Langages de Programmation, Interprétation, Compilation

Christine Paulin
Christine.Paulin@universite-paris-saclay.fr

Département Informatique, Faculté des Sciences d'Orsay, Université Paris-Saclay

LDD3 Informatique, Mathématiques - Magistère Informatique

2025–26

Analyse syntaxique ascendante

- Rappels
- Analyse descendante et ascendante
- Programmer un analyseur ascendant
- Construction de l'automate
- Conflits et Précédences
- Analyse LR(1)

Rappels

Une grammaire G est composée de :

- Ensemble de terminaux : T
- Ensemble de non-terminaux : N, symbole initial (axiome) $S \in N$.
- Ensemble de règles de la forme

$$X \prec x_1 \dots x_n \quad \text{avec} \quad x_i \in N \cup T$$

Exemple

$T = \{a, b\}$, $N = \{X\}$

$$\begin{array}{ll} X & \prec aXb \\ X & \prec \varepsilon \\ X & \prec XX \end{array}$$

Remarques

- Règles possiblement récursives $X \prec aXb$
- Règles qui produisent ou peuvent produire le mot vide

Dérivations

du symbole initial à un mot sur l'alphabet terminal

$$X \to XX \to aXbX \to aaXbbX \to aaXbbaXb \to aabbaXb \to aabbab$$

- Dérivation gauche : remplacement du non-terminal le plus à gauche ;
- Dérivation droite : remplacement du non-terminal le plus à droite.

Langage reconnu

Une grammaire reconnait un ensemble de mots formés sur les symboles terminaux.
Ce sont les mots obtenus par dérivation à partir du symbole initial.

$$S \to m_1 \to \cdots m_n \in T^*$$

On a $m \to m'$ si :

- m contient un non-terminal X
- La grammaire contient une règle $X \prec x_1 \dots x_n$
- m' est obtenu en remplaçant une occurrence de X par $x_1 \dots x_n$

Arbre de dérivation syntaxique

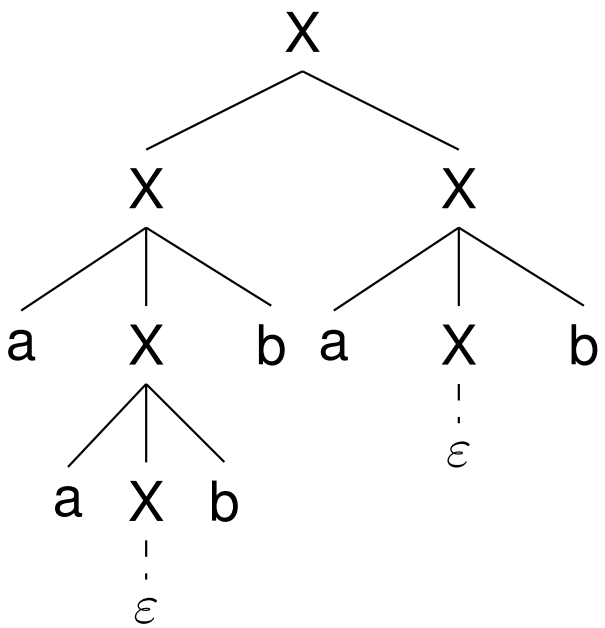
Un mot m est reconnu s'il existe un arbre de dérivation syntaxique.

- les noeuds internes contiennent des symboles non-terminaux
- les feuilles contiennent des symboles terminaux
- la racine est le symbole initial S
- dans un parcours infixe, les feuilles forment le mot m
- si un noeud interne étiqueté X a des sous-arbres $t_1, \ldots, t_n$ de racines $x_1, \ldots, x_n$ alors $X \prec x_1 \ldots x_n$ est une règle de la grammaire.

Remarques

- ne pas confondre arbre de dérivation syntaxique (associé à la grammaire) et arbre de syntaxe abstraite associé au langage.
- l'arbre de dérivation syntaxique n'est pas construit explicitement mais l'algorithme de reconnaissance établit son existence
- l'analyse propose de calculer une valeur récursivement sur cette structure (les équations récursives sont données dans des actions associées aux règles)

Exemple



Principe de l'analyse syntaxique

- Entrée : une suite de tokens (le mot à reconnaitre)
- Recherche d'un arbre de dérivation syntaxique pour ce mot
- Reconstruction d'informations à partir de cet arbre (une action/valeur associée à chaque règle)

Comment reconstruire l'arbre ?

Partie 2 : Analyse lexicale et syntaxique

- Rappels
- Analyse descendante et ascendante
- Programmer un analyseur ascendant
- Construction de l'automate
- Conflits et Précédences
- Analyse LR(1)

Principe de l'analyse descendante

On essaie de construire l'arbre de dérivation syntaxique en partant de la racine.

- Entrée : un mot m (ou l'indice dans une chaîne globale).
- On associe une fonction à chaque non terminal X.
  - la fonction associée au non-terminal X reconstruit (implicitement) un arbre de racine X dont les feuilles forment un préfixe de m ;
  - étant donné le non-terminal X et le mot $\mathtt{m}$, on doit décider de la règle $X \prec x_1 \dots x_n$ à appliquer ;
  - on traite alors les symboles $x_1, \dots, x_n$: lecture d'un terminal ou appel de la fonction associée à un non-terminal
- Pour reconnaître le mot entier, on part de la fonction associée au symbole initial S, on vérifie qu'on atteint la fin de chaîne.

A propos d'analyse descendante

- Assez simple à programmer sans faire appel à des automates.
  - Calcul de l'ensemble des premiers caractères dérivables à partir du membre droit d'une règle.
  - $$\begin{aligned}
    \text{Premier}(\varepsilon) &= \emptyset \\
    \text{Premier}(xm) &= \{x\} \quad x \in T \\
    \text{Premier}(Xm) &= \text{Premier}(X) \quad X \in N, X \not\longrightarrow^* \varepsilon \\
    \text{Premier}(Xm) &= \text{Premier}(X) \cup \text{Premier}(m) \quad X \in N, X \longrightarrow^* \varepsilon \\
    \text{Premier}(X) &= \cup_{X \prec m} \text{Premier}(m)
    \end{aligned}$$
  - Pour positionner une règle $X \prec m$ lorsque $m \longrightarrow^* \varepsilon$, on prend en compte les caractères suivants de X.

