire :

A’ → A : r0

A → (AB) : r1

A → V : r2

B → S : r3

B → \*A : r4

S → ,AS : r5

S → Ɛ : r6

V → entier : r7

V → nil : r8

TUN : Terminaux Union Non-Terminaux

d5 : décalage vers I5 (exemple : “,AS” : si tu as “,” c’est un décalage)

5 : est une transition et non un décalage

r : pour réduction car on est dans un item terminal (r7 = règle 7 de la grammaire)

Les trous c’est les cas d’erreur

Table d’actions et Table de transitions

I6 : comme on a un état terminal (vide) : tout se transforme en état terminal (conflit)

**Table LR0**

| I\Tun | ( | \* | , | ) | entier | nil | $ | A | V | S | B |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I0 | d5 |  |  |  | d3 | d4 |  | 1 | 2 |  |  |
| I1 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |  |  |  |  |
| I2 | r2 | r2 | r2 | r2 | r2 | r2 | r2 |  |  |  |  |
| I3 | r7 | r7 | r7 | r7 | r7 | r7 | r7 |  |  |  |  |
| I4 | r8 | r8 | r8 | r8 | r8 | r8 | r8 |  |  |  |  |
| I5 | d5 |  |  |  | d3 | d4 |  | 6 | 2 |  |  |
| I6 | ~~r6~~ | d8~~/r6~~ | d9~~/r6~~ | r6 | ~~r6~~ | ~~r6~~ | ~~r6~~ |  |  | 7 | 13 |
| I7 | r3 | r3 | r3 | r3 | r3 | r3 | r3 |  |  |  |  |
| I8 | d5 |  |  |  | d3 | d4 |  | 10 | 2 |  |  |
| I9 | d5 |  |  |  | d3 | d4 |  | 11 | 2 |  |  |
| I10 | r4 | r4 | r4 | r4 | r4 | r4 | r4 |  |  |  |  |
| I11 |  |  | d9 |  |  |  |  |  |  | 12 |  |
| I12 | r5 | r5 | r5 | r5 | r5 | r5 | r5 |  |  |  |  |
| I13 |  |  |  | d14 |  |  |  |  |  |  |  |
| I14 | r1 | r1 | r1 | r1 | r1 | r1 | r1 |  |  |  |  |

Quand dans un état il y a un item terminal, on fait une réduction sur toute la ligne (Exemple I6).

2 actions dans la même case sur la ligne I6 : c’est un conflit

Suivant :

Pour les suivants de A on cherche où se situent les A dans les parties droites de la grammaire.

Suivant(A) : { \* , ) }

* Premier(B)
  + { \* , }
* Suivant(B)
  + { ) }
* Premier(S)
  + { , }

Suivant(B) : { ) }

* { ) }

Suivant(S) : { ) }

* Suivant(B) → { ) }

Suivant(V) : { \* , ) }

* Suivant(A)

L’analyseur SLR1 ne fera la réduction que sur les suivants.

La réduction pour I6 se fera donc uniquement sur les suivants de S soit “ ( “. Donc r6 uniquement dans la case “ ) “. Cela devient une table SLR1.

**Table SLR1**

Table d’actions et Table de transitions

| I\Tun | ( | \* | , | ) | entier | nil | $ | A | V | S | B |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I0 | d5 |  |  |  | d3 | d4 |  | 1 | 2 |  |  |
| I1 |  |  |  |  |  |  | r0 |  |  |  |  |
| I2 |  | r2 | r2 | r2 |  |  | r2 |  |  |  |  |
| I3 |  | r7 | r7 | r7 |  |  | r7 |  |  |  |  |
| I4 |  | r8 | r8 | r8 |  |  | r8 |  |  |  |  |
| I5 | d5 |  |  |  | d3 | d4 |  | 6 | 2 |  |  |
| I6 |  | d8 | d9 | r6 |  |  |  |  |  | 7 | 13 |
| I7 |  |  |  | r3 |  |  |  |  |  |  |  |
| I8 | d5 |  |  |  | d3 | d4 |  | 10 | 2 |  |  |
| I9 | d5 |  |  |  | d3 | d4 |  | 11 | 2 |  |  |
| I10 |  |  |  | r4 |  |  |  |  |  |  |  |
| I11 |  |  | d9 |  |  |  |  |  |  | 12 |  |
| I12 |  |  |  | r5 |  |  |  |  |  |  |  |
| I13 |  |  |  | d14 |  |  |  |  |  |  |  |
| I14 |  | r1 | r1 | r1 |  |  | r1 |  |  |  |  |

Nouvelle grammaire :

