Exercice 1

- On s'intéresse à une grammaire GI de calcul d'itinéraire suivante:
  - G = <T,NT, {ROUTE}, P> avec
  - T = { go, tg, td, pan },
  - NT = {ROUTE, INST, PANNEAU , TOURNE}
  - Les règles P
    - ✓ ROUTE→ INST| INST ROUTE
    - ✓ INST→ go| PANNEAU TOURNE
    - ✓ TOURNE→ tg| td
    - ✓ PANNEAU→ ε | pan

1. Cette grammaire n'est pas LL(1) : pourquoi?
2. Donner une grammaire G' équivalente à G et qui vous semble LL(1).
3. Calculer les ensembles Premier et Suivant pour G'.
4. Donner la table d'analyse LL(1) de G'

Solution

G

- ✓ ROUTE→ INST| INST ROUTE
- ✓ INST→ go| PANNEAU TOURNE
- ✓ TOURNE→ tg| td
- ✓ PANNEAU→ ε | pan

1. Cette grammaire n'est pas LL(1) : pourquoi?

Premier(INST) = premier (INST ROUTE)

N'est pas une grammaire LL1

2. G' équivalente à G

- ✓ ROUTE → INST ROUTE'
- ✓ ROUTE' → ε | ROUTE
- ✓ INST→ go| PANNEAU TOURNE
- ✓ TOURNE→ tg| td
- ✓ PANNEAU→ ε | pan

Factorisation à gauche

Solution

G’

- ✓ ROUTE → INST ROUTE’
- ✓ ROUTE’ → ε | ROUTE
- ✓ INST→ go| PANNEAU TOURNE
- ✓ TOURNE→ tg| td
- ✓ PANNEAU→ ε | pan

Premier (PANNEAU) = **{ ε, pan }**

Premier (TOURNE) = **{tg , td}**

Premier (INST) = {go} U Premier(PANNEAU) U Premier (TOURNE si PANNEAU = ε)
= **{go , pan , tg , td }**

Premier (ROUTE) = Premier(INST) =**{go , pan , tg , td }**

Premier (ROUTE’) = { ε } U Premier (ROUTE) = **{go , pan , tg , td , ε }**

Suivant (PANNEAU) = Premier (TOURNE) = **{tg , td}**
Suivant (ROUTE) = **{ $ }**
Suivant (INST) = **{ go , pan , tg , td }** U Suivant (ROUTE si ROUTE’= ε) = **{ go , pan , tg , td , $ }**
Suivant (ROUTE’) = Suivant (ROUTE) = = **{ $ }**
Suivant (TOURNE) = Suivant (INST) = **{ go , pan , tg , td , $ }**

	First	Follow
ROUTE	{go , pan , tg , td }	{ $ }
ROUTE’	{go , pan , tg , td , ε }	{ $ }
INST	{go , pan , tg , td }	{go , pan , tg , td , $ }
TOURNE	{tg , td}	{ go , pan , tg , td , $
PANNEAU	{ ε, pan }	{tg , td}

- ✓ ROUTE → INST ROUTE’
- ✓ ROUTE’ → ε | ROUTE
- ✓ INST→ go| PANNEAU TOURNE
- ✓ TOURNE→ tg| td
- ✓ PANNEAU→ ε | pan

❑ Table d’analyse

	tg	**td**	**go**	**pan**	**$**
ROUTE	ROUTE → INST ROUTE’	ROUTE → INST ROUTE’	ROUTE → INST ROUTE’	ROUTE → INST ROUTE’	
ROUTE’	ROUTE’ → ROUTE	ROUTE’ → ROUTE	ROUTE’ → ROUTE	ROUTE’ → ROUTE	ROUTE’ → ε
INST	INST→ PANNEAU TOURNE	INST→ PANNEAU TOURNE	INST→ go	INST→ PANNEAU TOURNE	
TOURNE	TOURNE→ tg	TOURNE→ td			
PANNEAU	PANNEAU→ ε	PANNEAU→ ε		PANNEAU→ pan	

Exercice 2

- Soit la grammaire G suivante:
  - ✓E → E ou T | T
  - ✓T → T et F | F
  - ✓F → non F | (E) | vrai | faux