A propos d'analyse descendante

- Les grammaires LL(1) sont peu naturelles :

$$\begin{array}{lll} E & \prec ME' & \\ E' & \prec +ME' \mid \varepsilon & \end{array} \qquad \begin{array}{lll} M & \prec AM' & \\ M' & \prec *AM' \mid \varepsilon & \end{array} \qquad A \prec \texttt{id} \mid \texttt{(}E\texttt{)}$$

- Plus difficile de reconstruire les arbres de syntaxe abstraite dans les actions
- Plus difficile de gérer les précédences et associativité.

Analyse ascendante

- L'analyse construit l'arbre de dérivation syntaxique en partant des feuilles.
- On garde en mémoire dans une pile une liste de non-terminaux et de terminaux correspondant à la portion d'arbre reconstruite.
- Deux opérations :
  - lecture/shift : on fait passer le terminal du mot de l'entrée à lire vers la pile
  - réduction/reduce : on reconnait sur la pile la partie droite $x_1 \dots x_n$ d'une règle $X \prec x_1 \dots x_n$, on dépile $x_1 \dots x_n$ et on met) la place X
- Le mot est reconnu si on termine avec le mot vide à lire et le symbole initial sur la pile.

Exemple de dérivation

$$E \prec E{+}E \mid E{*}E \mid (E) \mid \mathtt{n}$$

	pile	entrée	action
1	ε	$n + n * n$	lecture
2	n	$+n * n$	reduction $E \prec n$
3	E	$+n * n$	lecture
4	$E+$	$n * n$	lecture
5	$E + n$	$*n$	reduction $E \prec n$
6	$E + E$	$*n$	lecture
7	$E + E*$	n	lecture
8	$E + E * n$	ε	reduction $E \prec n$
9	$E + E * E$	ε	reduction $E \prec E * E$
10	$E + E$	ε	reduction $E \prec E + E$
11	E	ε	reduction $S \prec E$

Partie 2 : Analyse lexicale et syntaxique

- Rappels
- Analyse descendante et ascendante
- Programmer un analyseur ascendant
- Construction de l'automate
- Conflits et Précédences
- Analyse LR(1)

Décider de l'action

- L'objectif est d'avoir sur la pile des parties droites de règles pour les réduire
- Stratégie en fonction du sommet de pile et de l'entrée
- Exemple

Sommet de pile	**Prochain léxème**	**Action**
n	quelconque	réduction $E \prec n$
$($ E $)$	quelconque	réduction $E \prec (E)$
E $*$ E	quelconque	réduction $E \prec E*E$
E $+$ E	$*$	progression
	autre que $*$	réduction $E \prec E+E$
autres cas	quelconque	progression

Version plus précise

Sommet de pile	Prochain lexème	Action
$\emptyset$	n ou (	progression
	autres	échec
n	quelconque	réduction $E \prec n$
(	n ou (	progression
	autres	échec
(E	) ou + ou $\star$	progression
	autres	échec
(E)	quelconque	réduction $E \prec (E)$
E	+ ou $\star$	progression
	fin de l'entrée	succès
	autres	échec
E $\star$	n ou (	progression
	autres	échec
E $\star$ E	quelconque	réduction $E \prec E \star E$
E +	n ou (	progression
	autres	échec
E + E	$\star$	progression
	autres	réduction $E \prec E+E$

Code associé

```
type token = INT of int | PLUS | STAR | LPAR | RPAR | EOF
type expr  =
  Cst of int
| Add of expr * expr
| Mul of expr * expr
type fragment = T of token | E of expr

let parse input : expr =
let rec ps (p: fragment list *  token list) : expr
  = match p with
      ....
in ps ([], input)
```

