Analyse syntaxique

Analyse syntaxique

Analyse ascendante *LR*

Rappel - Analyse *LL*

Nous avons vu les analyses descendantes *LL* dans lesquelles

- on cherche à construire une dérivation gauche à partir de l'axiome qui permette d'obtenir le mot d'entrée
- l'automate à pile part de la racine de l'arbre et progresse vers les feuilles en procédant à des remplacements de symboles non terminaux par la partie droite d'une règle dont ils sont partie gauche (parfois appelé *expansion*)

Exemple - Rappel analyse LL

Pour les expressions

Grammaire de départ

$$
\begin{array}{lcl}
E & \to & E + T \mid T \\
T & \to & T * F \mid F \\
F & \to & (E) \mid id \\
id & \to & 1 \mid 2
\end{array}
$$

Grammaire obtenue à la section précédente

$$
\begin{array}{lcl}
E & \to & TE' \\
E' & \to & +TE' \mid \varepsilon \\
T & \to & FT' \\
T' & \to & *FT' \mid \varepsilon \\
F & \to & (E) \mid id \\
id & \to & 1 \mid 2
\end{array}
$$

L'analyse $LL(1)$ nous avait conduit à reconstruire pour le mot $2 * (1 + 1)$ la dérivation gauche en partant de l'axiome :

$$
\begin{array}{lcl}
E & \to & TE' \to FT'E' \to idT'E' \to 2T'E' \to 2 * FT'E' \to 2 * (E)T' \\
 & \to & 2 * (TE')T' \to \ldots
\end{array}
$$

Analyse *LL* - Pas toujours possible

Considérons la grammaire

$$\begin{array}{lcl} S & \rightarrow & A \mid B \\ A & \rightarrow & aAb \mid c \\ B & \rightarrow & aBbb \mid d \end{array}$$

- est-elle ambigüe ?
- existe-t-il $k > 0$ tel qu'elle soit $LL(k)$?

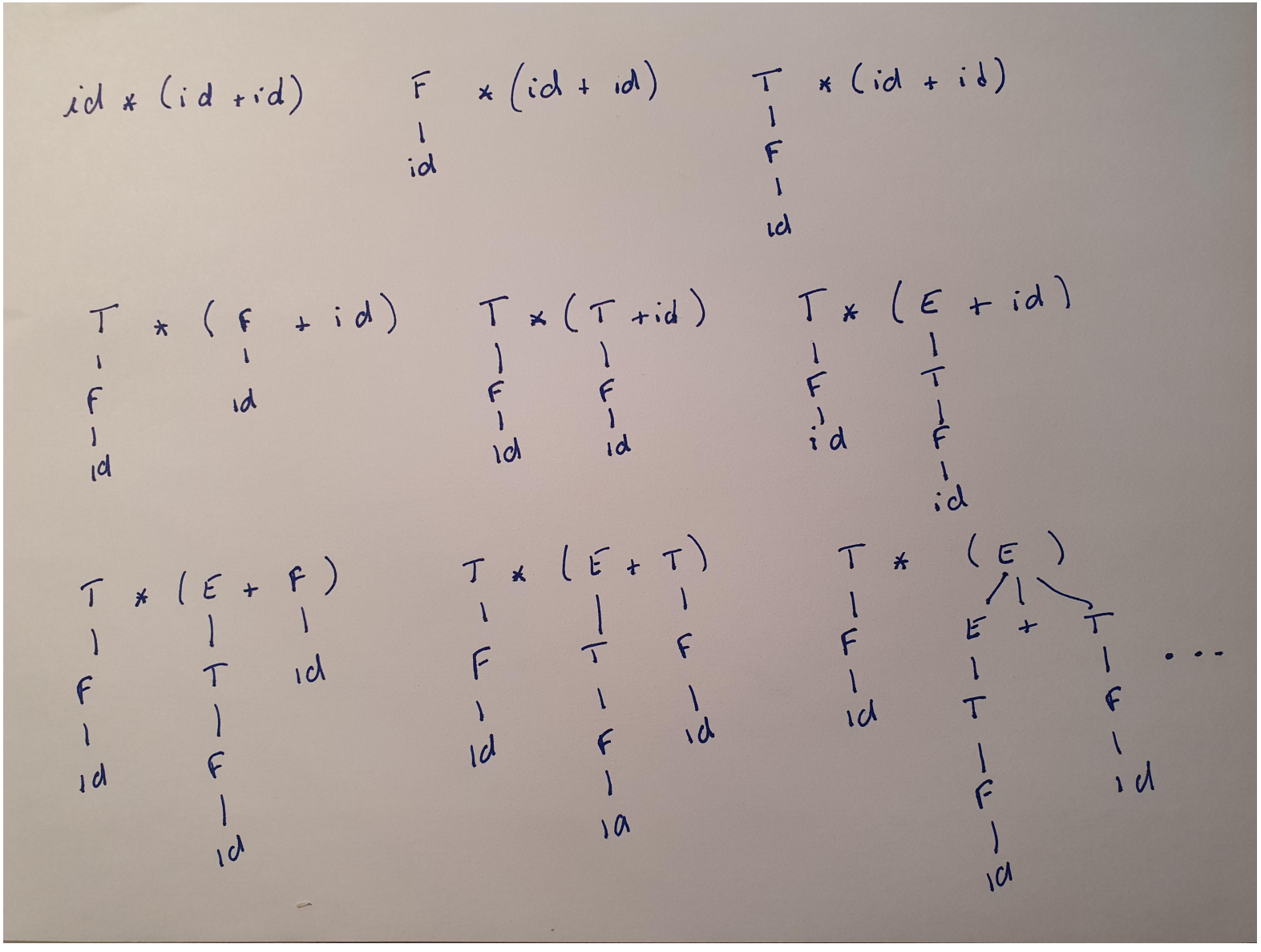
Idée générale - Analyse LR

Nous allons à présent aborder les analyses ascendantes *LR* **left-to-right, rightmost derivation** dans lesquelles

- l'idée est cette fois de construire, à partir d'un mot d'entrée, une dérivation droite : reconstruire « à l'envers » la dérivation à partir des feuilles de l'arbre
- partant des feuilles, on va cette fois remplacer une suite de symboles correspondant à la partie droite d'une règle par sa partie gauche (aussi appelé *réduction*)