r0 : S’ → S

r1 : S → G = D

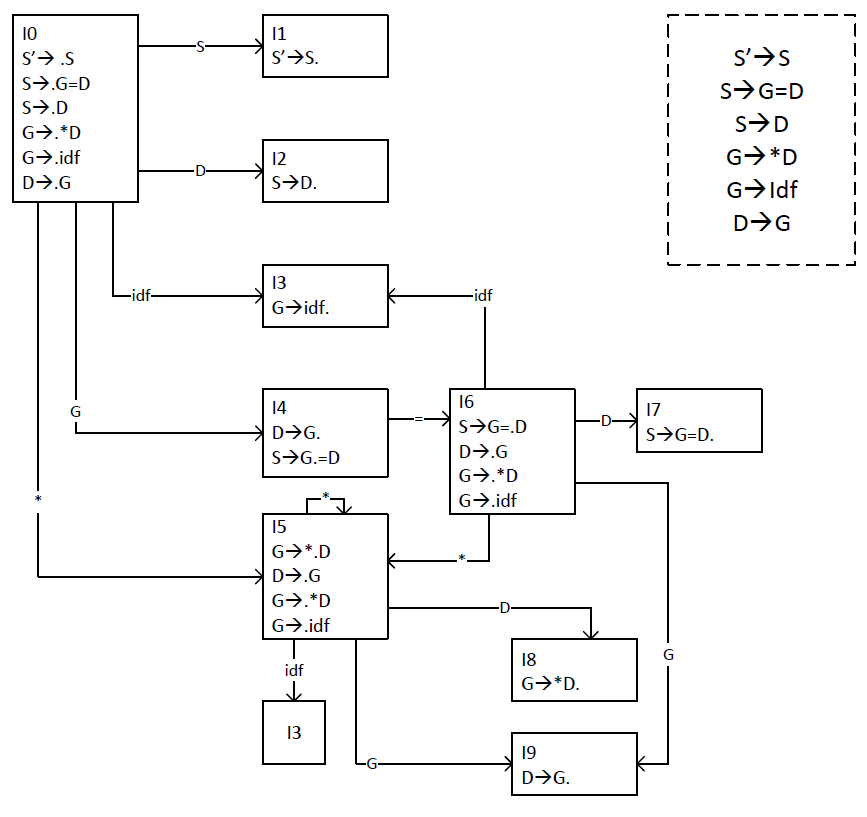
r2 : S → D

r3 : G → \*D

r4 : G → idf

r5 : D → G

Automate :



|  | S | G | D | S’ |
| --- | --- | --- | --- | --- |
| Suivant | $ | = $ | = $ | $ |

Table SLR1 :

|  | idf | = | \* | $ | S | D | G |
| --- | --- | --- | --- | --- | --- | --- | --- |
| I0 | d3 |  | d5 |  | 1 | 2 | 4 |
| I1 |  |  |  | r0 |  |  |  |
| I2 |  |  |  | r2 |  |  |  |
| I3 |  | r4 |  | r4 |  |  |  |
| I4 |  | d6/r5 |  | r5 |  |  |  |
| I5 | d3 |  | d5 |  |  | 8 | 2 |
| I6 | d3 |  | d5 |  |  | 7 | 9 |
| I7 |  |  |  | r1 |  |  |  |
| I8 |  | r3 |  | r3 |  |  |  |
| I9 |  | r5 |  | r5 |  |  |  |

Conflit en (I4, =) → on va pouvoir le résoudre en regardant les suivants généralisés ou régionalisés.

Grammaire :

A’ → A

A → (AB)

A → V  
B → S  
B → \*A

S → ,AS

S → ^ (équivaut à epsilon)  
V → entier

V → nil

Analyser les contextes :

Tous les items ont forcément un contexte.

On part de l’axiome :

Automate LR1 :

I0 (Axiome)

A’ → .A {$} (le **contexte** : le suivant de la partie gauche de la production : A’)

A → .(AB) {$}

A → .V {$}

V → .entier {$}

V → .nil {$}

I1 = AV(I0, entier)

V → entier. {$}

I2 = AV(I0, nil)

V → nil. {$}

I3  = AV(I0, V)

A → V. {$}

I4 = AV(I0, A)

A’ → A. {$}

I5 = AV(I0, “(“ )

A → (.AB) {$}

A → .(AB) {\* , )}

A → .V {\* , )}

V → entier {\* , )}

V → nil {\* , )}

I6 = AV(I5, nil)

V → nil. {\* , )}

I7 = AV(I5, entier)

V → entier. {\* , )}

I8 = AV(I5, V)

A → V. {\* , )}

I9 = AV(I5, ( )

A → (.AB) {\* , )}

A → .(AB) {\* , )}

A → .V {\* , )}

V → .entier {\* , )}

V → .nil {\* , )}

I10 = AV(I5, A)

A → (A.B) {$}

B → .S {)}

B → .\*A {)}

S → .,AS {)}

S → . {)}

I6 = AV(I9, nil)

I7 = AV(I9, entier)

I8 = AV(I9, V)

I9 = AV(I9, ()

I11 = AV(I9, A)

A → (A.B) {\* , )}

B → .S {)}

B → .\*A {)}

S → .,AS {)}

S → . {)}

I12 = AV(I10, B)

A → (AB.) {$}

I13 = AV(I10, S)

B → S. {)}

I14 = AV(I10, ,)

S → ,.AS {)}

A → .(AB) {, )}

A → .V {, )}

V → .entier {, )}

V → .nil {, )}

I15 = AV(I10, \*)