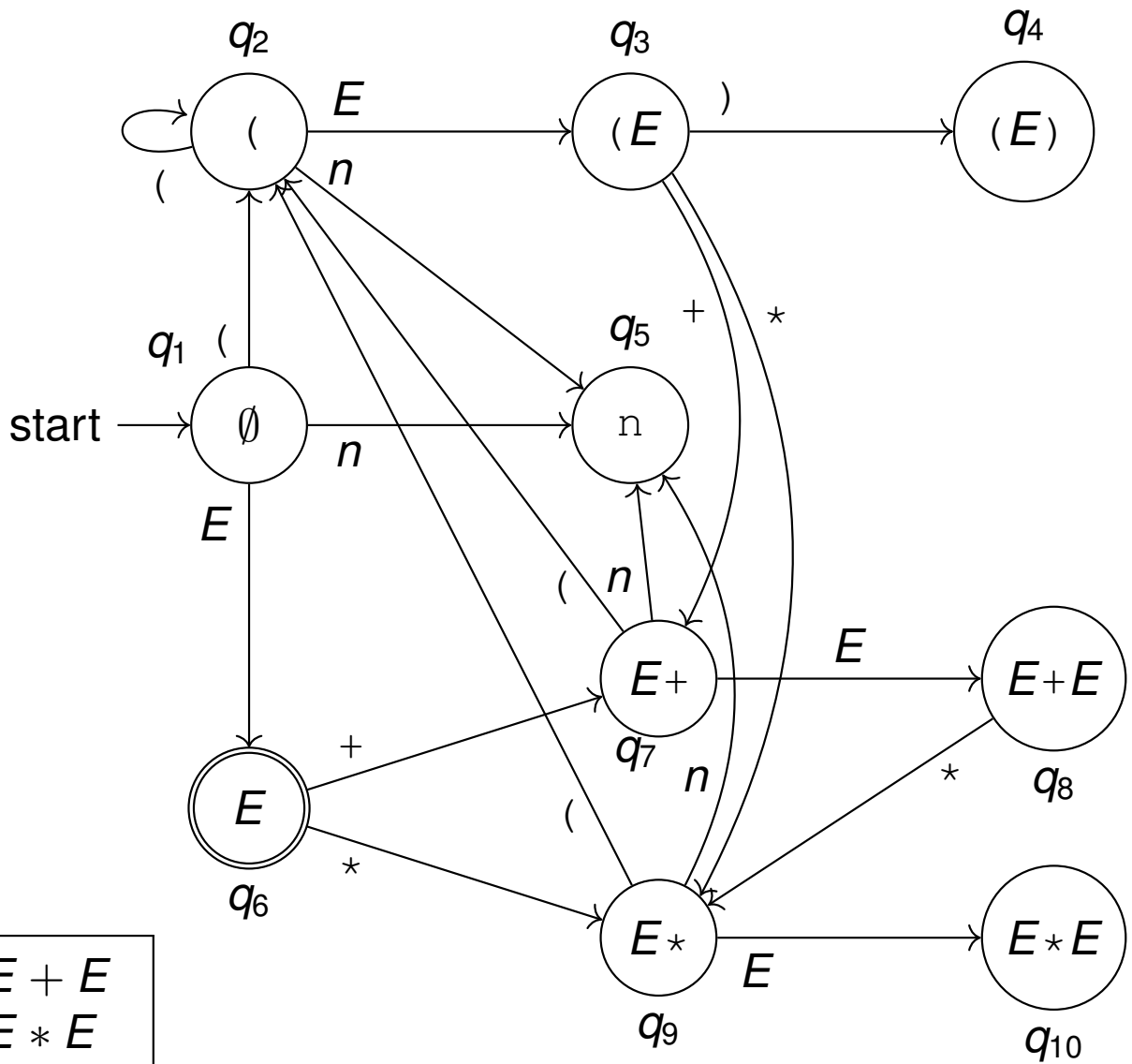
Code associé : fonction `ps`

```
let rec ps (p: fragment list * token list) : expr = match p with
    | ([E e], [EOF]) -> e

    | ([]    ,  (INT _ | LPAR as t) :: l) ->  ps([T t], l)

    | (T (INT n) :: s,  l)                    ->   ps(E (Cst n) :: s, l)

    | (T LPAR :: s,  (INT _ | LPAR  as t) :: l) ->  ps(T t :: T LPAR :: s, l)

    | (E e :: T LPAR :: s,  (RPAR | PLUS | STAR as t) :: l) ->
    ps(T t :: E e :: T LPAR :: s, l)

    | (T RPAR :: E e :: T LPAR :: s,  l) ->  ps(E e :: s, l)

    | (E e :: [],  ((PLUS | STAR as t) :: l) ->  ps(T t :: E e :: [], l)

    | (T (PLUS | STAR as o) :: E e :: s,  (INT _ | LPAR  as t) :: l) ->
    ps(T t :: T o :: E e :: s, l)

    | (E e2 :: T STAR :: E e1 :: s,  l) ->  ps( E (Mul (e1, e2)) :: s, l)

    | (E e1 :: T PLUS :: E e2 :: s,  STAR :: l) ->
    ps(T STAR :: E e1 :: T PLUS :: E e2 :: s,  l)

    | (E e2 :: T PLUS :: E e1 :: s,  l) -> ps(E (Add (e1, e2)) :: s, l)

    | _ -> failwith "syntax␣error"
```

Automate associé

- A chaque état de la pile est associé un état d'un automate fini.
- Une table indique en fonction de l'état courant de la pile et du mot restant à analyser si on doit lire ou réduire.
  - Si lecture le nouvel état dépend de l'état courant et du caractère lu
  - Si réduction par $X \prec x_1 \dots x_n$ alors le nouvel état dépend de l'état avant la reconnaissance de $x_1 \dots x_n$ et de X.
- Chaque état correspond aux parties droites de règles qui pourraient être reconnues à ce point de l'analyse.

Automate associé : exemple



$$
\begin{array}{ll}
E & \prec E + E \\
E & \prec E * E \\
E & \prec (E) \\
E & \prec n
\end{array}
$$

Analyse d'un mot à partir de l'automate

Pile	Pile d'états	Reste	Action
$\emptyset$	q_1	$n_1 \star (n_2 + n_3)$	progression
n_1	$q_1\ q_5$	$\star (n_2 + n_3)$	réd. $E \prec n$
E	$q_1\ q_6$	$\star (n_2 + n_3)$	progression
$E \star$	$q_1\ q_6\ q_9$	$(n_2 + n_3)$	progression
$E \star ($	$q_1\ q_6\ q_9\ q_2$	$n_2 + n_3)$	progression
$E \star (\ n_2$	$q_1\ q_6\ q_9\ q_2\ q_5$	$+ n_3)$	réd. $E \prec n$
$E \star (\ E$	$q_1\ q_6\ q_9\ q_2\ q_3$	$+ n_3)$	progression
$E \star (\ E +$	$q_1\ q_6\ q_9\ q_2\ q_3\ q_7$	$n_3)$	progression
$E \star (\ E + n_3$	$q_1\ q_6\ q_9\ q_2\ q_3\ q_7\ q_5$	$)$	réd. $E \prec n$
$E \star (\ E + E$	$q_1\ q_6\ q_9\ q_2\ q_3\ q_7\ q_8$	$)$	réd. $E \prec E+E$
$E \star (\ E$	$q_1\ q_6\ q_9\ q_2\ q_3$	$)$	progression
$E \star (\ E\ )$	$q_1\ q_6\ q_9\ q_2\ q_3\ q_4$	$\emptyset$	réd. $E \prec (E)$
$E \star E$	$q_1\ q_6\ q_9\ q_{10}$	$\emptyset$	réd. $E \prec E \star E$
E	$q_1\ q_6$	$\emptyset$	succès