1. La grammaire est-elle LL(1) ?
2. Supprimer la récursivité gauche.
3. Calculer les ensembles First et Follow des symboles variables de la nouvelle grammaire.
4. Donner la table d'analyse LL(1) de la nouvelle grammaire.
5. Donner la pile d'analyse du mot "vrai et (faux ou vrai)", et en déduire l'arbre de dérivation pour ce mot

Solution

- ✓E → E ou T | T
- ✓T → T et F | F
- ✓F → non F | (E) | vrai | faux

- ✓E → TE'
- ✓E' → ou TE' | ε
- ✓T → FT'
- ✓T' → et FT' | ε
- ✓F → non F | (E) | vrai | faux

Non récursive a gauche

Solution

- ✓E → TE’
- ✓E’ → ou TE’ | ε
- ✓T → FT’
- ✓T’ → et FT’ | ε
- ✓F → non F | (E) | vrai | faux

First(E) = First(T) =First(F)= **{ non , (, vrai , faux }**

First(E’) = **{ ou , ε}**

First(T’) = **{et , ε }**

Solution

- ✓E → TE’
- ✓E’ → ou TE’ | ε
- ✓T → FT’
- ✓T’ → et FT’ | ε
- ✓F → non F | (E) | vrai | faux

Follow (E) = {) , $ }

Follow (E’) = {) , $ }

Follow (T) = { ou,) , $ }

Follow (T’) = { ou,) , $ }

Follow (F) = { et , ou,) , $ }

Solution

Table d'analyse

	et	ou	Non	vrai	faux	(	)	$
E			E→TE'	E→TE'	E→TE'	E→TE'		
E'		E'→ouTE'					E'→ε	E'→ε
T			T→FT'	T→FT'	T→FT'	T→FT'		
T'	T'→etFT'	T'→ε					T'→ε	T'→ε
F			F→nonF	F→vrai	F→faux	F→(E)		

Vérification de mot : vrai et (faux ou vrai) (sur tableau)

Analyse Syntaxique Ascendante
LR

Analyse syntaxique Ascendante

LR : Left to right – Rightmost derivation

- Cette classe de méthodes **ascendantes** couvre la méthode d'analyse déterministe la plus générale connue applicable aux grammaires non ambiguës.

- Elle présente les avantages suivants :
  - Détection des erreurs de syntaxe le plus tôt possible, en lisant les **terminaux** de gauche à droite.
  - Analyse de toutes les constructions syntaxiques des langages courants.

Analyse syntaxique Ascendante

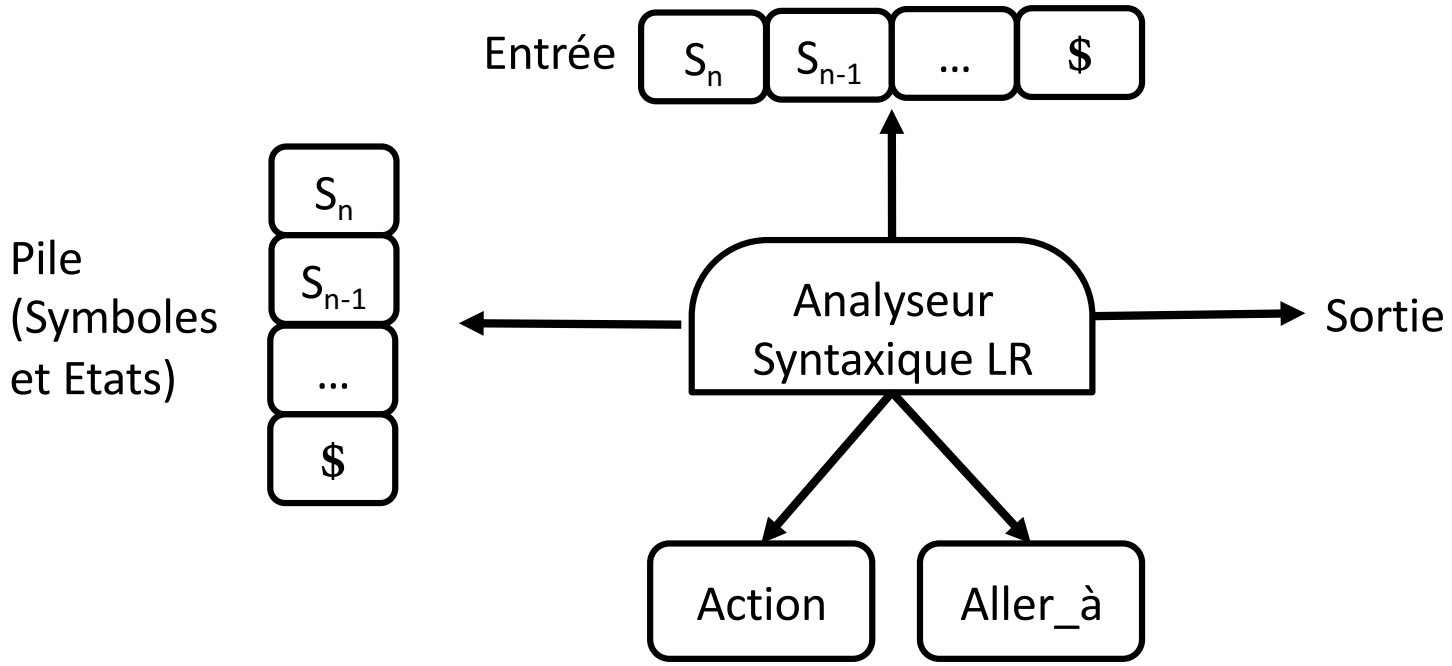
LR : Left to right – Rightmost derivation