B → \*.A {)}

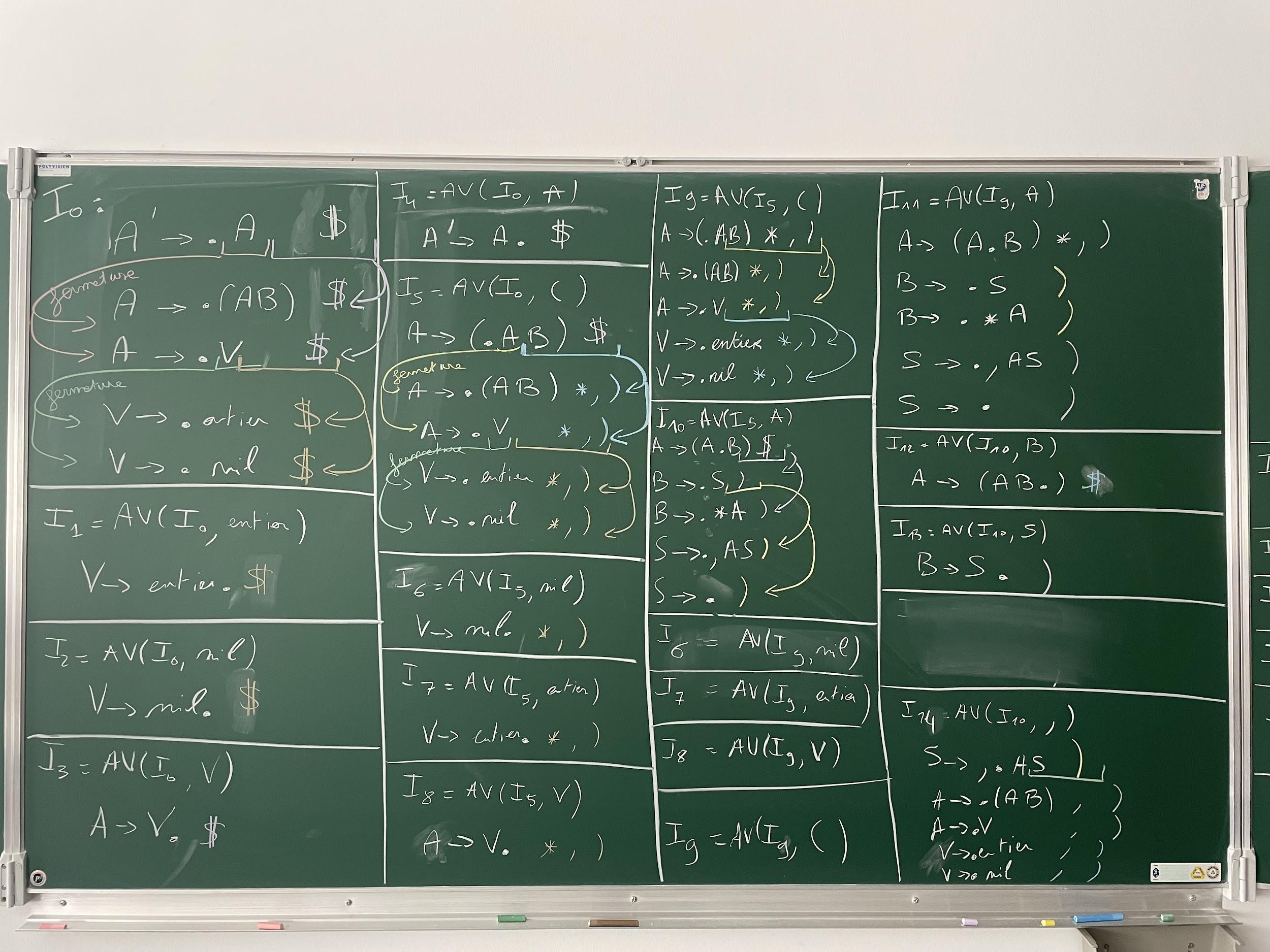
A → .(AB) {)}