Table de transition

	Actions						**Sauts**
	n	$($	$)$	$+$	$\star$	$\sharp$	E
q_1	q_5	q_2					q_6
q_2	q_5	q_2					q_3
q_3			q_4	q_7	q_9		
q_4	$E \prec (E)$						
q_5	$E \prec n$						
q_6				q_7	q_9	succès	
q_7	q_5	q_2					q_8
q_8	$E \prec E+E$				q_9	$E \prec E+E$	
q_9	q_5	q_2					q_{10}
q_{10}	$E \prec E \star E$						

Partie 2 : Analyse lexicale et syntaxique

- Rappels
- Analyse descendante et ascendante
- Programmer un analyseur ascendant
- Construction de l'automate
- Conflits et Précédences
- Analyse LR(1)

Automate LR(0)

- Chaque état de l'automate est un item (une règle de la grammaire avec une position marquée sur la partie droite par un $\bullet$) :
  - Exemple : $E \prec (\bullet\, E)$
- Règle initiale : on suppose (quitte à l'ajouter) qu'on a une seule règle pour le symbole axiome S de la grammaire de la forme $S \prec \alpha\sharp$ avec $\sharp$ un terminal spécial qui marque la fin de l'entrée
- Etat initial : item axiome avec la position au début :

$$S \prec \bullet\alpha\sharp$$

- États acceptants : item axiome avec position à la fin :

$$S \prec \alpha\sharp\bullet$$

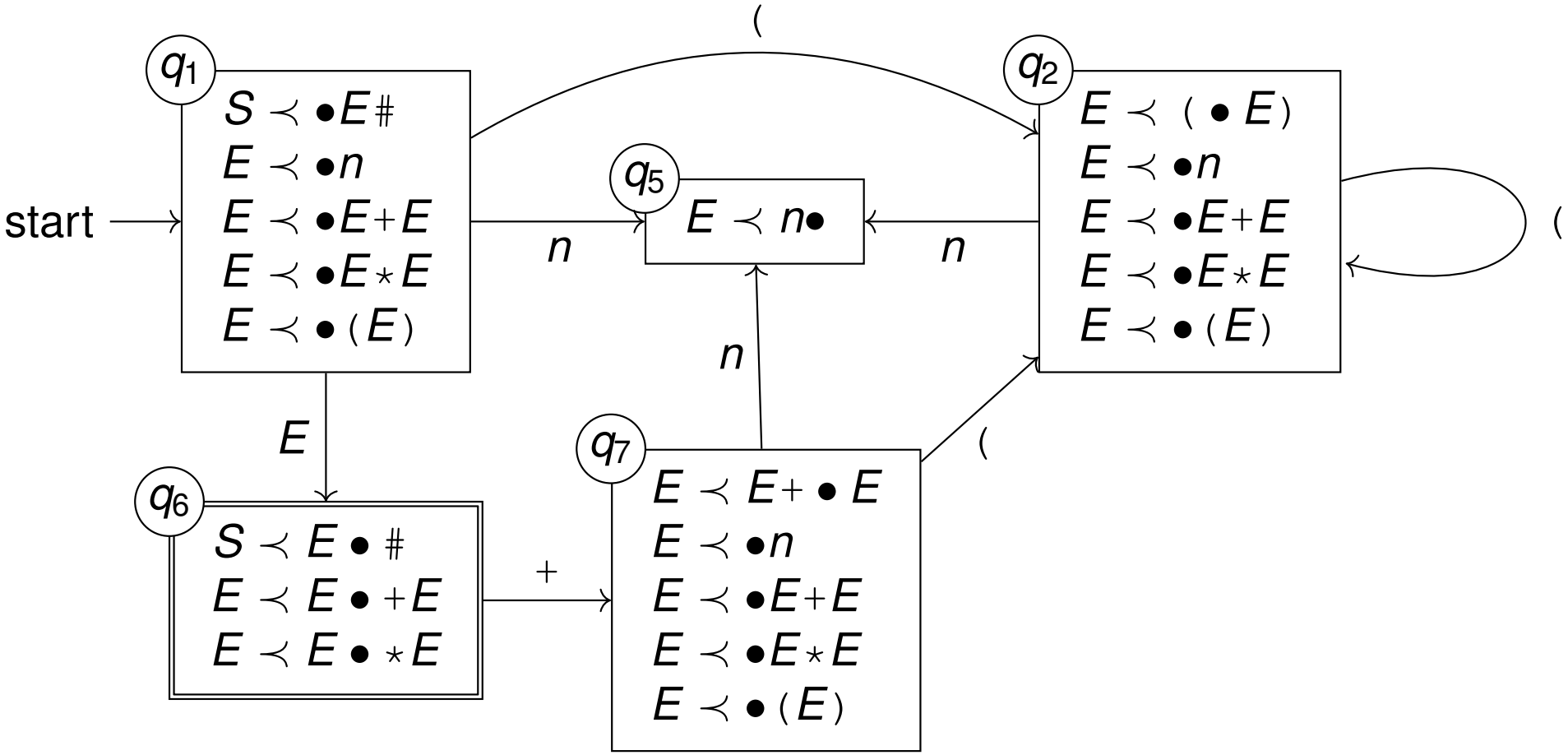
- Transitions : pour chaque règle $x \prec \alpha y \beta$ et (dans le deuxième cas) $y \prec \gamma$
  - $x \prec \alpha \bullet y\beta \xrightarrow{y} x \prec \alpha y \bullet \beta$ si y symbole terminal ou non-terminal
  - $x \prec \alpha \bullet y\beta \xrightarrow{\varepsilon} y \prec \bullet\gamma$ si y symbole non-terminal

Automate LR(0)

- L'automate est non-déterministe car asynchrone (ε-transition)
- Déterminisation :
  - Si un état contient un item $x \prec \alpha \bullet y\beta$ avec y un non-terminal, il contient aussi tous les items $y \prec \bullet\gamma$ pour toutes les règles du non-terminal y
  - Lorsqu'on fait une transition sur un symbole (terminal ou non), on avance la marque dans tous les items où c'est possible
- Parfois on n'écrit pas les items de complétion $y \prec \bullet\gamma$, mais il ne faut pas les oublier dans les transitions...