➥ Pour ce cours, référence principale : Dragon Book, 2007

Exemple - Expressions



Idée générale - Décalage/Réduction

En analysant un mot de gauche à droite, on cherche à se ramener à l'axiome en remplaçant un peu à la fois les parties droites de règles contruites par la partie gauche correspondante

En cours d'analyse, un proto-mot $\alpha\beta u$ est construit avec $u \in \Sigma^*$ la poursuite de la recherche peut considérer deux cas :

- il existe une règle $T \to \beta$ et une réduction est appliquée pour poursuivre avec le proto-mot $\alpha T u$
- $u = xu'$ avec $x \in \Sigma$, on applique un décalage pour poursuivre l'analyse avec le proto-mot $\alpha\beta' u'$ et $\beta' = \beta x$

Cette stratégie va mener à une dérivation droite, en effet, chaque fois qu'on applique une règle $T \to \beta$, le facteur du mot qui suit β est un mot de Σ^*

Idée générale - Décalage / Réduction - Ex $id * (id + id)$ (1/2)

Grammaire exemple « fil rouge » tirée du Dragon Book (ch 4) :

$E \to E + T \mid T$	$T \to T * F \mid F$	$F \to (E) \mid id$

Analyse

α	u	
ε	$id * (id + id)$	*shift*
id	$*(id + id)$	$F \to id$
F	$*(id + id)$	$T \to F$
T	$*(id + id)$	*shift*
$T*$	$(id + id)$	*shift*
$T * ($	$id + id)$	*shift*
$T * (id$	$+id)$	$F \to id$
$T * (F$	$+id)$	$T \to F$
$T * (T$	$+id)$	$E \to T$

$T * (E$	$+id)$	*shift*
$T * (E+$	$id)$	*shift*
$T * (E + id$	$)$	$F \to id$
$T * (E + F$	$)$	$T \to F$
$T * (E + T$	$)$	$E \to E + T$
$T * (E$	$)$	*shift*
$T * (E)$	ε	$F \to (E)$
$T * F$	ε	$T \to T * F$
T	ε	$E \to T$
E	ε	*success*

Dérivation droite produite : $E \to T \to T * F \to T * (E) \to T * (E + T) \to T * (E + F) \to T * (E + id) \to T * (T + id) \to \ldots$

Idée générale - Décalage / Réduction - Ex $id * (id + id)$ (2/2)

Grammaire : | $E \to E + T \mid T$ | $T \to T * F \mid F$ | $F \to (E) \mid id$ |

Remarque

α	u	
ε	$id * (id + id)$	*shift*
id	$*(id + id)$	$F \to id$
F	$*(id + id)$	$T \to F$
T	$*(id + id)$	~~*shift*~~
~~$T*$~~	~~$(id + id)$~~	~~*shift*~~
T	$*(id + id)$	$E \to T$
E	$*(id + id)$	*shift*
$E*$	$(id + id)$	*shift*
$E * ($	$id + id)$	*shift*
$E * (id$	$+id)$	*shift*
$E * (id+$	$id)$	*shift*
$E * (id + id$	$)$	*shift*
$E * (id + id)$		*fail*

➥On souhaite évidemment éviter le non déterminisme

Intuition

- Nous essayons de réduire les suffixes de la partie en cours d'analyse

➥ se prête bien à l'utilisation d'une pile

- Nous allons construire des automates à pile, ces automates sont aussi appelés *shift-reduce* car ils peuvent
  - *shift* (décalage de l'entrée) : lire un symbole de l'entrée et l'empiler
  - *reduce* (réduction) : réduire une règle, soit remplacer un mot en sommet de pile qui correspond à la partie droite d'une règle par la partie gauche de la règle
- Problème de déterminisme, à chaque étape
  - choix entre shift et reduce
  - choix de la partie du sommet de pile réduire, aussi appelé « handle »
  - choix de la règle à appliquer

Théorie - Automate shift/reduce

⚠ On représente le fond de pile à gauche et le sommet de pile à droite

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, on construit un automate à pile généralisé $\mathcal{B} = (\Sigma, V \cup \Sigma, \varepsilon, \{q_0\}, q_0, \emptyset, \delta)$:

- l'automate accepte le mot lorsque la pile contient uniquement le symbole S
- les transitions sont généralisées (on omet q_0) $(\Sigma \cup V)^* \times (\Sigma \cup \{\varepsilon\})$ dans $(\Sigma \cup V)$ et composées de :
  - shift : $\{(\varepsilon, x, x) \mid x \in \Sigma\}$
    ➥ Sans toucher au sommet de pile, on empile x après l'avoir lu sur l'entrée
  - reduce : $\{(\alpha, \varepsilon, T) \mid T \to \alpha \in R\}$
    ➥ si le mot α est en sommet de pile et qu'il existe une règle $T \to \alpha$, sans avancer dans l'entrée, on peut remplacer α par T en sommet de pile

Pourquoi cela fonctionne-t-il ?

Intuition. En cours de dérivation droite, le proto-mot a une forme $\alpha T u$ où $T \in V$ est le prochain symbole à dériver et $u \in \Sigma^*$, la règle utilisée pour dériver T peut produire ou pas le prochain non terminal à dériver :

Cas 1	*Pile*	*Mot*	*Action*
$\alpha T u \to \alpha\beta Z v u \to \alpha\beta\gamma v u$	$\alpha\beta\gamma$	vu	*reduce*
application de $T \to \beta Z v$	$\alpha\beta Z$	vu	*shift**
puis $Z \to \gamma$	$\alpha\beta Z v$	u	*reduce*
	αT	u	

Cas 2	*Pile*	*Mot*	*Action*
$\alpha T u \to \alpha' Z w v u \to \alpha'\gamma w v u$	$\alpha'\gamma$	wvu	*reduce*
application de $T \to v$	$\alpha' Z$	wvu	*shift**
puis $Z \to \gamma$	$\alpha' Z w v$	u	*reduce*
	$\alpha' Z w T$	u	

