

INTRODUCTION À LA COMPILATION

Cours 5 : Analyse syntaxique ascendante (deuxième partie)

Yann Régis-Gianas
`yrg@pps.jussieu.fr`

PPS - Université Denis Diderot – Paris 7

vendredi 27 février 2010

Vers une analyse ascendante déterministe

Retour sur l'algorithme de Earley

- ▶ L'algorithme de Earley améliore l'analyse ascendante naïve en la dirigeant à l'aide d'une analyse descendante.
 - ▶ L'analyse descendante est modélisée par les ensembles d'états d'analyse.
 - ▶ Ces états correspondent à plusieurs analyses **concurrentes**.
 - ▶ Le traitement de ces analyses concurrentes peut mener à une exécution en temps cubique.
 - ▶ En s'intéressant aux grammaires pour lesquelles il n'y a qu'une seule analyse possible à la fois, on peut s'assurer d'une complexité linéaire.
- ⇒ Ce sont les algorithmes de la famille LR dont nous allons parlés aujourd'hui.

Algorithme LR(0)

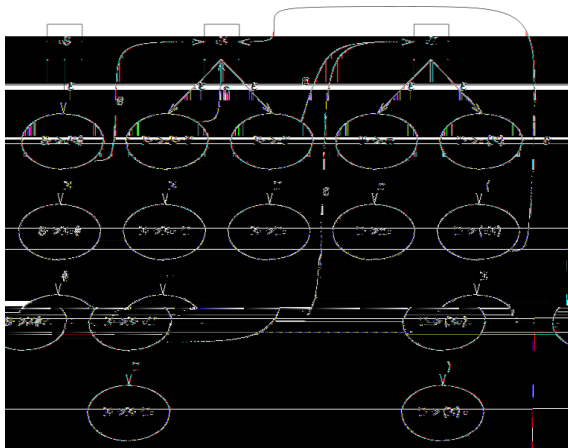
Un automate qui prédit le futur état de l'analyse

- ▶ Dans l'algorithme de Earley, plusieurs non-terminaux pouvaient être en cours d'analyse. En effet, la fonction `predict` introduit autant d'états d'analyse qu'il y a de règles définissant le non-terminal considéré.
- ▶ On peut formuler ce comportement à l'aide d'un **automate fini non-déterministe** qui **analyse la phrase intermédiaire reconnue jusqu'à maintenant** pour déterminer le **prochain état d'analyse**.

Un automate qui prédit le futur état de l'analyse

- ▶ Pour chaque état de l'analyse, on veut trouver un état dans l'automate dont chaque transition est étiquetée par le symbole à reconnaître pour passer de cet état à un état d'analyse suivant.
- ▶ De façon à simplifier la formulation de l'automate, pour chaque non-terminal N , on définit un état « gare » représentant le début de l'analyse de ce non-terminal, que l'on note « $\bullet N$ »
- ▶ Ainsi, si on a un état d'analyse de la forme « $E \rightarrow \dots \bullet F \dots$ », on insert une transition instantanée vers l'état-gare « $\bullet F$ » et une transition étiquetée par F vers l'état « $E \rightarrow \dots F \bullet \dots$ »

Exemple

$$\begin{aligned} S &::= E \# \\ E &::= E - T \mid T \\ T &::= n \mid (E) \end{aligned}$$


tirée de "Parsing Techniques - A Practical Guide" – Grune, Jacobs

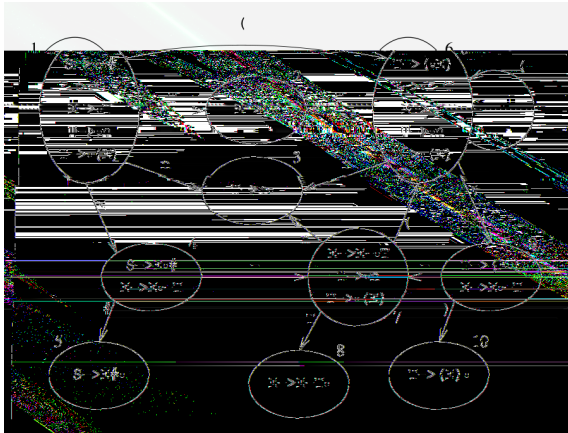
Détermination de l'automate

- ▶ On peut appliquer la méthode de détermination des automates finis.
- ▶ L'automate obtenu s'appelle l'automate LR(0).