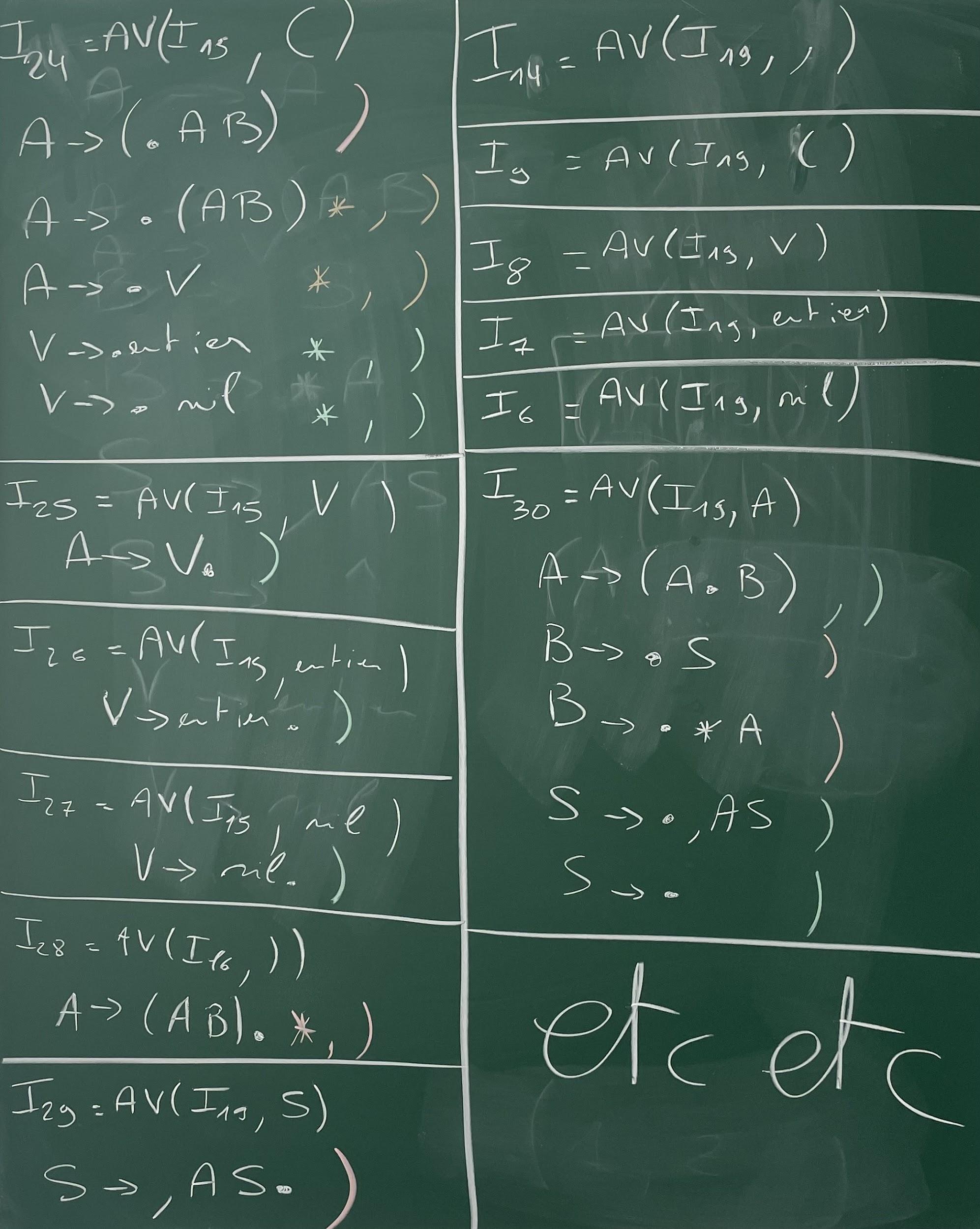
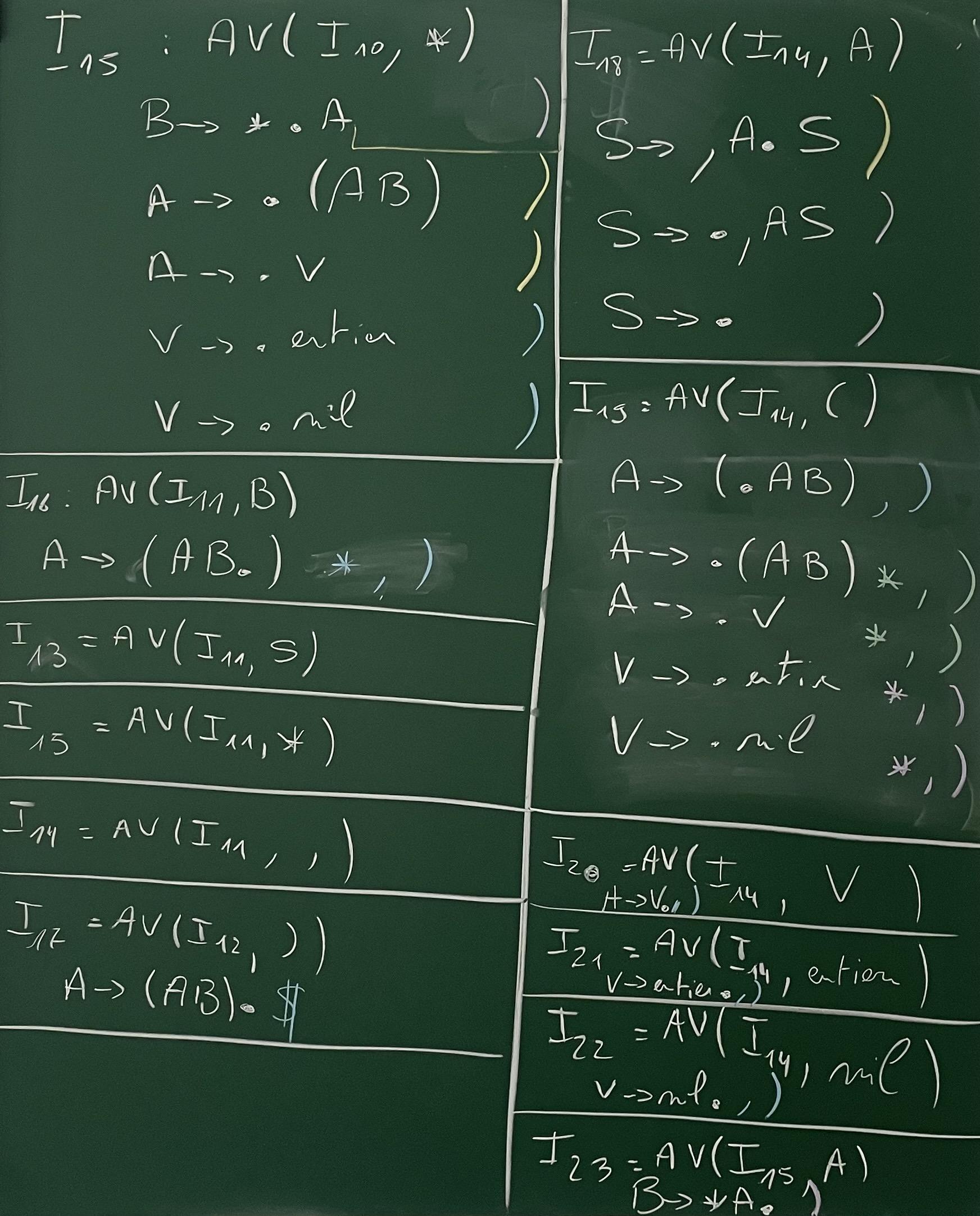
A → .V {)}