Automate LR(0) : exemple

- L'état initial $S \prec \bullet E\sharp$ est complété par toutes les règle $E \prec \bullet\alpha$
- Automate (partiel)



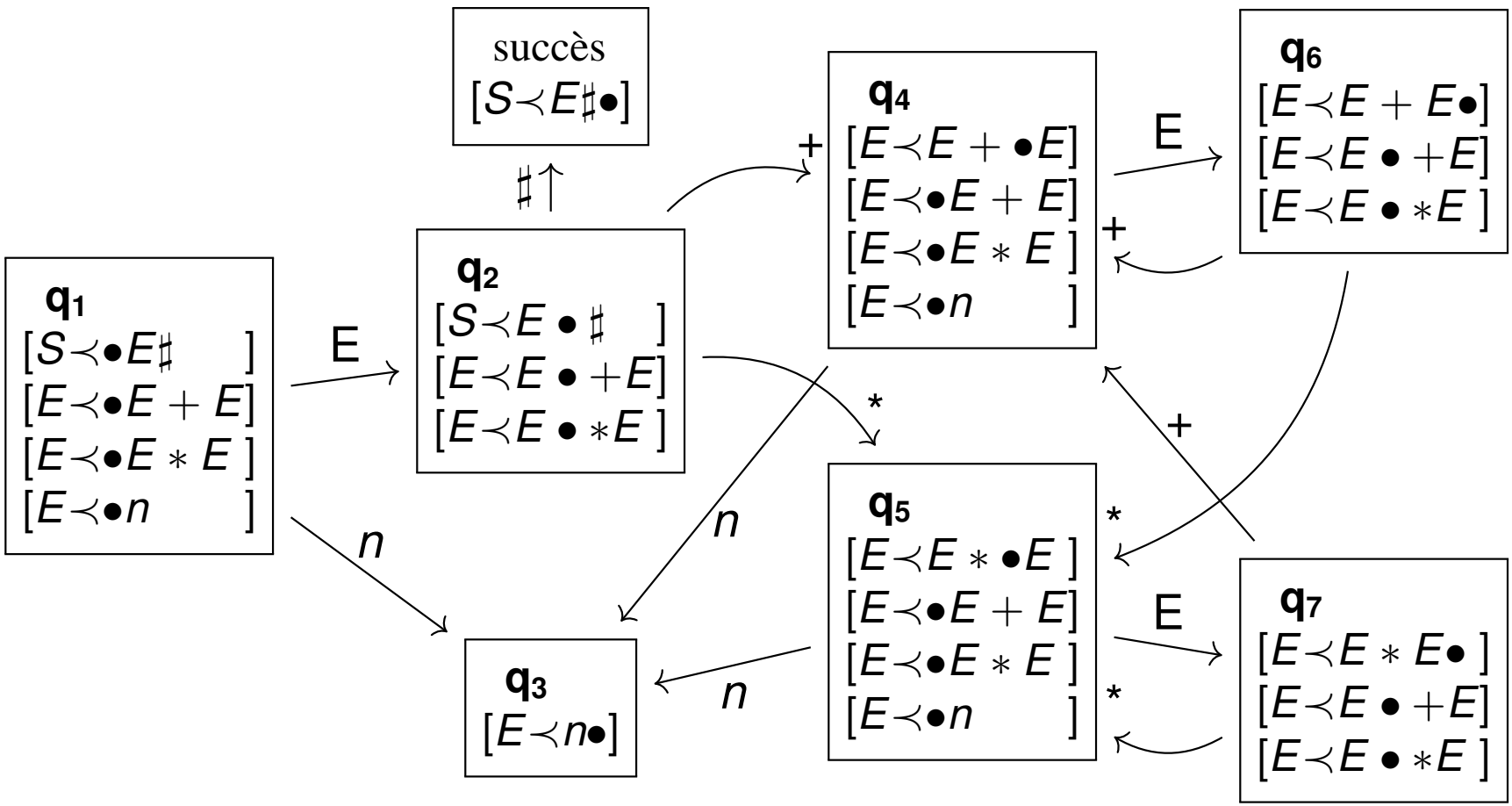
Actions possibles

- Si $s \xrightarrow{a} s'$ avec $a \in T$ alors on peut lire a à partir de s et passer en s'.
- Si s contient un item final $X \to x_1 \dots x_n.$ alors on peut faire une réduction par la règle $X \prec x_1 \dots x_n$.

  On repart de l'état s' avant $x_1 \dots x_n$ et on applique la transition $s' \xrightarrow{X} s''$.
- Conflits :
  - shift/reduce entre une lecture et une réduction
  - reduce/reduce entre deux réductions
- Les conflits peuvent parfois être résolus en identifiant les terminaux qui peuvent suivre le non-terminal lié à la réduction
- Les conflits peuvent correspondre à des ambiguités dans la grammaire ou bien à des limitations de l'analyse.