Proposition. L'automate shift/reduce reconnaît le langage généré par la grammaire

k-conflits

Soient $x, x' \in \Sigma$, $T, T' \in V$, $u, u', v \in \Sigma^*$ et $\alpha, \beta, \alpha', \beta' \in (\Sigma \cup V)^*$

- Il existe un k-conflit $shift - reduce$ si il existe deux calculs de l'automate

$$\begin{array}{lll} \beta\alpha & \xrightarrow[reduce]{\varepsilon} & \beta T \xrightarrow[*]{u} S \\ \beta\alpha & \xrightarrow[shift]{x} & \beta\alpha x \xrightarrow[*]{v} S \end{array}$$

  tels que $First_k(u) = First_k(xv)$

- Il existe un k-conflit $reduce - reduce$ si il existe deux calculs de l'automate

$$\begin{array}{lll} \beta\alpha & \xrightarrow[reduce]{\varepsilon} & \beta T \xrightarrow[*]{u} S \\ \beta'\alpha' & \xrightarrow[reduce]{\varepsilon} & \beta' T' \xrightarrow[*]{u'} S \end{array}$$

  tels que $\beta\alpha = \beta'\alpha'$ et $First_k(u) = First_k(u')$

Grammaires augmentées

Problème : par exemple la grammaire $S \to Sa \mid a$, n'a aucun conflit mais admet des étapes de calcul pour lesquels l'axiome est seul en pile
➥problème de décision de l'arrêt

Définition

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, la **grammaire augmentée** de $\mathcal{G}$ est $\mathcal{G}' = (\Sigma, V \cup \{S'\}, S', R \cup \{S' \to S\})$ avec $S' \notin V$

Une grammaire est $LR(k)$ si sa grammaire augmentée n'a aucun k-conflit

Exemple. Grammaire augmentée pour les expressions

$$\begin{array}{lcl} E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid id \end{array}$$

$$\begin{array}{lcl} E' & \to & E \\ E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid id \end{array}$$

Grammaire $LR(k)$ - Remarques

Remarques sur les LR

- La hiérarchie des grammaires $LR(k)$ est stricte
- On ne peut pas décider s'il existe un k tel qu'une grammaire donnée soit $LR(k)$
- Tout langage déterministe peut être engendré par une grammaire $LR(1)$

Liens LL et LR

- Toute grammaire $LL(k)$ est $LR(k)$
- On peut décider si une grammaire $LR(k)$ est $LL(k)$
- On ne peut pas décider s'il existe un k tel qu'une grammaire donnée $LR(i)$ soit $LL(k)$

Construction d'un analyseur *LR*

Pour construire un analyseur, il faut savoir décider à tout moment :

- s'il faut appliquer un *shift* ou un *reduce*
- s'il s'agit de réduire, quelle règle appliquer
- quand s'arrêter et si l'arrêt note un succès ou un échec

➥ sans anticiper la lecture des symboles de l'entrée

Nous allons construire un automate qui va permettre de guider l'analyse en mémorisant à quel stade l'analyse en est

Analyseur $LR(k)$

Soit $\mathcal{G} = (\Sigma, V, S', R)$ une grammaire augmentée

L'**analyseur** $LR(k)$ de $\mathcal{G}$ est défini par

- un automate fini déterministe dit « automate des contextes » qui va assurer le contrôle
- une fonction *action* qui à un état de Q et un mot $u \in \Sigma^*$ avec $|u| \leq k$ associe une action *success*, *fail*, *shift*, ou une action $reduce_{T \to \alpha}$ pour les $T \to \alpha \in R$

Une **configuration** de l'analyseur est une paire $(\alpha, u) \in (\Sigma \cup V)^* \times \Sigma^*$ dans laquelle α est le contenu de la pile et u le mot restant à lire sur l'entrée

Dans une configuration (α, u) l'analyseur $LR(k)$ effectue le pas $action(goto(q_0, \alpha), First_k(u))$

Architecture d'un analyseur syntaxique *LR*

L'algorithme d'analyse syntaxique *LR* va toujours s'appuyer sur

- une entrée : sur laquelle on lit des mots de l'alphabet terminal
- une sortie
- une table d'analyse syntaxique permettant de décider l'action à accomplir, et de connaitre la fonction de transition *goto* de l'automate des contextes
- une pile

Dans la définition précédente, on indique que dans une configuration (α, u) l'analyseur $LR(k)$ effectue le pas $action(goto(q_0, \alpha), First_k(u))$

- pour éviter de recalculer les *goto* à chaque pas, il suffirait de stocker l'état courant dans la pile
- nous verrons que par construction de l'automate des contextes, il suffit de garder l'état courant

Analyse syntaxique

LR(0)

0−item ou item

Un **0-item** ou **item** d'une grammaire $\mathcal{G} = (\Sigma, V, S, R)$ est une règle de R dans la partie droite de laquelle un point a été inséré : intuitivement, il sépare le début de la partie droite qui a déjà été reconnue de la fin de la partie droite que l'on espère trouver

Exemple

La règle	T	$\rightarrow$	$\alpha\beta\gamma$
donne	T	$\rightarrow$	$\bullet\alpha\beta\gamma$
	T	$\rightarrow$	$\alpha\bullet\beta\gamma$
	T	$\rightarrow$	$\alpha\beta\bullet\gamma$
	T	$\rightarrow$	$\alpha\beta\gamma\bullet$

➥$\alpha\beta\bullet\gamma$ indique que l'on a trouvé sur l'entrée une chaîne qui se dérive de $\alpha\beta$ et que l'on espère trouver une chaîne qui se dérive de γ

➥$\alpha\beta\gamma\bullet$ indique que l'on peut envisager de réduire $\alpha\beta\gamma$ en T

Remarque $T \rightarrow \varepsilon$ a pour seul item $T \rightarrow \bullet$

0−item - Exemple

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & aSb \mid ab
\end{array}
$$

Les 0−items de la grammaire

$$
\begin{array}{lcl}
S' & \rightarrow & \bullet S \\
S' & \rightarrow & S\bullet \\
S & \rightarrow & \bullet aSb \\
S & \rightarrow & a \bullet Sb \\
S & \rightarrow & aS \bullet b \\
S & \rightarrow & aSb\bullet \\
S & \rightarrow & \bullet ab \\
S & \rightarrow & a \bullet b \\
S & \rightarrow & ab\bullet
\end{array}
$$

item valide

Tout les préfixes d'un proto-mot ne peuvent pas apparaître sur la pile d'un analyseur *shift* − *reduce* car l'analyseur doit réduire les éléments en sommet de pile quand c'est possible avant de décaler