Exercice

Déterminez l'automate précédent.

Exemple : Automate LR(0)



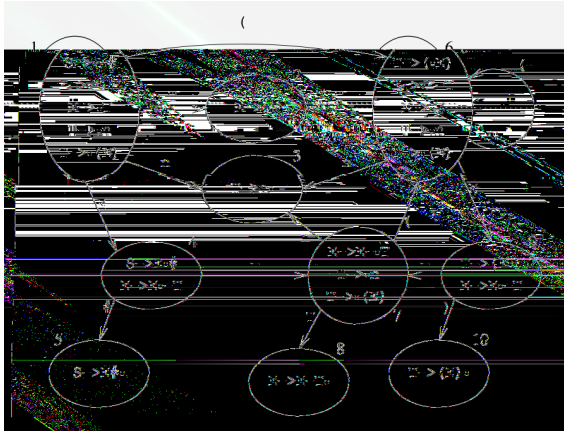
tirée de "Parsing Techniques - A Practical Guide" – Grune, Jacobs

Utilisation de l'automate LR(0)

- ▶ L'automate LR(0) sert à déterminer la **prochaine réduction** à effectuer.
- ▶ On lit la phrase intermédiaire d'analyse de gauche à droite.
- ▶ Cette phrase intermédiaire correspond à la pile de l'automate.
- ▶ Un état d'acceptation de l'automate doit contenir un unique état d'analyse à réduire (c'est-à-dire de la forme « $N \rightarrow \dots \bullet$ »).
- ▶ On réduit en utilisant la règle indiquée par cet état en réécrivant la phrase intermédiaire.
- ▶ Le procédé s'arrête si :
 - ▶ Acceptation de l'entrée : on a réduit le non-terminal d'entrée de la grammaire.
 - ▶ Rejet de l'entrée : une forme intermédiaire est rejetée par l'automate LR(0).

Exemple d'analyse (naïve) LR(0)

- L'analyse de « n - n - n » par l'automate précédent suit les étapes suivantes :



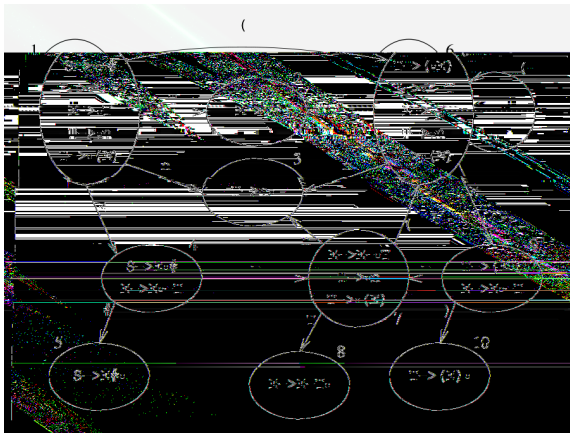
1. n - n - n #
2. T - n - n #
3. E - n - n #
4. E - T - n #
5. E - n #
6. E - T #
7. E #
8. S

Utilisation efficace de l'automate LR(0)

- ▶ Plutôt que de reprendre l'analyse à partir du début de la phrase intermédiaire, on peut se contenter de repartir des derniers états de l'analyse.
- ▶ Pour se souvenir de ces états, on les empile.
- ▶ La pile est donc une succession d'états et de symboles reconnus.