Automate LR(0) : grammaire simplifiée

$$
\begin{array}{lll}
S & \prec & E\sharp \\
E & \prec & E + E \\
E & \prec & E * E \\
E & \prec & n
\end{array}
$$



Analyse de l'exemple

	pile	entrée	action
1	q_1	$\mathtt{n} + \mathtt{n} * \mathtt{n}\sharp$	lecture
2	$q_1.\mathtt{n}.q_3$	$+\mathtt{n} * \mathtt{n}\sharp$	reduction $E \prec \mathtt{n}$
3	$q_1.E.q_2$	$+\mathtt{n} * \mathtt{n}\sharp$	lecture
4	$q_1.E.q_2.+.q_4$	$\mathtt{n} * \mathtt{n}\sharp$	lecture
5	$q_1.E.q_2.+.q_4.\mathtt{n}.q_3$	$*\mathtt{n}\sharp$	reduction $E \prec \mathtt{n}$
6	$q_1.E.q_2.+.q_4.E.q_6$	$*\mathtt{n}\sharp$	lecture
7	$q_1.E.q_2.+.q_4.E.q_6.*.q_5$	$\mathtt{n}\sharp$	lecture
8	$q_1.E.q_2.+.q_4.E.q_6.*.q_5.\mathtt{n}.q_3$	$\sharp$	reduction $E \prec \mathtt{n}$
9	$q_1.E.q_2.+.q_4.E.q_6.*.q_5.E.q_7$	$\sharp$	reduction $E \prec E * E$
10	$q_1.E.q_2.+.q_4.E.q_5$	$\sharp$	reduction $E \prec E + E$
11	$q_1.E.q_2$	$\sharp$	reduction $S \prec E$
12	$q_1.S.\mathrm{succ}$	ε	succès

Conflits identifiés

- Etats dans lesquels deux réductions sont possibles (conflit reduce/reduce) ou bien une lecture et une réduction (conflit shift/reduce)
- $\boxed{\begin{array}{l}[S \prec E \bullet \sharp \quad] \\ [E \prec E \bullet + E] \\ [E \prec E \bullet * E\ ]\end{array}}$ le caractère suivant sur l'entrée détermine l'action à réaliser
- $\boxed{\begin{array}{l}[E \prec E + E \bullet] \\ [E \prec E . + E\ ] \\ [E \prec E \bullet * E\ ]\end{array}}$ réelle ambiguïté (la grammaire est ambigüe)
- Les règles d'associativité et de priorité indiquent une stratégie
- Priorité entre opérateurs mais le mécanisme doit choisir entre
  - utiliser une règle
  - lire un léxème de plus
- On compare donc la priorité d'une règle et d'un léxème
- Par défaut la règle a la priorité du terminal le plus à droite

Grammaire arithmétique non ambigüe

$$
\begin{array}{lcl}
S & \prec & E\ \# \\
E & \prec & E + M \\
 & | & M \\
M & \prec & M * A \\
 & | & A \\
A & \prec & n \\
 & | & (\ E\ )
\end{array}
$$