- C'est la méthode la plus générale d'analyse syntaxique par décalage-réduction sans retour-arrière.
- Nous pouvons construire des analyseurs **LR** reconnaissant quasiment toutes les constructions des langages.
- La classe des grammaires analysées est un sur-ensemble de la classe des grammaires analysées en **LL**.

Analyse syntaxique Ascendante

LR : Left to right – Rightmost derivation

Architecture générale



Analyse syntaxique Ascendante
LR

- Les méthodes **LR** sont les plus générales, au prix de tables d'analyse volumineuses.
- La méthode **LR** comprend plusieurs cas particuliers, correspondant au même algorithme d'analyse :
  - **SLR** où **S** signifie **simple** : c'est la construction de l'automate **LR** à partir de la grammaire. Transitions données uniquement par la table ALLER_A.
  - **LALR** où **LA** signifie LookAhead **(YACC/Bison)** : ce cas couvre beaucoup de langages, avec une taille de table d'analyse de la même taille que **SLR**.
- Les méthodes **LR** construisent l'**arbre d'analyse** de dérivation en ordre inverse, en partant des feuilles.

Analyse syntaxique Ascendante
SLR

- Une position d'analyse **LR** placé dans le corps de chaque production de la grammaire est schématisée par un point •.

- Ce • indique que nous avons accepté ce qui précède dans la production, et que nous sommes prêts à accepter ce qui suit le point.

Analyse syntaxique Ascendante

LR : Left to right – Rightmost derivation

Exemple

expression ➔ expression • ’’ + ‘’ terme

- L’idée centrale de la méthode **LR** est : Étant donnée une position d’analyse •, nous cherchons à obtenir par **fermeture transitive** toutes les possibilités de continuer l’analyse du texte source, en tenant compte de toutes les productions de la grammaire par décalage ou réduction.

Analyse syntaxique Ascendante

SLR

Décalage – Réduction (shift-reduce)

- Réduction (REDUCE) : remonter la dérivation du handle en chaînage arrière

- Décalage (SHIFT) : Quand la frontière haute ne contient aucun « manche », l'analyseur repousse (décale) la frontière en ajoutant un token à droite de la frontière.