Un item $T \to \alpha \bullet \beta$ est valide dans le contexte γ s'il existe une dérivation droite

$$S' \underset{*}{\to} \delta T u \to \delta\alpha\beta u \text{ avec } \gamma = \delta\alpha$$

On note $I_0(\gamma)$ l'ensemble des items valides dans le contexte γ

De plus

- si β est vide, alors un *reduce* est utile
- si β n'est pas vide, un *shift* est utile

Clôture d'un ensemble d'items

Soit I un ensemble d'items, **la clôture de I**, notée $clos(I)$ est le plus petit ensemble d'items tel que :

- $I \subseteq clos(I)$
- si $T \to \alpha \bullet U\beta \in clos(I)$ et $U \to \gamma$ alors $U \to \bullet\gamma \in clos(I)$

Intuition. Si nous considérons $\alpha \bullet U\beta$ alors nous espérons trouver un mot qui se dérive de $U\beta$, le préfixe de ce mot sera obtenu par dérivation de U

Exemple

$$
\begin{array}{lcl}
S' & \to & S \\
S & \to & A \mid B \\
A & \to & aAb \mid c \\
B & \to & aBbb \mid d
\end{array}
$$

$$
clos(\{S' \to \bullet S\}) = \{
\begin{array}{l}
S' \to \bullet S \\
S \to \bullet A \\
S \to \bullet B \\
A \to \bullet aAb \\
A \to \bullet c \\
B \to \bullet aBbb \\
B \to \bullet d
\end{array}
\}
$$

Clôture d'un ensemble d'items - Exemple

Exemple de la grammaire augmentée pour les expressions

$$
\begin{array}{lcl}
E' & \rightarrow & E \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid id
\end{array}
$$

$$
\begin{array}{lll}
clos(\{E' \rightarrow \bullet E\}) & = \{ & E' \rightarrow \bullet E \\
 & & E \rightarrow \bullet E + T \\
 & & E \rightarrow \bullet T \\
 & & T \rightarrow \bullet T * F \\
 & & T \rightarrow \bullet F \\
 & & F \rightarrow \bullet (E) \\
 & & F \rightarrow \bullet id \\
 & \} & \\
clos(\{T \rightarrow T * \bullet F\}) & = \{ & T \rightarrow T * \bullet F \\
 & & F \rightarrow \bullet (E) \\
 & & F \rightarrow \bullet id \\
 & \} &
\end{array}
$$

Fonction *goto*

Soit I un ensemble d'items et x un symbole de $\Sigma \cup V$, alors

$$goto(I, x) = clos(\{T \to \alpha x \bullet \beta \mid T \to \alpha \bullet x\beta \in I\})$$

Intuition

Cette fonction va permettre de définir les transitions de l'automate dont les états seront des ensembles d'items

Exemple

$$\begin{array}{lcl} S' & \to & S \\ S & \to & A \mid B \\ A & \to & aAb \mid c \\ B & \to & aBbb \mid d \end{array}$$

$$\begin{array}{l} I = \{A \to \bullet aAb, B \to d\bullet\} \\ goto(I, a) = \{ \quad A \to a \bullet Ab \\ \qquad\qquad\quad A \to \bullet aAb \\ \qquad\qquad\quad A \to \bullet c \\ \qquad\qquad \} \end{array}$$

goto - Exemple pour les expressions

Notons $I = clos(\{E' \to \bullet E\})$

$$
\begin{aligned}
I \quad = \{ \quad & E' \to \bullet E \\
& E \to \bullet E + T \\
& E \to \bullet T \\
& T \to \bullet T * F \\
& T \to \bullet F \\
& F \to \bullet (E) \\
& F \to \bullet id \\
\}
\end{aligned}
$$

Calculons $goto(I, id)$

- ajouter $F \to id\bullet$
- calculer la clôture : rien à ajouter

$goto(I, id) = \{F \to id\bullet\}$

Calculons $goto(I, T)$

- ajouter $E \to T\bullet$, $T \to T \bullet *F$
- calculer la clôture : rien à ajouter

$goto(I, id) = \{E \to T\bullet, T \to T \bullet *F\}$

Calculons $goto(I, ()$

- ajouter $F \to (\bullet E)$
- calculer la clôture : $goto(I, () = \{F \to (\bullet E), E \to \bullet E + T, E \to \bullet T, T \to \bullet T * F, T \to \bullet F, F \to \bullet (E), F \to \bullet id\}$

Remarques

L'ensemble des items valides a des propriétés intéressantes que nous ne démontrerons pas. Pour tout $\gamma \in (\Sigma \cup V)^*$:

- $I_0(\gamma)$ est clos
- $goto(I_0(\gamma), x) \subseteq I_0(\gamma x)$

Automate des contextes

Nous allons construire un automate

- dont les états sont construits à partir des *items*
- dont les transitions sont données par la fonction *goto*

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, alors son automate des contextes $\mathcal{C} = (\Sigma \cup V, Q, q_0, F, goto)$ est tel que :

- Q un ensemble d'ensembles d'items
- $q_0 = clos(\{S' \to \bullet S\})$ (avec S' l'axiome de la grammaire augmentée de $\mathcal{G}$
- $F = Q$

Remarque. En fait, pour tout $\gamma \in (\Sigma \cup V)^*$, $I_0(\gamma) = goto(q_0, \gamma)$ en pratique on ne va pas utiliser tous les ensembles d'items possibles mais seulement ceux qui sont utiles : on prend $Q =$ l'ensemble canonique d'ensemble d'items

Construction de l'ensemble canonique d'ensembles d'items

Cet ensemble canonique C permet de construire directement l'automate des contextes (ou automates $LR(0)$) avec l'ensemble d'états adéquat, c'est le plus petit ensemble tel que

- $clos(\{S' \rightarrow \bullet S\}) \in C$
- $\forall x \in \Sigma \cup V, \forall I \in C, (goto(I, x) \neq \emptyset) \Rightarrow (goto(I, x) \in C)$