Automate déterministe associé

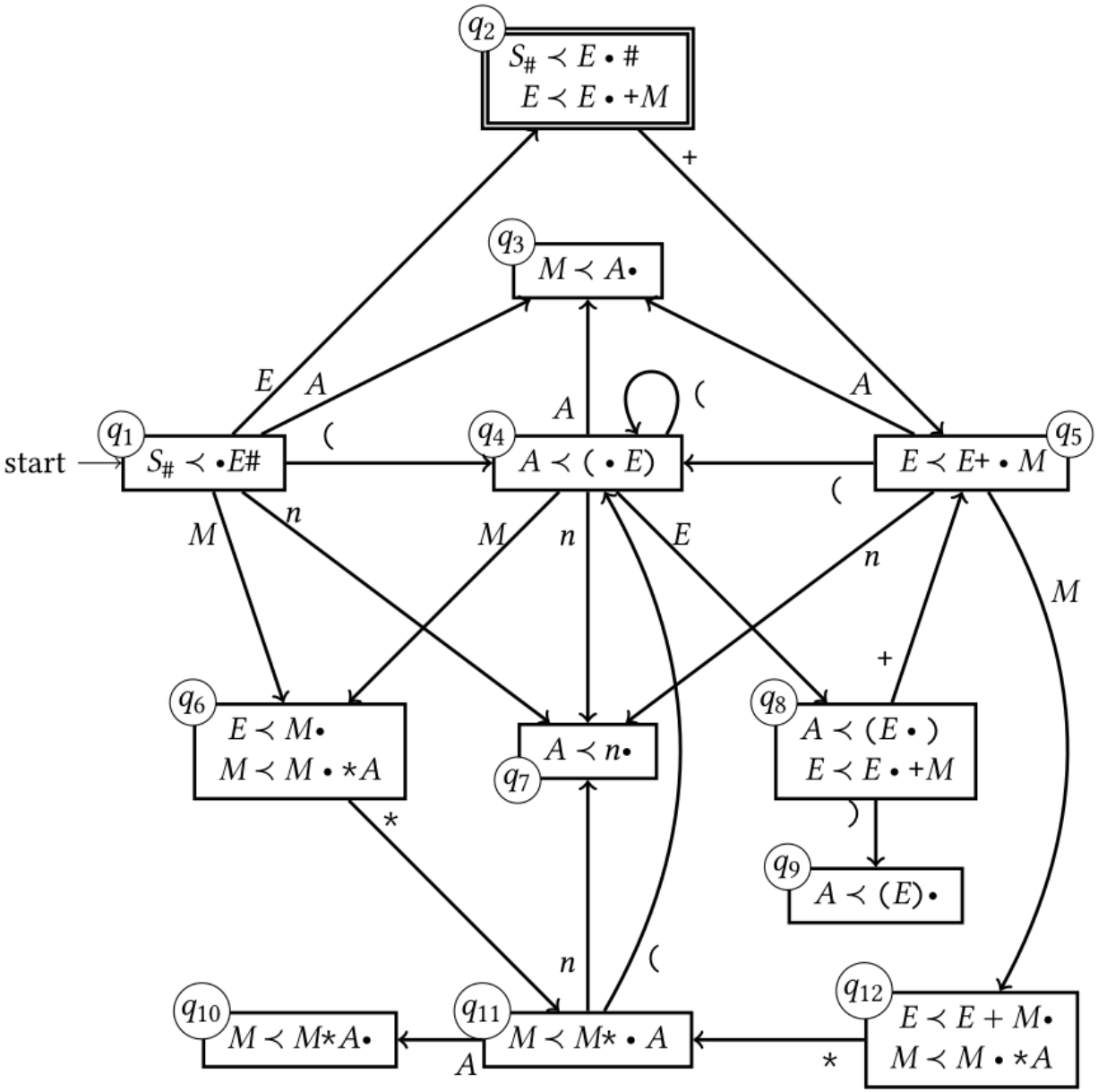


Table d'analyse

	Actions						**Sauts**		
	n	(	)	$+$	$\star$	#	E	M	A
q_1	q_7	q_4					q_2	q_6	q_3
q_2				q_5		ok			
q_3	$M \prec A$								
q_4	q_7	q_4					q_8	q_6	q_3
q_5	q_7	q_4						q_{12}	q_3
q_6					q_{11}				
	$E \prec M$								
q_7	$A \prec n$								
q_8			q_9	q_5					
q_9	$A \prec (E)$								
q_{10}	$M \prec M \star A$								
q_{11}	q_7	q_4							q_{10}
q_{12}					q_{11}				
	$E \prec E + M$								

Conflits et analyse SLR(1)

- La grammaire n'est pas ambigüe mais la table présente encore des conflits.
- état q_6 : $\boxed{\begin{array}{l}[E \prec M\bullet \quad\quad] \\ [M \prec M \bullet * A]\end{array}}$
- état q_{12} : $\boxed{\begin{array}{l}[E \prec E + M\bullet\ ] \\ [M \prec M \bullet * A]\end{array}}$
- Le calcul des suivants de E va permettre de résoudre ce conflit
- Une réduction $E \prec \gamma$ n'aboutit que si le prochain caractère d'entrée appartient à l'ensemble des caractères suivants de E

Calcul des suivants

$$
\begin{array}{lcl}
S & \prec & E\ \sharp \\
E & \prec & E + M \\
 & | & M \\
M & \prec & M * A \\
 & | & A \\
A & \prec & n \\
 & | & (\ E\ )
\end{array}
$$

- Equations

$$
\begin{array}{lll}
\mathtt{suiv}(E) & = \{\sharp, +,)\} & \\
\mathtt{suiv}(M) & = \{*\} \cup \mathtt{suiv}(E) & \{\sharp, +, *,)\} \\
\mathtt{suiv}(A) & = \mathtt{suiv}(M) & \{\sharp, +, *,)\}
\end{array}
$$

- Analyse SLR(1) : On ne réduit une règle $X \prec \beta$ que sur les suivants de X.
- Grammaire SLR(1) : la table d'analyse avec cette stratégie n'a plus de conflits

Nouvelle table d'analyse

	Actions						**Sauts**		
	n	$($	$)$	$+$	$\star$	$\#$	E	M	A
q_1	q_7	q_4					q_2	q_6	q_3
q_2				q_5		ok			
q_3			$M \prec A$						
q_4	q_7	q_4					q_8	q_6	q_3
q_5	q_7	q_4						q_{12}	q_3
q_6			$E \prec M$		q_{11}	$E \prec M$			
q_7			$A \prec n$						
q_8			q_9	q_5					
q_9			$A \prec (E)$						
q_{10}			$M \prec M \star A$						
q_{11}	q_7	q_4							q_{10}
q_{12}			$E \prec E+M$		q_{11}	$E \prec E+M$			

Précédences

- Les conflits reduce-reduce sont à éviter (modifier la grammaire).
- Les conflits shift-reduce peuvent aussi être résolus à l'aide de précédences :
  - La règle a une précédence qui est par défaut celle du terminal le plus à droite mais qui peut être forcée
  - On compare la précédence de la règle R et du caractère à lire c :

    $\texttt{prec}(R) < \texttt{prec}(c)$ lecture

    $\texttt{prec}(R) > \texttt{prec}(c)$ réduction

    $\texttt{prec}(R) = \texttt{prec}(c)$ associativité gauche : réduction

    associativité droite : lecture

Limites de l'analyse SLR(1)

- L'analyse SLR(1) reste grossière : on réduit sur l'ensemble des suivants du non-terminal, sans tenir compte de la position de la règle dans l'arbre
- Item LR(1) :
  - garder trace des caractères qui suivront la réduction de la règle
  - $[X \prec \alpha \bullet \beta, a_1 \dots a_k]$ avec $X \prec \alpha\beta \in R$ et $a_i \in$ Suivants(X).
  - on cherche à reconnaître $\alpha\beta$ forcément suivi d'un des terminaux a_i
- Un état LR(1) correspond à un état LR(0) plus les infos de suivants
- Un état LR(0) peut se transformer en plusieurs état LR(1)

Automates LR(1)

- Etat initial est $[S \prec \bullet A, \#]$.
- Etat acceptant est $[S \prec A\bullet, \#]$.
- Trois types de transition :
  - $[Y \prec \alpha \bullet a\beta, b] \xrightarrow{a} [Y \prec \alpha a \bullet \beta, b]$ avec a symbole terminal,
  - $[Y \prec \alpha \bullet X\beta, b] \xrightarrow{X} [Y \prec \alpha X \bullet \beta, b]$ avec X symbole non terminal,
  - $[Y \prec \alpha \bullet X\beta, b] \xrightarrow{\varepsilon} [X \prec \bullet\gamma, c]$ avec $X \prec \gamma$ une règle et $c = \text{Premiers}(\beta b)$.
- Table d'actions : réduction dans l'état q pour le lexème suivant a si $[X \prec \alpha\bullet, l] \in q$ et $a \in l$.