Analyse syntaxique Ascendante

SLR

Décalage – Réduction (shift-reduce)

Exemple

Pile

Flux d'entré

Au départ

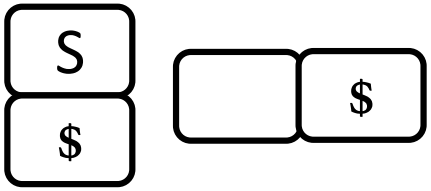
En cours

- On décale de l'entrée vers la pile 0, 1 ou plusieurs symboles jusqu'à ce qu'un manche se trouve en sommet de pile.
- On le remplace par la partie gauche de la production

Fin

- Détection d'une Erreur

ou



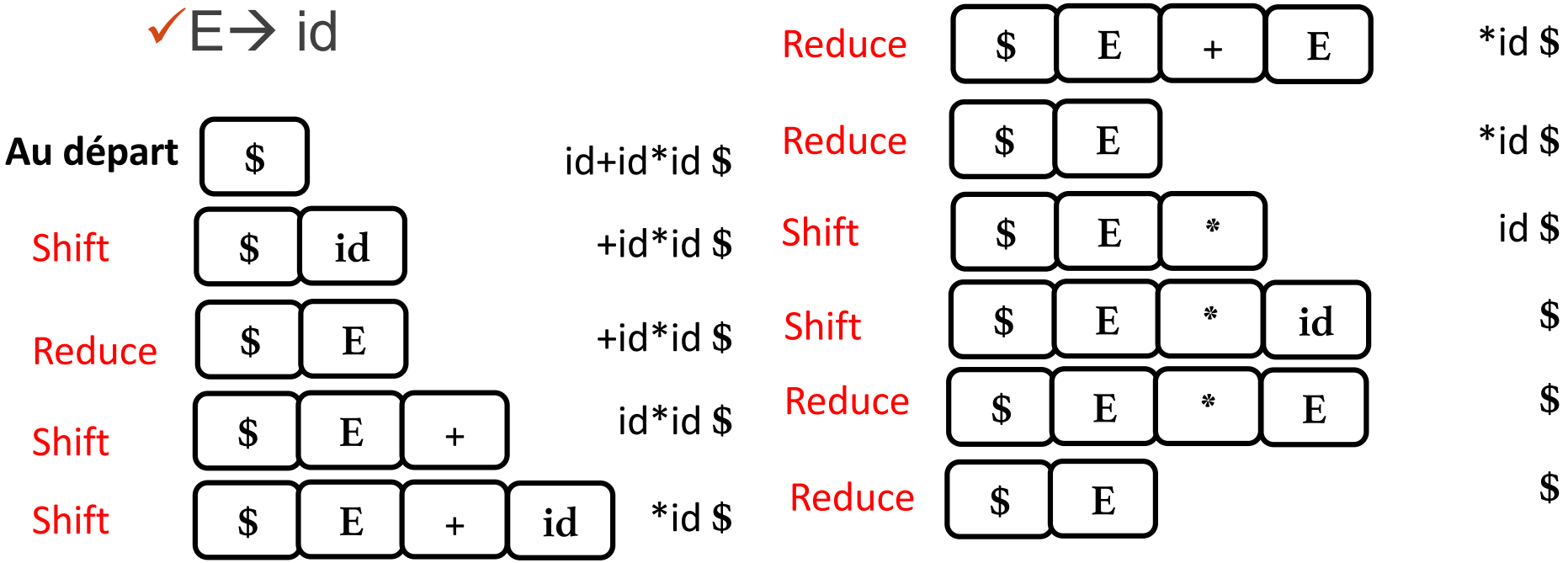
Analyse syntaxique Ascendante

Analyseur LR

Décalage – Réduction (shift-reduce)
Exemple

- ✓E → E+E
- ✓E →E * E
- ✓E → (E)
- ✓E→ id

Flux d'entrée



Analyse syntaxique Ascendante

Fermeture transitive

Définition

- **Transitive** signifie que nous propageons la connaissance que nous avons de la position d'analyse en tenant compte des productions définissant la notion non terminale que nous sommes prêts à accepter.
- **Fermeture** signifie que nous faisons cette propagation de toutes les manières combinatoires possibles, jusqu'à saturation.

Analyse syntaxique Ascendante

LR : Left to right – Rightmost derivation

Exemple

- Une position d'analyse est de la forme:
  - notion ➔ préfixe • non-terminal suffixe
- Sa **fermeture transitive** (transitive closure) se construit suivant toutes les productions définissant la notion **non-terminal** de la forme:
  - **Non-terminal** ➔ corps
- Nous ajoutons à l'état d'analyse le point • pour marquer son début :
  - **Non-terminal** ➔ • corps

Analyse syntaxique Ascendante

Analyse SLR

Exemple

- Soit la grammaire G, avec des productions récursives à gauche suivantes :
  - ✓S→ exp
  - ✓exp →exp + term | term
  - ✓term → term * fact | fact
  - ✓fact → (exp) | Entier

- L'**axiome** S permet d'avoir qu'un seul état accepteur (méthode ascendante, Grammaire Augmentée).