Exemple d'analyse LR(0)

- L'analyse de « n - n - n » par l'automate précédent suit les étapes suivantes :



1. 1 n 3 ⊢ - n - n #
2. 1 T 2 ⊢ - n - n #
3. 1 E 4 - 7 n 3 ⊢ - n #
4. 1 E 4 - 7 T 8 ⊢ - n #
5. 1 E 4 - 7 n 3 ⊢ #
6. 1 E 4 - 7 T 8 ⊢ #
7. 1 E 4 # 5 ⊢
8. S ⊢

Algorithme LR(0)

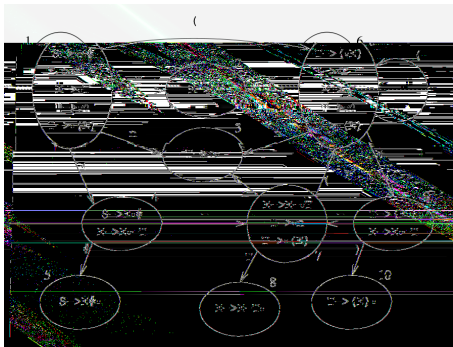
- L'algorithme d'utilisation de l'automate LR(0) avec cette pile contenant des états et symboles s'énonce ainsi :

1. Soit s , l'état au sommet de la pile.
 - Si s est un état final qui contient une règle de la forme « $N \rightarrow w$ » alors dépiler $|w|$ symboles (et leurs états associés) et réduire (pousser N sur la pile).
 - Si s n'est pas un état final, décaler.
2. Soit S , le symbole au sommet de la pile.
 - Pousser l'état de destination de la transition correspondant au symbole S dans l'automate LR(0), si elle existe.
 - Sinon, rejeter.

Représentation sous forme de tables

- ▶ On représente usuellement l'automate LR(0) sous forme de deux tables.
- ▶ La table ACTION a une colonne et une ligne par état de l'automate.
- ▶ Chaque ligne indique ce qui doit être fait en fonction de l'état au sommet de la pile, c'est-à-dire :
 - ▶ « Décaler » ;
 - ▶ ou bien « Réduire » une règle de la forme « $N \rightarrow w$ ».
- ▶ La table GOTO contient une colonne par symbole de la grammaire et une ligne par état de l'automate.
- ▶ Chaque ligne indique les transitions sortantes de l'état associé.

Exemple d'analyse LR(0)



tirée de "Parsing Techniques - A Practical Guide" – Grune, Jacobs

ACTION		GOTO							
		n	-	()	#	E	T	
1	décale	3	⊥	6	⊥	⊥	4	2	1
2	$E \rightarrow T$	2							2
3	$T \rightarrow n$	3							3
4	décale	4	⊥	7	⊥	⊥	5		4
5	$S \rightarrow E \#$	5							5
6	décale	6	3	⊥	6	⊥	⊥	9	2
7	décale	7	3	⊥	6	⊥	⊥		8
8	$E \rightarrow E - T$	8							8
9	décale	9	⊥	7	⊥	10	⊥		9
10	$E \rightarrow (E)$	10							10

Conflits LR(0)

- ▶ On peut construire un automate LR(0) pour toute grammaire hors-contexte.
- ▶ Malheureusement, l'automate LR(0) n'est pas toujours utilisable.
- ▶ En effet, il peut contenir des conflits :

décalage/réduction ou réduction/réduction

⇒ Une grammaire dont l'automate LR(0) n'a pas de conflits est dite LR(0).

Conflits LR(0)

- ▶ Un conflit « décalage/réduction » apparaît lorsqu'un état de l'automate LR(0) contient un état d'analyse à réduire mais n'est pas un état final de l'automate.
 - ⇒ On a le choix entre "décaler T" où T est l'étiquette d'une transition sortante ou bien "réduire" l'état d'analyse à réduire.
- ▶ Un conflit « réduction/réduction » apparaît lorsqu'un état final de l'automate LR(0) contient deux états d'analyse à réduire distincts.
 - ⇒ On ne sait pas quelle réduction effectuer.

Règles de production vide

- ▶ Toute grammaire qui possède une règle de production vide n'est pas LR(0).

Exercice

Pourquoi ?

Règles de production vide : exemple

$$\begin{aligned} S &::= E \# \\ E &::= E Q F \mid F \\ F &::= a \\ Q &::= / \mid \epsilon \end{aligned}$$

- ▶ Dans l'automate non déterministe, à partir de l'état d'analyse « $E \rightarrow E \bullet Q F$ », on doit créer une transition instantanée vers l'état « $Q \rightarrow \bullet$ » et vers l'état « $Q \rightarrow \bullet /$ ».
 - ▶ Dans l'automate déterministe correspondant, ces états sont contenus dans le même état (par fermeture transitive des transitions instantanées).
- ⇒ C'est un conflit décalage/réduction.