L'algorithme de construction de cet ensemble part de $clos(\{S' \rightarrow \bullet S\})$ et sature l'ensemble par itérations successives

➥Tous les états de l'automate des contextes sont accessibles

Automate des contextes - Exemple

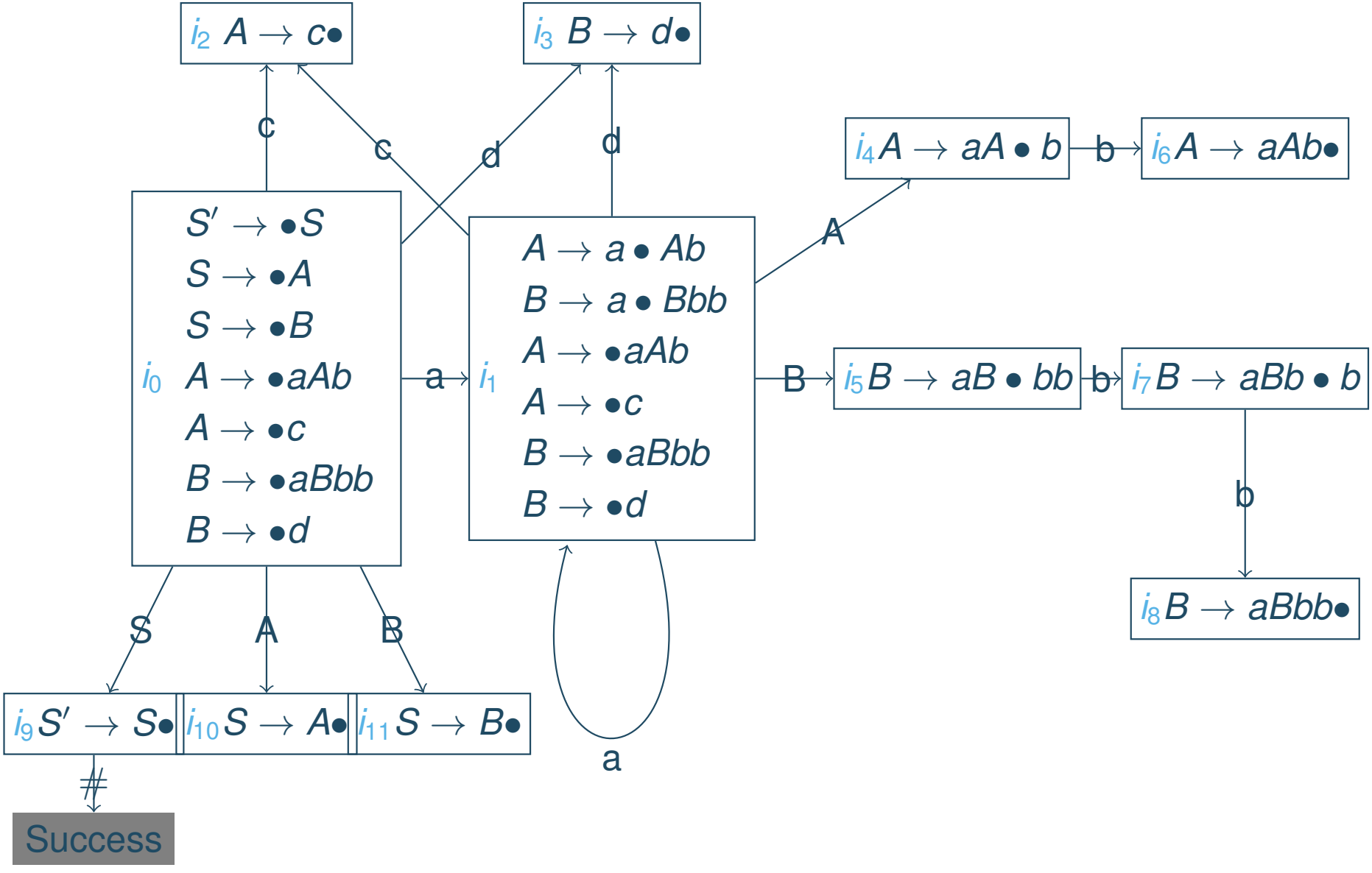


Table des actions de l'analyse *SLR*

Soit I un ensemble d'items, $x \in \Sigma$, et $u \in \Sigma \cup \{\varepsilon\}$

- $action(I, x) = shift$ si $\exists T \to \alpha \bullet x\beta \in I$
  ➥ on peut ajouter x à la construction de partie droite de règle en cours
- $action(I, u) = reduce_{T \to \alpha}$ si $T \to \alpha\bullet \in I$, $u \in Follow(T)$ et $T \neq S'$
  ➥ On a analysé un mot qui se dérive à partir de α et le suffixe u peut suivre la dérivation de la variable T
- $action(I, \varepsilon) = success$ si $S' \to S\bullet \in I$
  ➥ Les réductions successives ont mené à l'axiome
- $action(I, u) = fail$ sinon
  ➥ Dans les autres cas, on est bloqué

Table des actions de l'analyse *SLR* - Exemple (1/2)

Rappel de la grammaire

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & A \mid B \\
A & \rightarrow & aAb \mid c \\
B & \rightarrow & aBbb \mid d
\end{array}
$$

Calcul des *Follow*

$$
\begin{array}{lcl}
Follow(S') & = & \{\#\} \\
Follow(S) & = & \{\#\} \\
Follow(A) & = & \{\#, b\} \\
Follow(B) & = & \{\#, b\}
\end{array}
$$

Une grammaire est *SLR* s'il n'y a pas de conflit dans la table *action* de son analyseur *SLR*