V → .entier {)}

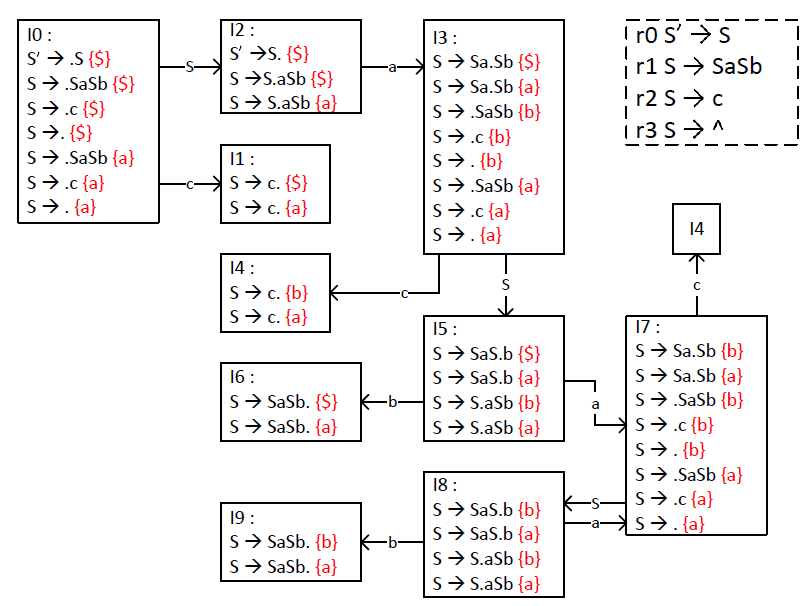
V → .nil {)}

## Automate LR1 : Collection des états LR1





////----////----////----////----////----////----////----////----////----////



| LR(1) | a | b | c | $ | S |
| --- | --- | --- | --- | --- | --- |
| I0 | r3 |  | d1 | r3 | 2 |
| I1 | r2 |  |  | r2 |  |
| I2 | d3 |  |  | r0 |  |
| I3 | r3 | r3 | d4 |  | 5 |
| I4 | r2 | r2 |  |  |  |
| I5 | d7 | d6 |  |  |  |
| I6 | r1 |  |  | r1 |  |
| I7 | r3 | r3 | d4 |  | 8 |
| I8 | d7 | d9 |  |  |  |
| I9 | r1 | r1 |  |  |  |

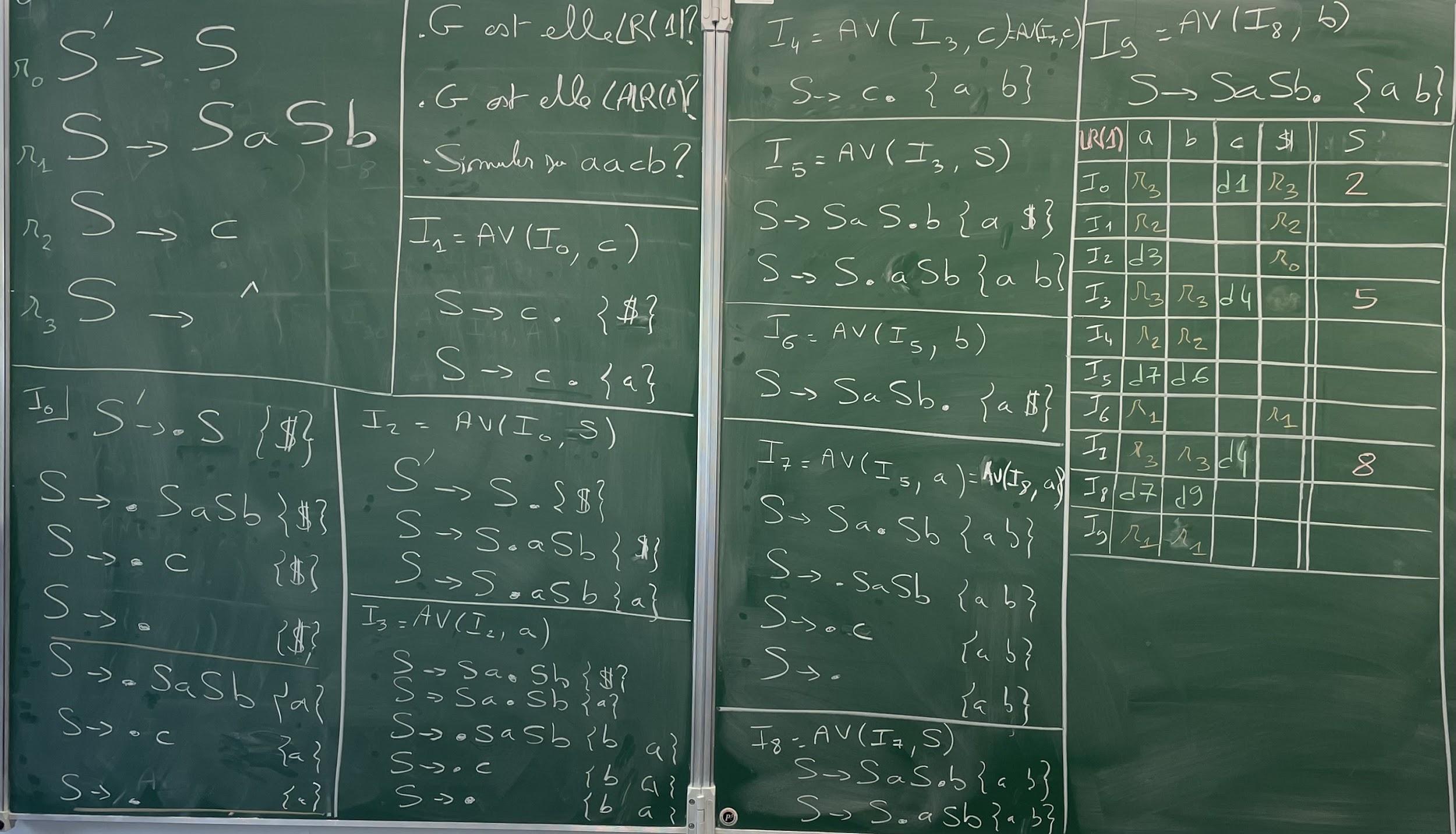
r = réduction (quand item terminal)