Analyse syntaxique Ascendante

SLR

Exemple

- Au début de l'analyse nous nous trouvons dans la position initiale : noté **Etat_0** dans l'automate **SLR**.
  - ✓S→ • exp
- Nous n'avons encore rien consommé et nous sommes prêt à accepter une exp :
  - ✓exp → • exp + term
  - ✓exp → • term
- D'après les productions de notre grammaire nous sommes dans l'une des positions d'analyse initiales suivantes:
  - ✓term → • term * fact
  - ✓term → • fact
  - ✓fact → • (exp)
  - ✓fact → • Entier

Analyse syntaxique Ascendante

Analyse SLR

Exemple

- De **l'état_0** initial, nous pouvons accepter tout ce qui se trouve à droite du point d'analyse; nous nous retrouvons alors dans un des états suivants :

Etat_1 // accepter exp

S→ exp •

exp → exp • + term

Etat_2 // accepter term

exp → term •

term → term • * fact

Etat_3 // accepter fact

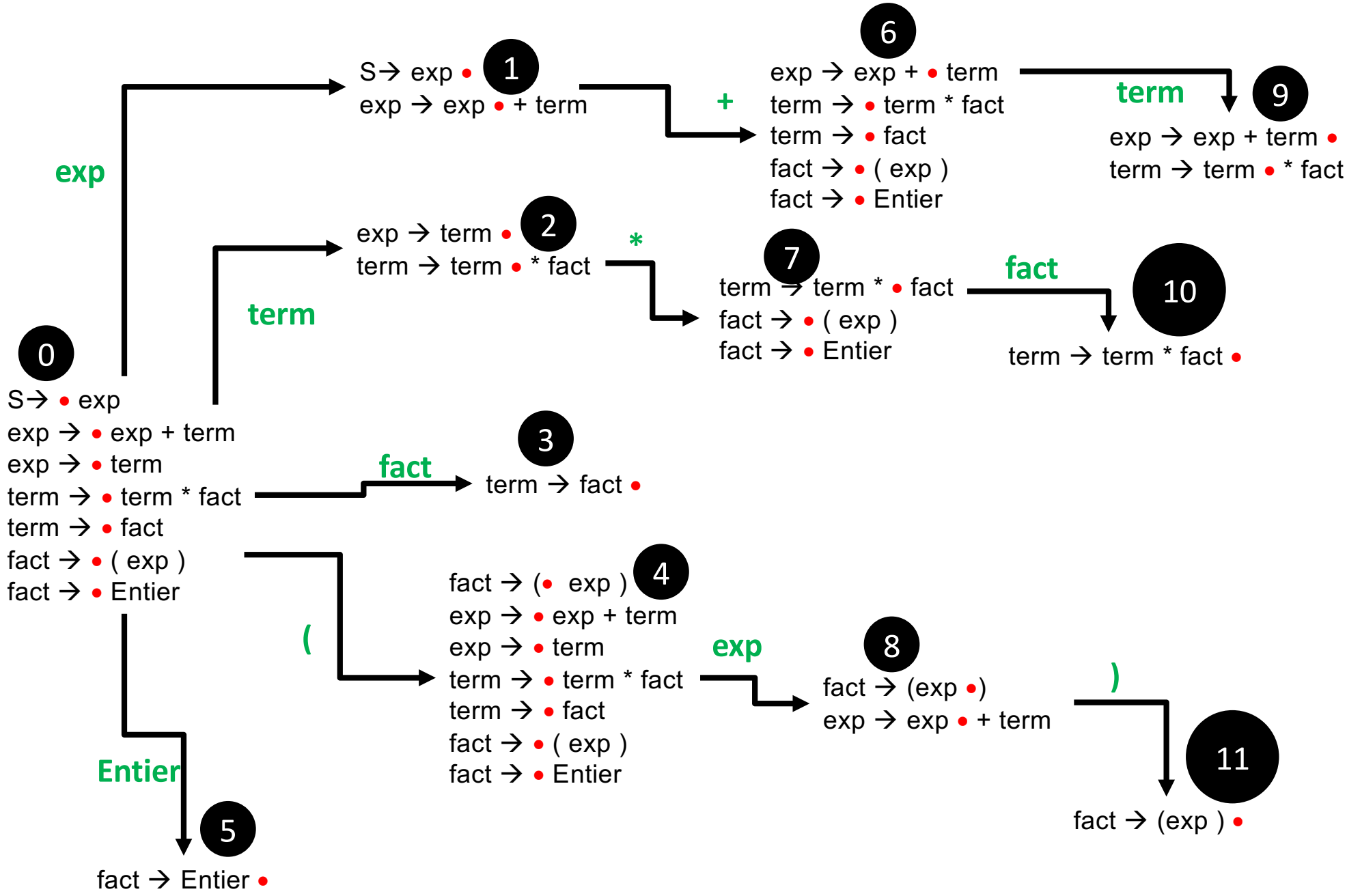
term → fact •

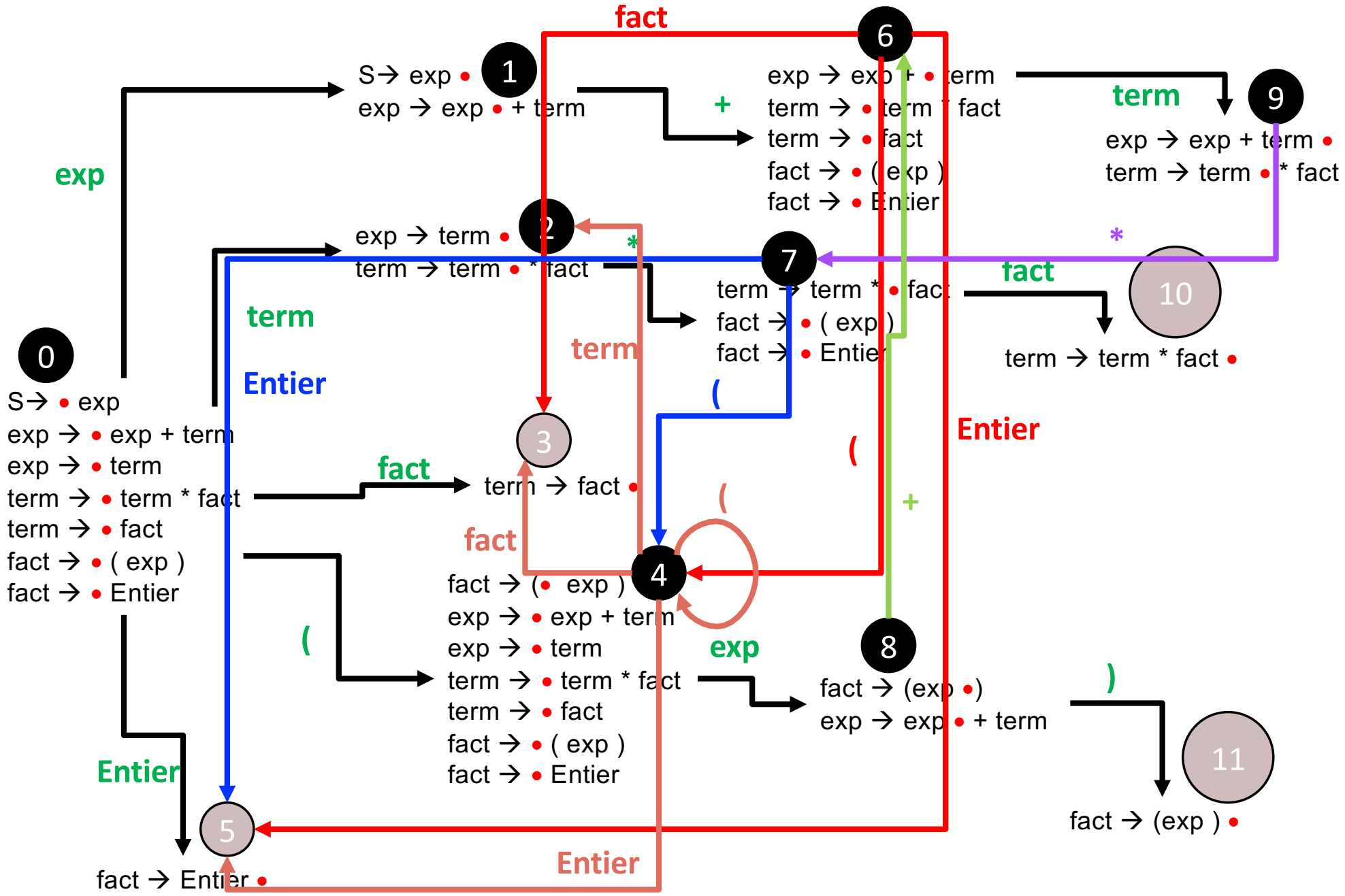
Etat_4 // accepter (

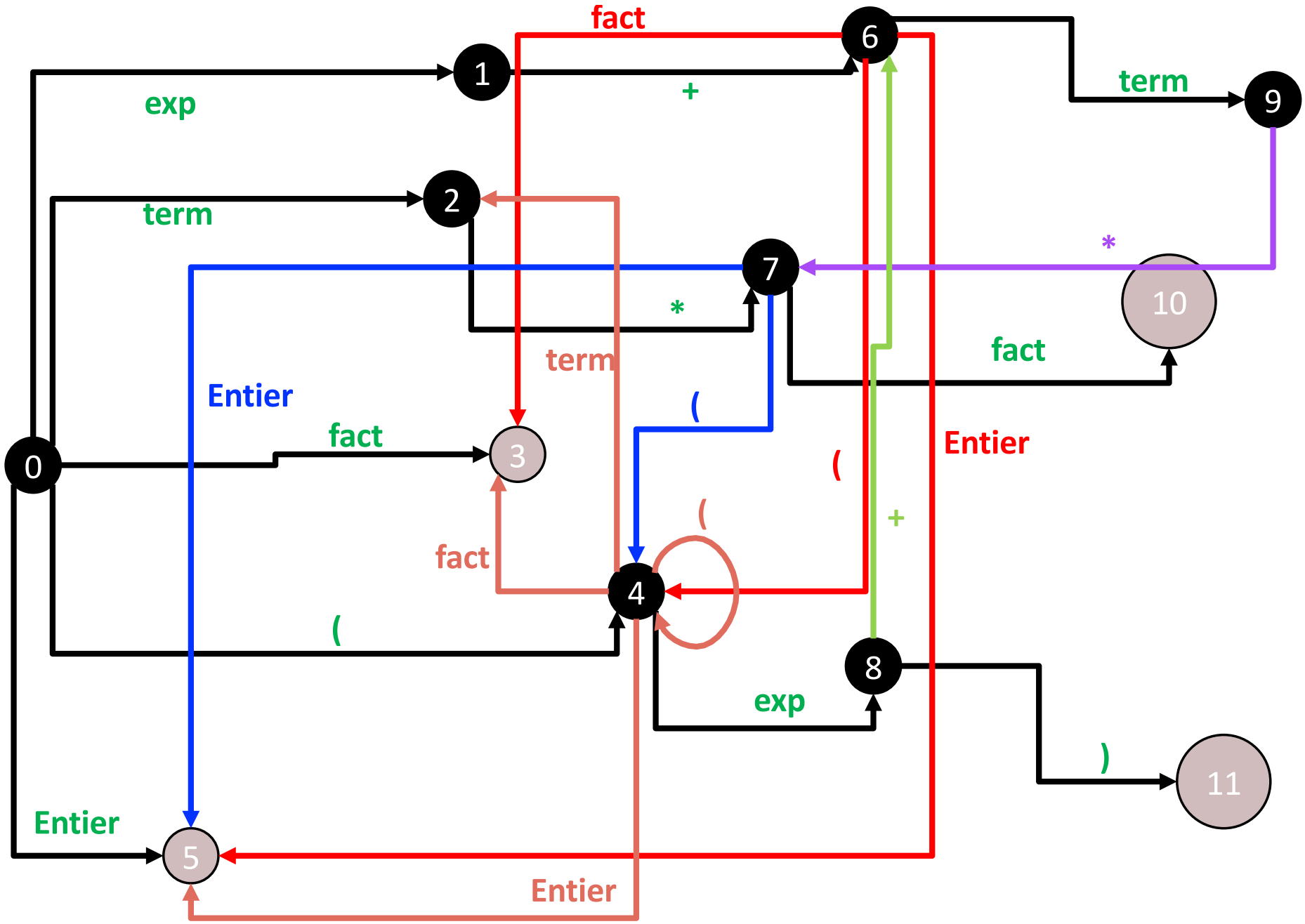
term → (• exp)