Automate des contextes - Rappel

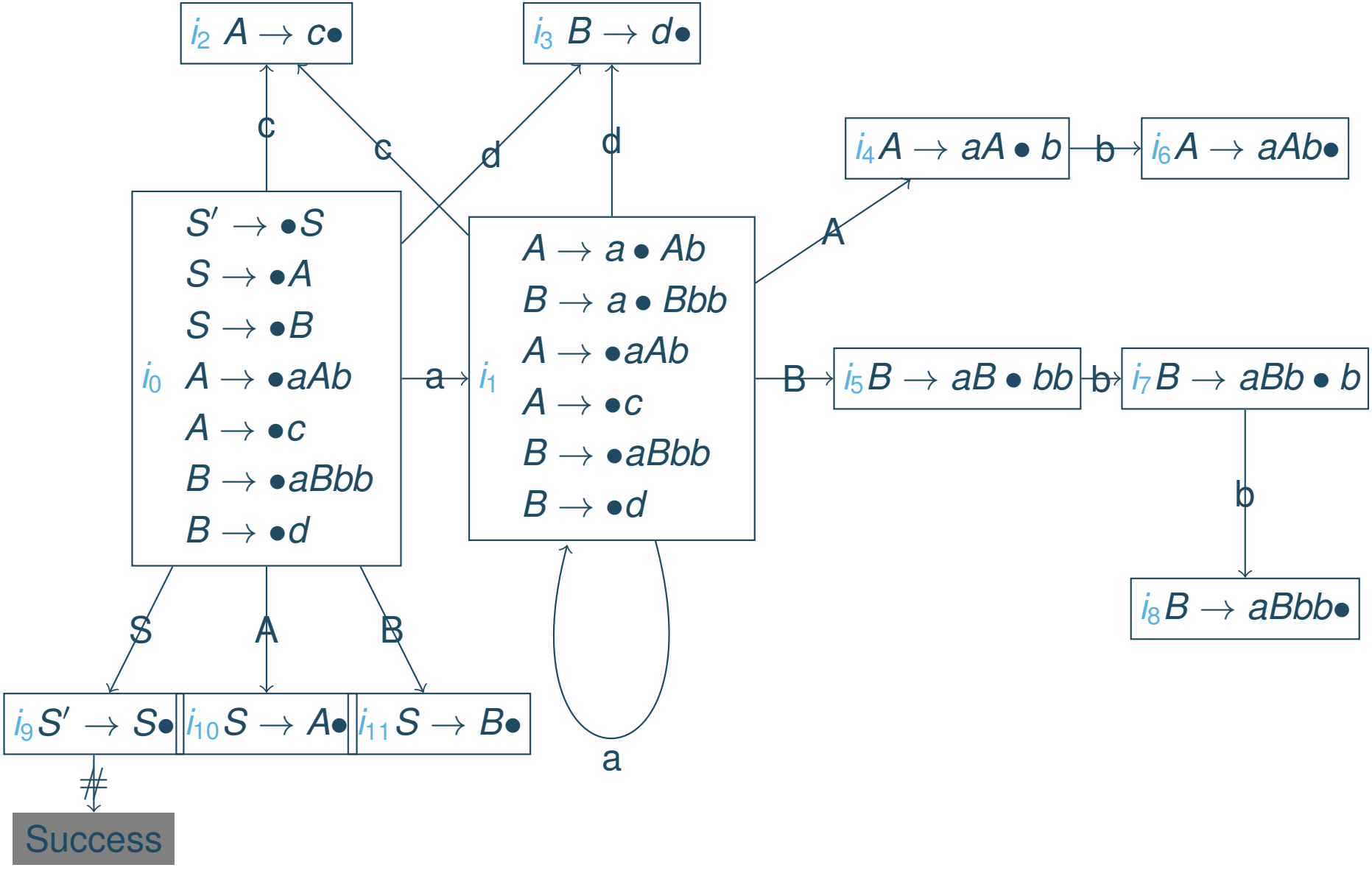


Table des actions de l'analyse *SLR* - Exemple (2/2)

	a	b	c	d	$\#$
i_0	*shift*		*shift*	*shift*	
i_1	*shift*		*shift*	*shift*	
i_2		$reduce_{A \to c}$			$reduce_{A \to c}$
i_3		$reduce_{B \to d}$			$reduce_{B \to d}$
i_4		*shift*			
i_5		*shift*			
i_6		$reduce_{A \to aAb}$			$reduce_{A \to aAb}$
i_7		*shift*			
i_8		$reduce_{B \to aBbb}$			$reduce_{B \to aBbb}$
i_9					*success*
i_{10}					$reduce_{S \to A}$
i_{11}					$reduce_{S \to B}$

Analyse *SLR*

Exemple du mot *adbb*

Configuration

Pile α	*Mot* u	*action* $action(goto(i_0, \alpha), First(u))$
ε	*adbb*#	*shift*
a	*dbb*#	*shift*
ad	*bb*#	$reduce_{B \to d}$
aB	*bb*#	*shift*
aBb	*b*#	*shift*
aBbb	#	$reduce_{B \to aBbb}$
B	#	$reduce_{S \to B}$
S	#	*success*

Les limites de l'analyse *SLR*

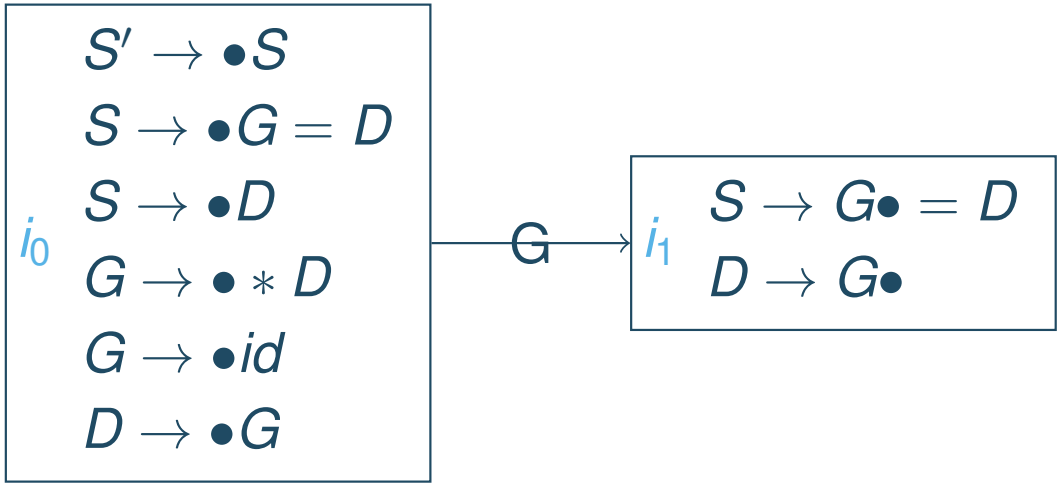
Exemple de grammaire (toujours du Dragon Book) non ambigüe