d = décalage

Dans “S” on indique la case dans laquelle on vas quand on lis S

LR0 on regarde rien, LR1 on regarde le contexte (le suivant)

**Exercice type :**



**LALR1 :**

Etat avec même noyau = état avec les mêmes items mais avec des contextes différents.

(Identiques au contexte près) (rappel : noyau ex : S→ .S, contexte ex : {$})

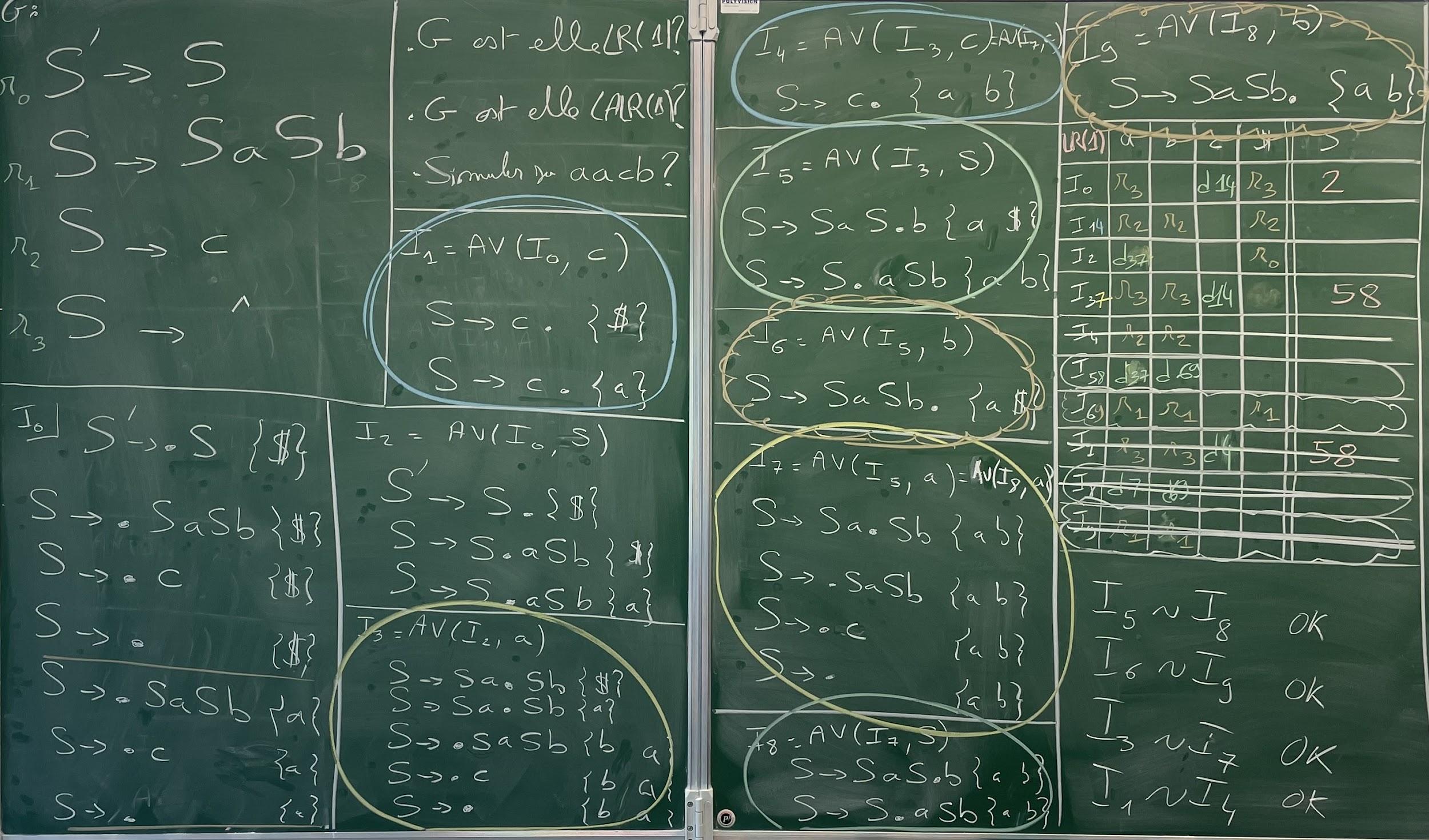
On va fusionner ces états.