Etat_5 // accepter Entier

term → Entier •


0
S→ • exp
exp → • exp + term
exp → • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
exp
term
fact
(
Entier
1
S→ exp •
exp → exp • + term
2
exp → term •
term → term • * fact
3
term → fact •
4
fact → (• exp)
exp → • exp + term
exp → • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
5
fact → Entier •
6
exp → exp + • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
+
7
term → term * • fact
fact → • (exp)
fact → • Entier
*
8
fact → (exp •)
exp → exp • + term
exp
9
exp → exp + term •
term → term • * fact
term
10
term → term * fact •
fact
11
fact → (exp) •
)


0
S→ • exp
exp → • exp + term
exp → • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
1
S→ exp •
exp → exp • + term
2
exp → term •
term → term • * fact
3
term → fact •
4
fact → (• exp)
exp → • exp + term
exp → • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
5
fact → Entier •
6
exp → exp + • term
term → • term * fact
term → • fact
fact → • (exp)
fact → • Entier
7
term → term * • fact
fact → • (exp)
fact → • Entier
8
fact → (exp •)
exp → exp • + term
9
exp → exp + term •
term → term • * fact
10
term → term * fact •
11
fact → (exp) •
exp
term
Entier
fact
(
+
*
)



Automate final

Analyse syntaxique Ascendante

SLR

Construction de la table d'analyse: Action et Aller_à

- Si l'état **état_i** contient une position d'analyse de la forme: notion ⇒ préfixe ● **terminal** suffixe
- et que: Aller_à(état_i, terminal) = état_destination
  - alors on choisit:
  - **ACTION** (état_i, terminal) = **SHIFT** état_destination
- Si l'état état_i contient une position d'analyse: notion ⇒ **corps** ● où notion n'est pas S,
  - alors pour tout terminal de **Suivant(notion)** on fixe:
  - **ACTION**(état_i, terminal) = **REDUCE** notion ⇒ corps
- Si l'état **état_i** contient la position d'analyse : S ⇒ **axiome** ●
  - alors on choisit :
  - **ACTION**(état_i, **$**) = **ACCEPT (dernier REDUCE)**
- Toutes les entrées de la table **action** qui n'ont **pas été garnies** par les quatre considérations ci-dessus sont marquées par:
  - **ACTION**(état_i, terminal_i) = **ERROR**
- on **garde** le contenu de la table **ALLER_A** pour toutes les entrées dont le second argument est une notion **non terminale**

Analyse syntaxique Ascendante

Analyse SLR

Construction de la table Action

Etat	Entier	+	*	(	)	$
0	S 5			S 4		
1		S 6				Acc
2		R 2	S 7		R 2	R 2
3		R 4	R 4		R 4	R 4
4	S 5			S 4		
5		R 6	R 6		R 6	R 6
6	S 5			S 4		
7	S 5			S 4		
8		S 6			S 11	
9		R 1	S 7		R 1	R 1
10		R3	R 3		R 3	R 3
11		R5	R5		R 5	R 5

Etats x $\sum$ = T-->états

S i= Shift, décaler
et empiler l'état i

R j = Reduce avec la régle j

R1: exp → exp + term
R2: exp → term
R3: term → term * fact
R4: term → fact
R5: fact → (exp)
R6: fact → Entier

Analyse syntaxique Ascendante

Construction de la table Aller_à

Etats x $\sum$ = NT-->états

Etat	exp	term	fact
0	1	2	3
1			
2			
3			
4	8	2	3
5			
6		9	3
7			10
8			
9			
10			
11			

Exemple

R1: exp → exp + term
R2: exp → term
R3: term → term * fact
R4: term → fact
R5: fact → (exp)
R6: fact → Entier

Flux d'entrée