Automate ϵ -LR(0)

- ▶ Pour éviter ce conflit, on pourrait essayer de rajouter directement l'état d'analyse « $E \rightarrow EQ \bullet F$ » sans considérer l'état conflictuel « $Q \rightarrow \bullet$ » par une sorte de réduction "au vol" (comme dans l'algorithme de Earley).
- ▶ Cependant, c'est **incorrect** car ce serait supposer une réduction d'un non-terminal Q qui n'a pas vraiment eu lieu.
- ▶ (Dans l'algorithme de Earley, on manipule des ensembles d'analyses potentielles. Dans les algorithmes LR, qui sont déterministes, on ne suppose qu'un chemin d'analyse à la fois donc les réductions supposées doivent réellement avoir lieu.)

Automate ϵ -LR(0)

- ▶ Au lieu de cela, on se donne un nouveau non-terminal « \overline{Q} » qui représente l'**absence** du non-terminal Q .
- ▶ La réduction "au vol" de Q rajoute l'état d'analyse « $E \rightarrow E\overline{Q} \bullet F$ ».

Automate ϵ -LR(0)

- ▶ Lors de la détermination de l'automate, on **identifie** les états d'analyse en supposant que $Q = \overline{Q}$.
 - ▶ Cependant, pendant l'analyse, lorsqu'un état au sommet de la pile représente l'état d'analyse $E \rightarrow EQ F \bullet$, il faut traiter les deux cas suivants :
 - ▶ Soit Q a été réduit par une entrée non vide et alors la pile est de la forme $E Q F$. On doit dépiler ces trois non-terminaux et pousser E .
 - ▶ Soit Q a été réduit par un mot vide et alors la pile est de la forme $E F$. On doit dépiler ces deux non-terminaux et pousser E .
- ⇒ La résolution de conflit s'effectue dynamiquement par observation de la pile.
- ⇒ Un algorithme plutôt complexe !

Algorithme de construction directe de l'automate LR(0)

- ▶ Il n'est pas nécessaire de construire explicitement un automate non-déterministe LR.
- ▶ Un automate déterministe LR peut être directement construit à partir de la grammaire.
- ▶ Il s'agit d'effectuer la déterminisation au fur et à mesure de l'analyse de la grammaire.

Algorithme de construction directe de l'automate LR(0)

- ▶ Soit \mathcal{A} , l'automate LR(0) en cours de construction. On le représente par :
 - ▶ une liste S de paires formées d'un entier et d'un ensemble contenant des états d'analyse et des états-gare. Les entiers servent à numérotter les états de l'automate.
 - ▶ une liste T de transitions « $i \xrightarrow{s} j$ » où i et j sont des numéros d'états et « s » est un symbole (terminal ou non-terminal).
- ▶ Soit une liste \mathcal{L} , une liste d'entiers représentant les états restant à traiter.
- ▶ L'algorithme extrait un élément u de \mathcal{L} , et pour tous les symboles s de la grammaire, fait la chose suivante :

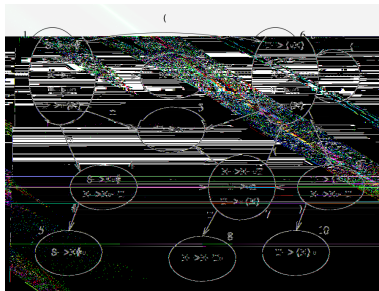
1. Soit \mathcal{E} , un ensemble initialement vide.
2. Pour tous les états d'analyse de l'état u de la forme « $A \rightarrow \alpha \bullet s\beta$ », on insère l'état d'analyse « $A \rightarrow \alpha s \bullet \beta$ » dans \mathcal{E} .
3. Si $\mathcal{E} \not\subseteq S$ alors on lui affecte un numéro non utilisé k . On insère dans (k, \mathcal{E}) dans S , la transition $u \xrightarrow{s} k$ dans T et k dans \mathcal{L} .