$S' \rightarrow S$

$S \rightarrow G = D \mid D$

$G \rightarrow *D \mid id$

$D \rightarrow G$



Dans la table on a pour la case $action(i_1, =)$, *shift* mais aussi $reduce_{D \rightarrow G}$ car $= \in Follow(D)$, il y a donc un conflit

➥ L'analyse *SLR* est donc simple à mettre en œuvre mais pas assez puissante pour toutes les grammaires

Analyse syntaxique

Analyse $LR(1)$

Limites de l'analyse *SLR*

Exemple de grammaire (toujours du Dragon Book)

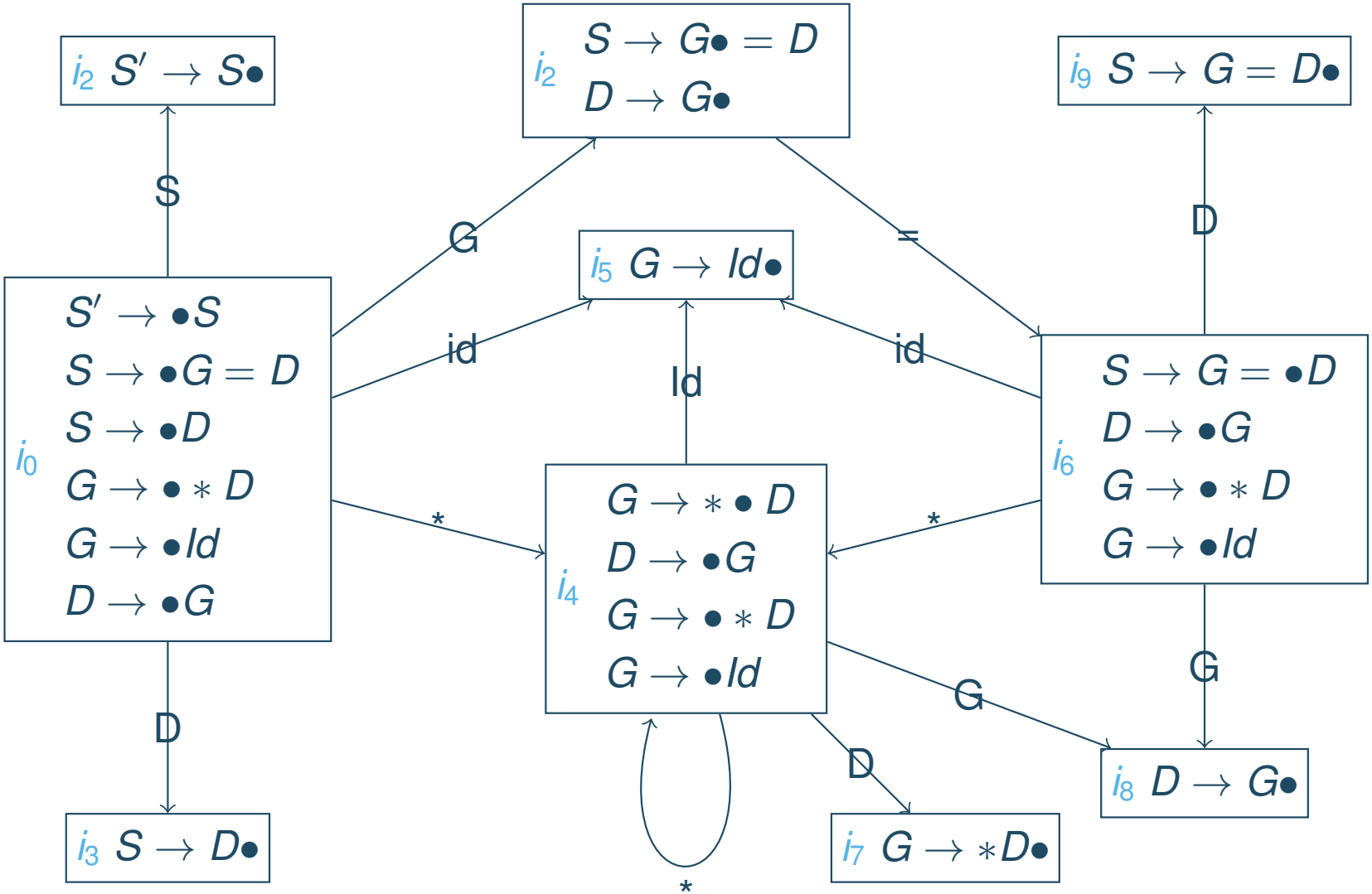
$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & G = D \mid D \\
G & \rightarrow & *D \mid id \\
D & \rightarrow & G
\end{array}
$$

La grammaire est non ambiguë (rappel : c'est indécidable), exemple de deux dérivations

$$S' \rightarrow S \rightarrow D \rightarrow G \rightarrow *D \rightarrow *G \rightarrow *id$$

$$S' \rightarrow S \rightarrow G = D \rightarrow G = G \rightarrow G = id \rightarrow *D = id \rightarrow *G = id \rightarrow *id = id$$

Exemple


i_2 $S' \to S\bullet$
i_2 $S \to G\bullet = D$
$D \to G\bullet$
i_9 $S \to G = D\bullet$
S
G
=
D
i_5 $G \to Id\bullet$
id
Id
id
i_0 $S' \to \bullet S$
$S \to \bullet G = D$
$S \to \bullet D$
$G \to \bullet * D$
$G \to \bullet Id$
$D \to \bullet G$
i_6 $S \to G = \bullet D$
$D \to \bullet G$
$G \to \bullet * D$
$G \to \bullet Id$
*
*
i_4 $G \to * \bullet D$
$D \to \bullet G$
$G \to \bullet * D$
$G \to \bullet Id$
G
D
G
D
i_3 $S \to D\bullet$
i_7 $G \to *D\bullet$
i_8 $D \to G\bullet$
*