Exemple

$$\begin{array}{lll} I & \prec & L{=}R \mid R \\ L & \prec & *R \mid \texttt{id} \\ R & \prec & L \end{array}$$

Automate LR(0)

- Etat initial : $\boxed{\begin{array}{l} [S \prec \bullet I \sharp \;\;] \\ [I \prec \bullet L{=}R] \\ [I \prec \bullet R \;\;] \\ [L \prec \bullet *R \;] \\ [L \prec \bullet \texttt{id} \;] \\ [R \prec \bullet L \;\;] \end{array}}$
- Après une transition L on arrive sur l'état $\boxed{\begin{array}{l} [I \prec L \bullet {=}R] \\ [R \prec L \bullet \;\;] \end{array}}$
- Deux possibilités : réduire $R \prec L$ ou bien lire $=$
- Le caractère $=$ fait partie des suivants de R donc cela ne suffit pas

Automate LR(1)

$$
\begin{array}{lll}
I & \prec & L{=}R \mid R \\
L & \prec & *R \mid \texttt{id} \\
R & \prec & L
\end{array}
$$

- Etat initial :

$$
\boxed{\begin{array}{l}
[S \prec \bullet I, \sharp\ \ ] \\
[I \prec \bullet L{=}R, \sharp] \\
[I \prec \bullet R, \sharp\ \ ] \\
[L \prec \bullet *R, \sharp\ ] \\
[L \prec \bullet \texttt{id}, \sharp\ ] \\
[R \prec \bullet L, \sharp\ \ ]
\end{array}}
$$

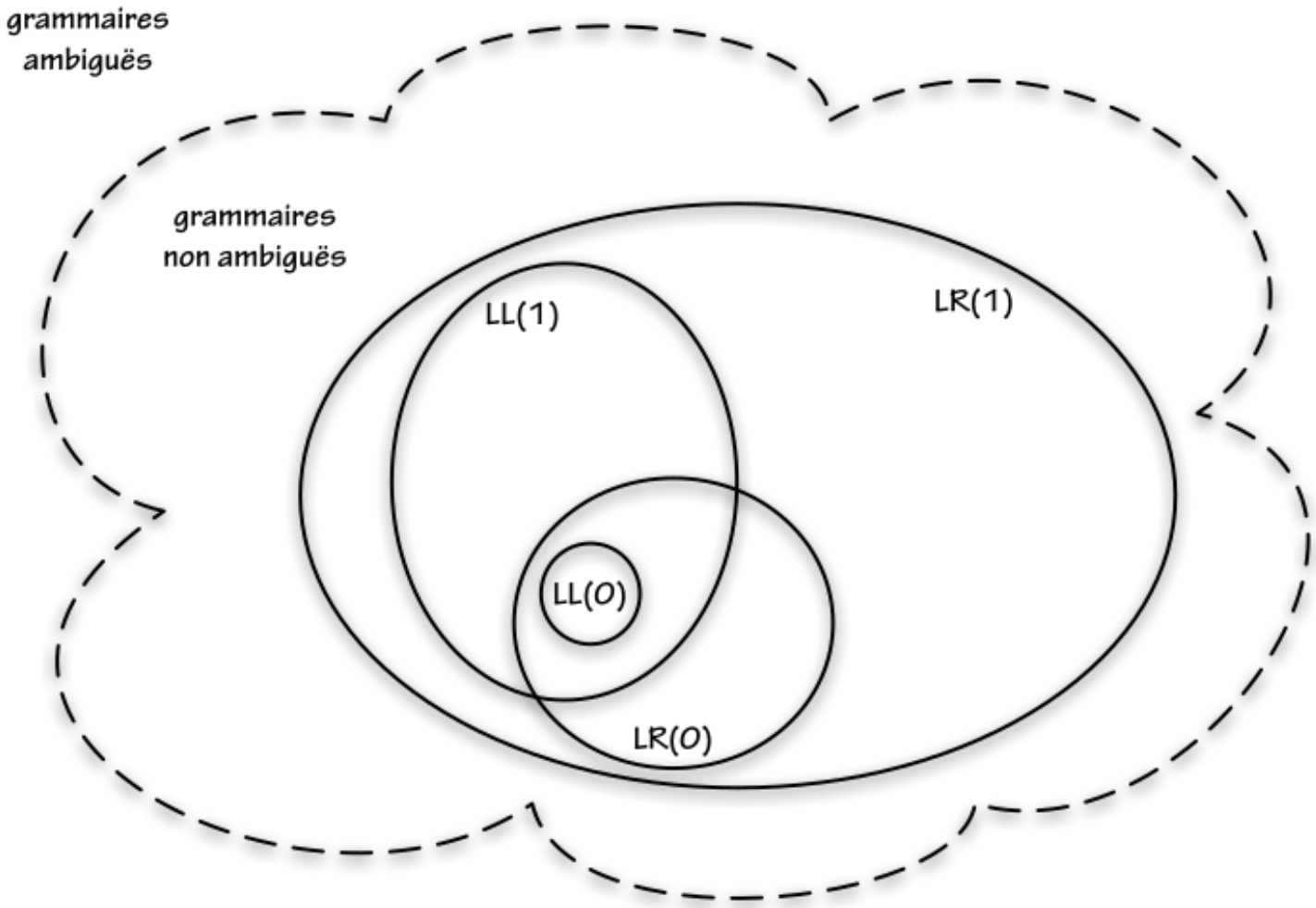
- Après une transition L on arrive sur l'état

$$
\boxed{\begin{array}{l}
[I \prec L \bullet {=}R, \sharp] \\
[R \prec L \bullet, \sharp\ \ ]
\end{array}}
$$

- La réduction n'a lieu que si le caractère suivant est #
- Il faut construire tout l'automate pour vérifier LR(1)

Quelques classes de grammaires

- image reprise du cours de François Pottier (auteur de Menhir)
- les grammaires algébriques même ambigües sont décidables



Synthèse

A retenir

- Le principe de l'analyse ascendante : opérations de lecture/réduction
- La forme des états de l'analyse LR(0) et de l'analyse LR(1)

Savoir faire

- Comprendre les situations de conflit sur l'automate
- Faire le lien entre les choix de précédence et les résolutions de conflit