Exemple

$$\begin{aligned} S &::= E \# \\ E &::= E - T \mid T \\ T &::= n \mid (E) \end{aligned}$$

Exercice

Appliquez cet algorithme pour reconstruire l'automate :



tirée de "Parsing Techniques - A Practical Guide" – Grune, Jacobs

Les limites des grammaires LR(0)

- ▶ Outre les difficultés posées par les règles de production vide, il s'avère que très peu de grammaire sont LR(0) !
- ▶ Les grammaires LR(0) reconnaissent les langages pour lesquels l'état d'analyse courant suffit à décider ce que l'on doit faire, sans avoir à observer l'entrée.
- ▶ L'entrée sert seulement à confirmer ce choix *a posteriori*

Exemple : une modification légère ?

$$\begin{array}{lcl} S & ::= & E \\ E & ::= & E - T \mid T \\ T & ::= & n \mid (E) \end{array}$$

Exercice

Calculez l'automate LR(0) de cette grammaire.

Exemple : une modification légère ?

$$\begin{array}{lcl} S & ::= & E \\ E & ::= & E - T \mid T \\ T & ::= & n \mid (E) \end{array}$$

- L'automate LR(0) contient un conflit décalage/réduction dans l'état contenant les états d'analyse :

$$S \rightarrow E \bullet - T$$

et

$$S \rightarrow E \bullet$$

⇒ En observant le prochain terminal de l'entrée, ce conflit se résout aisément !

Algorithme LR(1)

Une idée déjà rencontrée ...

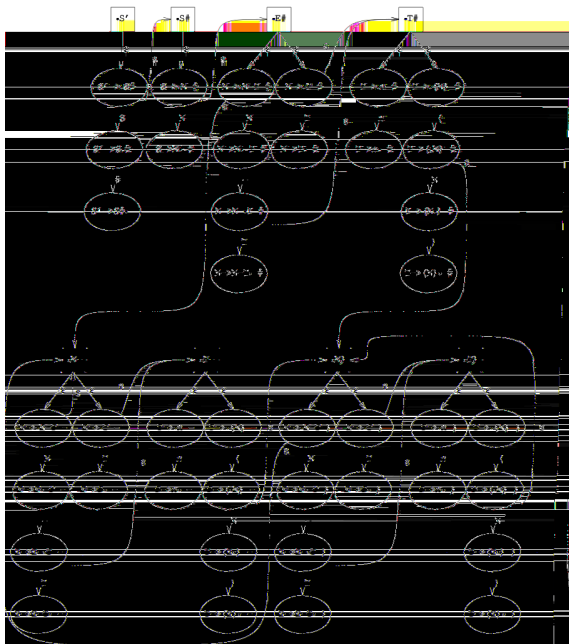
- ▶ Comme pour l'algorithme de Earley, on étend la syntaxe des états d'analyse :

$$N \rightarrow s_1 \dots s_k \bullet s'_1 \dots s'_n \quad [a_1 \dots a_m]$$

Les a_i sont les terminaux attendus en tête de l'entrée une fois que cette analyse sera terminée.

- ⇒ On peut construire un automate fini non déterministe en réutilisant la même méthode que pour l'automate fini non déterministe LR(0). Il suffit seulement en sus de **propager** les terminaux attendus.
- ▶ Comme ces terminaux sont pris en compte pour caractériser les états, l'automate obtenu est par contre beaucoup plus gros !