On va réussir à passer d’un analyseur LR1 à un LALR1 si on arrive à fusionner les états identiques au contexte près sans créer de conflits dans la table.

On dit que 2 états sont équivalents Ex : I5 ~ I8

On regarde donc les 2 lignes de I5 et I8, pour “a” on voit qu’on à “d7” pour les 2 lignes donc pas de souci. Pour “b”, on a “d6” et “d9”, or, on peut regarder si I6 et I9 peuvent fusionner.

Comme I6 et I9 peuvent fusionner, I6 et I9 fusionnent et “d6” et “d9” deviennent “d69”, donc les cases qui contiennent des “d6” ou de “d9” deviennent des “d69”, ainsi, on peut fusionner I5 et I8. (les chiffres également dans les colonnes des non-terminaux peuvent fusionner avec cette règle).



| LALR(1) | a | b | c | $ | S |
| --- | --- | --- | --- | --- | --- |
| I0 | r3 |  | d14 | r3 | 2 |
| I14 | r2 | r2 |  | r2 |  |
| I2 | d37 |  |  | r0 |  |
| I37 | r3 | r3 | d4 |  | 58 |
| I58 | d37 | d69 |  |  |  |
| I69 | r1 | r1 |  | r1 |  |

I5 ~ I8 : OK

I6 ~ I9 : OK

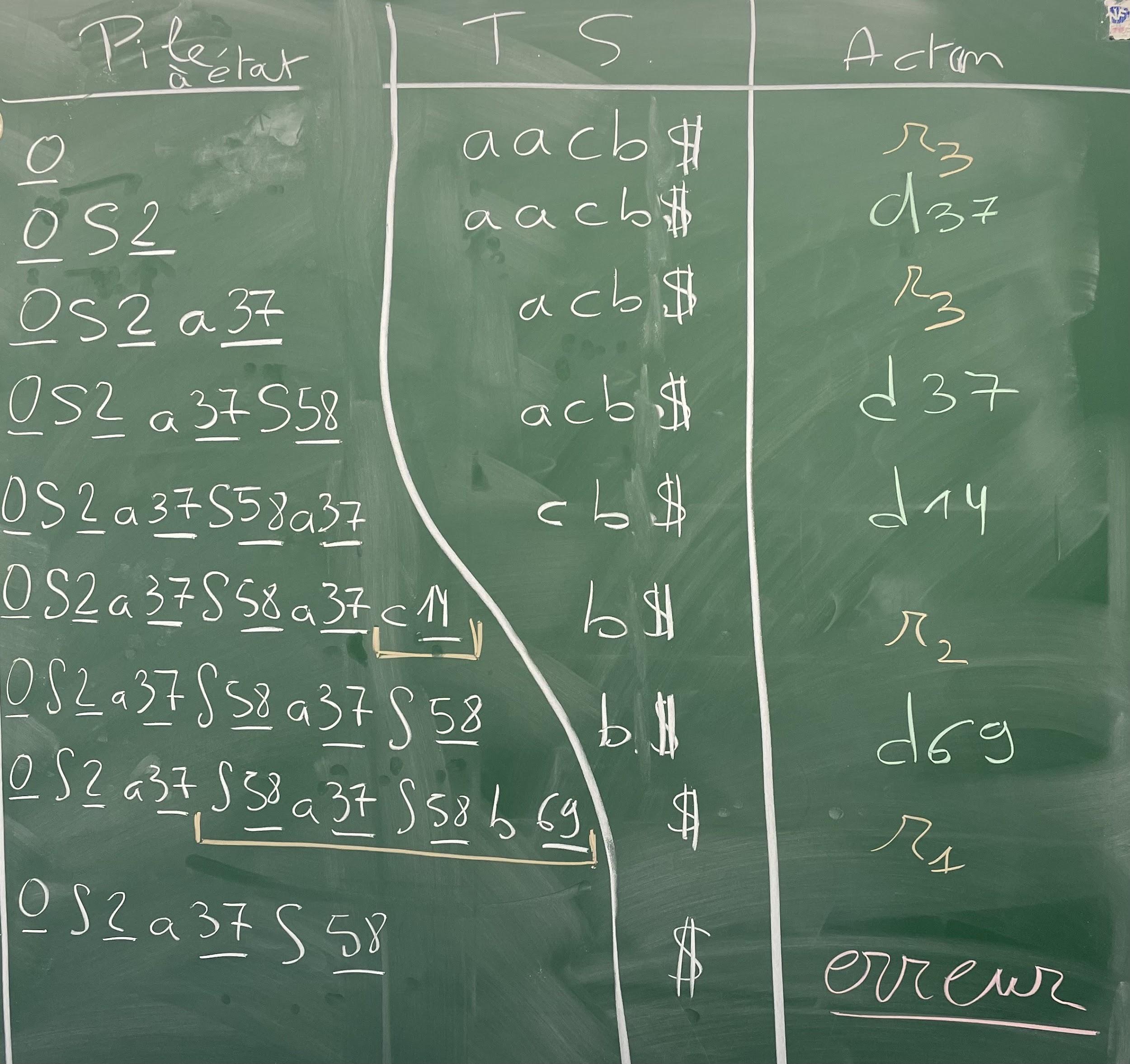
I3 ~ I7 : OK

I1 ~ I4 : OK

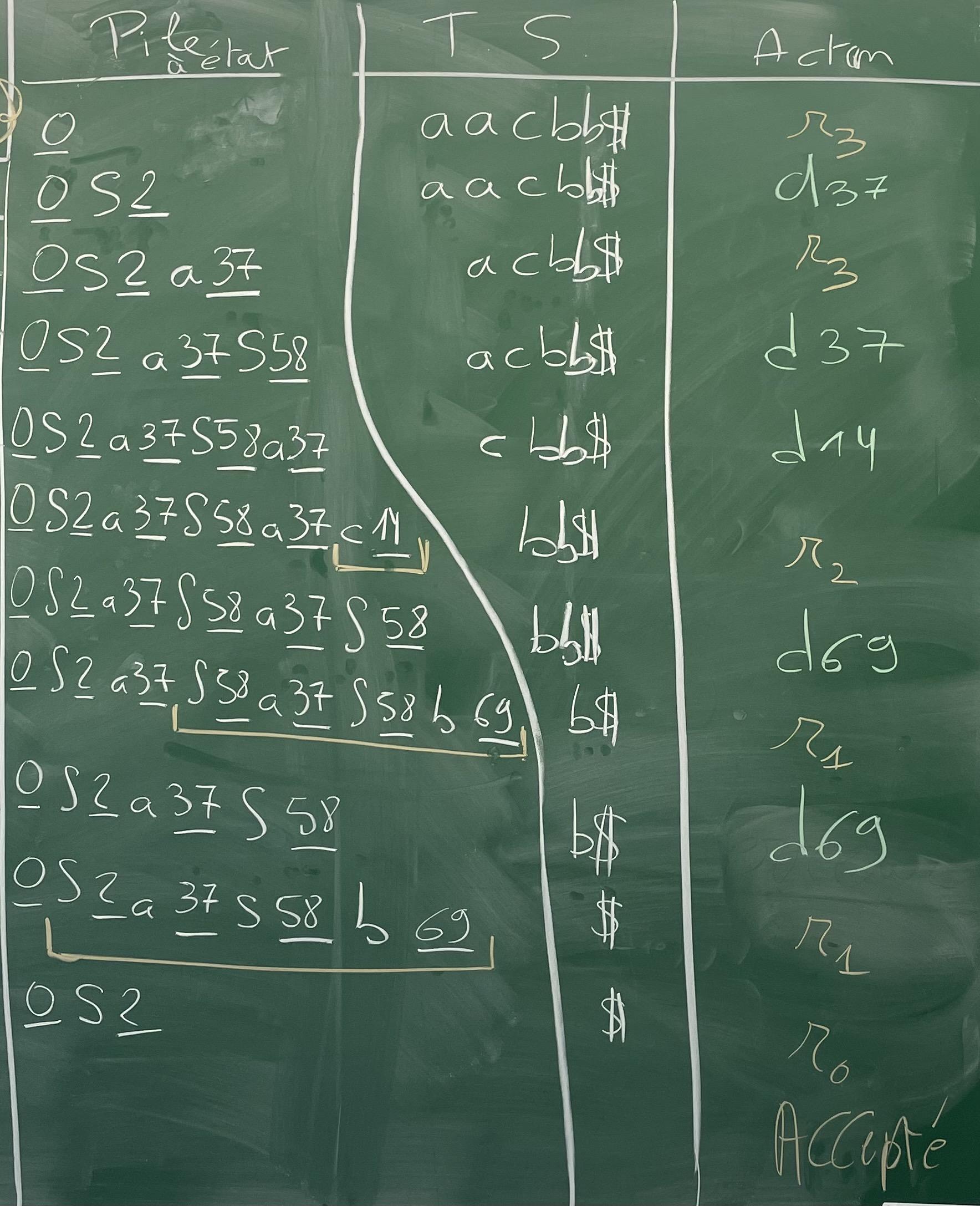
Tous les états fusionnables ont été fusionnés sans conflit, la grammaire est donc LALR1.

Simuler sur aacb ?

| Pile à état | Texte Source (TS) | Action |
| --- | --- | --- |
| I0  I0 S I2  I0 S I2 a I37  I0 S I2 a I37 S I58  I0 S I2 a I37 S I58 a I37  I0 S I2 a I37 S I58 a I37 c I14  I0 S I2 a I37 S I58 a I37 c I14 S I58  I0 S I2 a I37 S I58 a I37 S I58 b d69  I0 S I2 a I37 S I58  I0 S I2 a I37 S I58 b I69  I0 S I2 | .aacbb$  .aacbb$  a.acbb$  a.acbb$  aa.cbb$  aac.bb$  aac.bb$  aacb.b$  aacb.b$  aacbb.$  aacbb.$ | r3  d37  r3  d37  d14  r2  d69  r1  d69  r1  r0  OK |



En ajoutant un “b” à la fin du mot, on voit que le mot est accepté.



Une fois que l’on a effectué l’action “r0”, c’est-à-dire l’axiome, le mot est accepté et est donc reconnu par la grammaire.