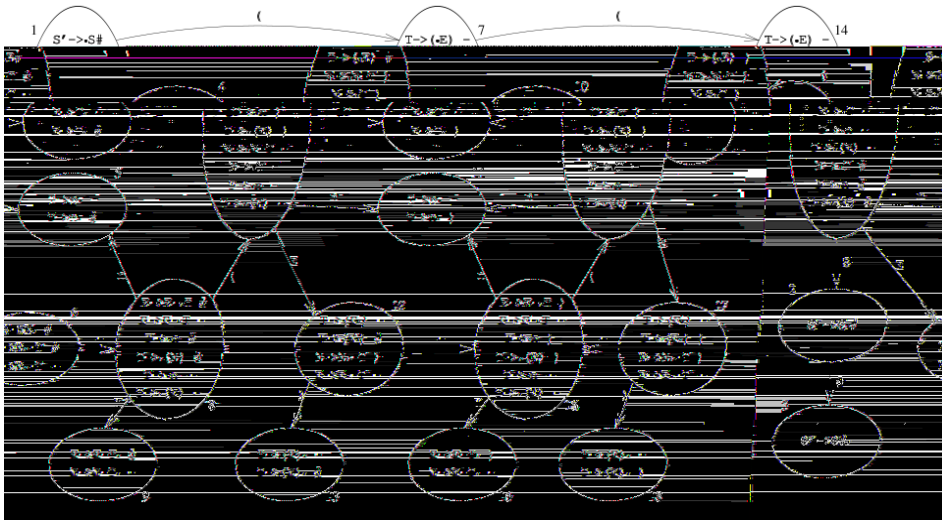
Exemple tirée de "Parsing Techniques - A Practical Guide" – Grune, Jacobs



Propagation des terminaux attendus

- ▶ Les terminaux attendus sont propagés de deux façons distinctes.
- ▶ Des états-gare vers leurs états d'analyses associés : on copie les terminaux attendus.
- ▶ Des états de prédictions de la forme « $A \rightarrow \dots \bullet B s \dots$ » vers l'état-gare concerné (ici, celui de B) :
 - ▶ Si s est un terminal alors c'est le terminal attendu dans la station « $B [s]$ ».
 - ▶ Si s est un non-terminal alors il faut $FIRST(s)$ est l'ensemble des terminaux attendus pour B .

Déterminisation



Sous forme de tables

	ACTION				
	n	-	()	#
1	d	⊥	d	⊥	⊥
2	⊥	r3	⊥	⊥	r3
3	⊥	r4	⊥	⊥	r4
4	⊥	d	⊥	⊥	r1
6	d	⊥	d	⊥	⊥
7	d	⊥	d	⊥	⊥
8	⊥	r2	⊥	⊥	r2
9	⊥	r3	⊥	r3	⊥
10	⊥	r4	⊥	r4	⊥
11	⊥	d	⊥	d	⊥
12	⊥	r5	⊥	⊥	r5
13	d	⊥	d	⊥	⊥
14	d	⊥	d	⊥	⊥
15	⊥	r2	⊥	r2	⊥
16	⊥	d	⊥	d	⊥
17	⊥	r5	⊥	r5	⊥

	GOTO							
	n	-	()	#	S	E	T
1	3		6			T	4	2
2								
3								
4		7						
6	10		13				11	9
7	3		6					8
8								
9								
10								
11		14		12				
12								
13	10		13				16	9
14	10		13					15
15								
16		14		17				
17								

- ▶ « ⊥ » signifie « rejeter ».
- ▶ « ⊥ » signifie « accepter ».
- ▶ « d » signifie « décaler ».
- ▶ « rN » signifie « réduire par la règle N ».

L'automate LR(1)

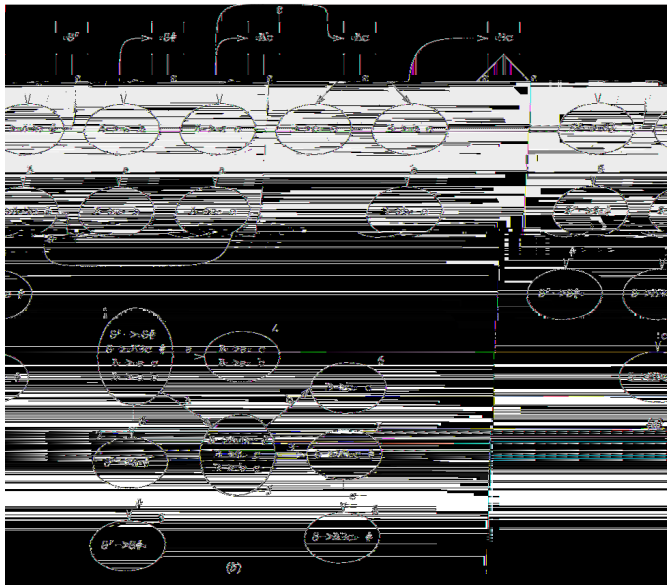
- ▶ L'automate déterministe obtenu est l'automate LR(1).
- ▶ Une grammaire dont l'automate LR(1) n'a pas de conflit est dite LR(1).
- ▶ Deux avantages :
 - ▶ beaucoup de grammaires utiles en pratique sont LR(1) ;
 - ▶ les règles de production vide sont traitées "naturellement" (cf. la suite)
- ▶ Un défaut de taille : le nombre d'états de l'automate est très important !

Les règles de production vide

$$\begin{array}{lcl} S & ::= & ABc \\ A & ::= & a \\ B & ::= & b \mid \epsilon \end{array}$$

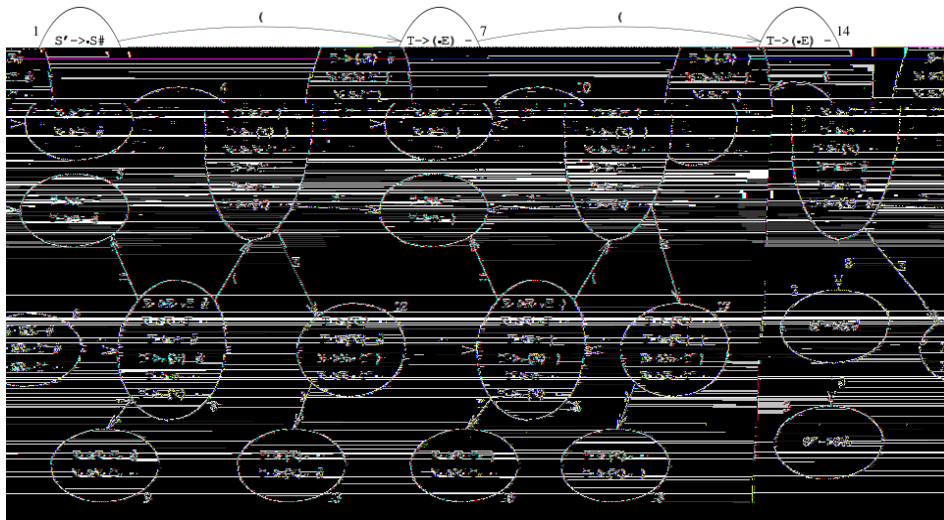
- ▶ $FIRST(B) = \{b, \epsilon\}$ donc il faut ajouter c aux terminaux attendus après A .
- ▶ Il n'y a pas d'autre difficulté !

Les règles de production vide



Algorithme LALR(1)

« Redondance » dans l'automate LR(1)



Agglomération d'états de l'automate LR(1)

- ▶ On appelle **noyau** d'un état de l'automate LR(1), l'ensemble des états d'analyse de cet automate dont on a **e acé**, pour chacun, l'ensemble des terminaux attendus.
- ▶ Ainsi l'état 2 de l'automate LR(1) :

$$\begin{aligned} E &\rightarrow T \bullet [-] \\ E &\rightarrow T \bullet [\#] \end{aligned}$$

... et l'état 9 :

$$\begin{aligned} E &\rightarrow T \bullet [-] \\ E &\rightarrow T \bullet [)] \end{aligned}$$

... sont agglomérables en un unique état :

$$\begin{aligned} E &\rightarrow T \bullet [\#] \\ E &\rightarrow T \bullet [-] \\ E &\rightarrow T \bullet [)] \end{aligned}$$

... qui correspond à l'état 2 de l'automate LR(0) mais avec des terminaux attendus associés aux états d'analyse !

« Redondance » éliminée



Élimination du conflit initial

- ▶ On peut remarquer que le conflit initial de l'automate LR(0) est éliminé par la présence des terminaux attendus, comme dans le cas de l'automate LR(1).
- ▶ En outre, l'automate obtenu est beaucoup plus petit.
- ▶ Cet automate se nomme **LALR(1)**, pour

Look Ahead LR(0) with a look-ahead of 1 token

⇒ C'est l'automate produit par yacc et bison.
(mais pas par MENHIR)

Construction directe de l'automate

- ▶ En pratique, il serait contre-productif d'utiliser le procédé précédent pour construire l'automate LALR(1) puisqu'on s'est appuyé sur l'automate LR(1) !
- ⇒ On peut éviter de **matérialiser** l'automate LR(1) en opérant l'agglomération des états "au vol" pendant la construction de la table.

Algorithme de construction directe de l'automate LALR(1)

1. Soit \mathcal{E} , un ensemble initialement vide.
2. Pour tous les états d'analyse de l'état u de la forme
« $A \rightarrow \alpha \bullet s\beta [a_1 \dots a_n]$ », on insère dans \mathcal{E} l'état d'analyse :
 - ▶ « $A \rightarrow \alpha s \bullet \beta [FIRST(\beta)]$ »
 - ▶ « $A \rightarrow \alpha s \bullet \beta [FIRST(\beta) \cup \{a_1 \dots a_n\}]$ » si $\epsilon \in FIRST(\beta)$.
3. Si il n'existe pas d'état dans S possédant le même noyau que \mathcal{E} alors on lui affecte un numéro non utilisé k . On insère dans (k, \mathcal{E}) dans S , la transition $u \xrightarrow{s} k$ dans T et k dans \mathcal{L} .
4. Si il existe un état \mathcal{U} de S possédant le même noyau que \mathcal{E} alors on fusionne \mathcal{U} et \mathcal{E} en un état \mathcal{W} que l'on rajoute dans la liste \mathcal{L} si il est différent de \mathcal{U} .

Exemple

$$\begin{array}{lcl} S & ::= & E \\ E & ::= & E - T \mid T \\ T & ::= & n \mid (E) \end{array}$$

Exercice

Calculez l'automate LALR(1) de cette grammaire à l'aide l'algorithme précédent.

Où a-t-on triché?

- ▶ Par malchance, la fusion des états ayant le même noyau n'est pas une opération **conservative** : on peut introduire des conflits dans l'automate LALR qui n'étaient pas présents dans l'automate LR.
- ⇒ Cette situation est très rare mais, lorsqu'elle arrive, elle est difficile à expliquer.
- ⇒ C'est pour cela que `MENHIR` utilise une compression **conservative** de l'automate LR.

Conflits LALR

- ▶ Si l'automate LALR a un conflit « décalage/réduction », alors ce conflit état déjà présent dans l'automate LR.
- ▶ En effet, si un automate contient un état possédant un état d'analyse à réduire et un état d'analyse à décaler alors cela signifie qu'un état de l'automate LR avait un noyau identique et donc, un conflit lui aussi.
- ▶ Par contre, certains conflits « réduction/réduction » peuvent être présents dans l'automate LALR et absents de l'automate LR. . .

Exemple de conflits LALR

S	$::=$	$E\#$
E	$::=$	$aBc \mid bCc \mid aCd \mid bBd$
B	$::=$	e
C	$::=$	e

- L'automate LR contient un état « $C \rightarrow e \bullet [c], B \rightarrow e \bullet [d]$ » et un **autre** état de même noyau « $C \rightarrow e \bullet [d], B \rightarrow e \bullet [c]$ »
- ⇒ La fusion de ces deux états introduit le conflit.

Synthèse

Analyse ascendante

- ▶ Nous avons découvert l'analyse syntaxique LR, c'est-à-dire une classe d'algorithmes s'appuyant sur des automates déterministes pour réaliser une analyse ascendante.
- ▶ Ces automates sont les plus couramment utilisés par les générateurs d'analyseurs syntaxiques.
- ▶ La prochaine fois, nous concluerons ce cours sur l'analyse syntaxique en répondant aux questions suivantes :
 - ▶ Comment construire un arbre de production à l'aide des algorithmes étudiés ?
 - ▶ Comment utiliser `MENHIR` ?
 - ▶ Comment résoudre un conflit ?

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐

QCM : Question 1

☐☐☐☐