Exemple

Nous avons $= \in \mathit{Follow}(D)$, donc dans la table d'action nous allons trouver

$$\mathit{action}(i_2, =) = \begin{cases} \mathit{shift} \\ \mathit{reduce}_{D \to G} \end{cases}$$

donc la table contient un conflit et la grammaire n'est pas $LR(0)$

➥ L'analyse *SLR* est donc simple à mettre en œuvre mais pas assez puissante pour toutes les grammaires

Nous allons voir des analyses plus puissantes : elles ne permettent pas d'analyser sans conflit toutes les grammaires, mais sont en général suffisantes pour les langages de programmation

Principes de l'extension

- Afin d'augmenter la puissance de l'analyse, on va désormais s'autoriser l'exploration d'un symbole en avant pour prendre une décision sur l'action à appliquer
- Il s'agit d'un « lookahead » comme dans le cas des *LL*
- On va généraliser la notion d'item sous la forme

$$[T \to \alpha \bullet \beta, x]$$

  en considérant pour x les symboles terminaux qui peuvent suivre T à ce stade de l'analyse, et donc $x \in \mathit{Follow}(T)$
- Le comportement de l'analyseur va être modifié en conséquence : dans un état contenant $[T \to \alpha\bullet, x]$ la réduction $T \to \alpha$ est appliquée si et seulement si le prochain symbole d'entrée est x

Calcul des $1-$items

La définition des $0-$items est étendue pour inclure une information supplémentaire : un symbole terminal (ou fin de chaîne)

Définition Les 1-items pour la grammaire $\mathcal{G} = (\Sigma, V, S, R)$ sont les

$$[T \to \alpha \bullet \beta, x] \text{ avec } T \to \alpha\beta \in R \text{ et } x \in \Sigma \cup \{\#\}$$

Remarques

- Si $\beta \neq \varepsilon$, le symbole x n'a pas d'effet, il n'influence donc pas les actions *shift*
- Si $\beta = \varepsilon$ alors l'action $reduce_{T \to \alpha}$ ne sera effectuée que dans une configuration (γ, v) avec $x = First(v)$
- Cette méthode va générer un grand nombre d'items

1−item valide

Un 1-item $[T \to \alpha \bullet \beta, x]$ est valide dans le contexte γ s'il existe dans $\mathcal{G}$ une dérivation droite (on note $I_1(\gamma)$ l'ensemble des 1−items valides pour γ)

$$S' \underset{*}{\to} \delta T u \to \delta\alpha\beta u \quad \text{avec} \quad \gamma = \delta\alpha$$

et $x = First(u)$

ou $x = \#$ en pratique

Exemple

Grammaire

$$\begin{array}{lcl} S' & \to & S \\ S & \to & A \mid B \\ A & \to & aAb \mid c \\ B & \to & aBbb \mid d \end{array}$$

Dérivation

$S' \underset{*}{\to} aAb \to aaAbb \to aaaAbbb$

Par exemple $[A \to a \bullet Ab, b]$ est valide dans le contexte $\gamma = aaa$ car on a

$$\begin{array}{lcll} S' & \underset{*}{\to} & \delta T u & \to \delta\alpha\beta u \\ S' & \underset{*}{\to} & aaAbb & \to aaaAbbb \end{array}$$

avec $\gamma = aaa$ et $b \in First(bb)$

Clôture d'un ensemble de 1−items

Soit I un ensemble d'1−items, **la clôture de I**, notée $clos(I)$ est le plus petit ensemble d'items tel que :

- $I \subseteq clos(I)$
- si $[T \to \alpha \bullet U\beta, x] \in clos(I)$ et $U \to \delta$ et $y \in First(\beta x)$ alors $[U \to \bullet\delta, y] \in clos(I)$

Pourquoi ?

Pour $[T \to \alpha \bullet U\beta, x]$ valide pour un contexte γ, il existe une dérivation droite

$$S \underset{*}{\to} \eta T x u \to \eta\alpha U\beta x u \text{ avec } \gamma = \eta\alpha$$

donc $[U \to \bullet\delta, y]$ est valide dans le contexte γ avec $y \in First(\beta x)$

$$S \underset{*}{\to} \eta T x u \to \eta\alpha U\beta x u = \gamma U\beta x u \to \gamma\delta\beta x u$$

Clôture d'un ensemble de $1-$items - Exemple

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & G = D \mid D \\
G & \rightarrow & *D \mid id \\
D & \rightarrow & G
\end{array}
$$

$$
\begin{array}{rcll}
clos(\{[S' \rightarrow \bullet S, \#]\}) & = \{ & [S' \rightarrow \bullet S, \#], \\
 & & [S \rightarrow \bullet G = D, \#], \\
 & & [S \rightarrow \bullet D, \#], \\
 & & [G \rightarrow \bullet * D, =], \\
 & & [G \rightarrow \bullet id, =], \\
 & & [D \rightarrow \bullet G, \#] \\
 & & [G \rightarrow \bullet * D, \#], \\
 & & [G \rightarrow \bullet id, \#] \\
 & \} &
\end{array}
$$

Fonction *goto*

Soit I un ensemble d'1−items et x un symbole de $\Sigma \cup V$, alors

$$goto(I, x) = clos(\{[T \to \alpha x \bullet \beta, y] \mid [T \to \alpha \bullet x\beta, y] \in I\})$$

Remarques

L'ensemble des 1−items valides a les mêmes propriétés intéressantes que I_0 que nous ne démontrerons pas. Pour tout $\gamma \in (\Sigma \cup V)^*$:

- $I_1(\gamma)$ est clos
- $goto(I_1(\gamma), x) \subseteq I_1(\gamma x)$

Automate des contextes

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire, alors son automate des contextes $\mathcal{C}_1 = (\Sigma \cup V, Q, q_0, F, goto)$ est tel que :

- Q un ensemble d'ensembles d'$1-$items
- $q_0 = clos(\{[S' \to \bullet S, \#]\})$ (avec S' l'axiome de la grammaire augmentée de $\mathcal{G}$)
- $F = Q$

On ne considère par la suite que les états accessibles

Remarque. L'automate $\mathcal{C}_1$ calcule les $1-$items valides, pour tout $\gamma \in (\Sigma \cup V)^*$, $I_1(\gamma) = goto(q_0, \gamma)$

Automate des contextes - Exemple $LR(1)$

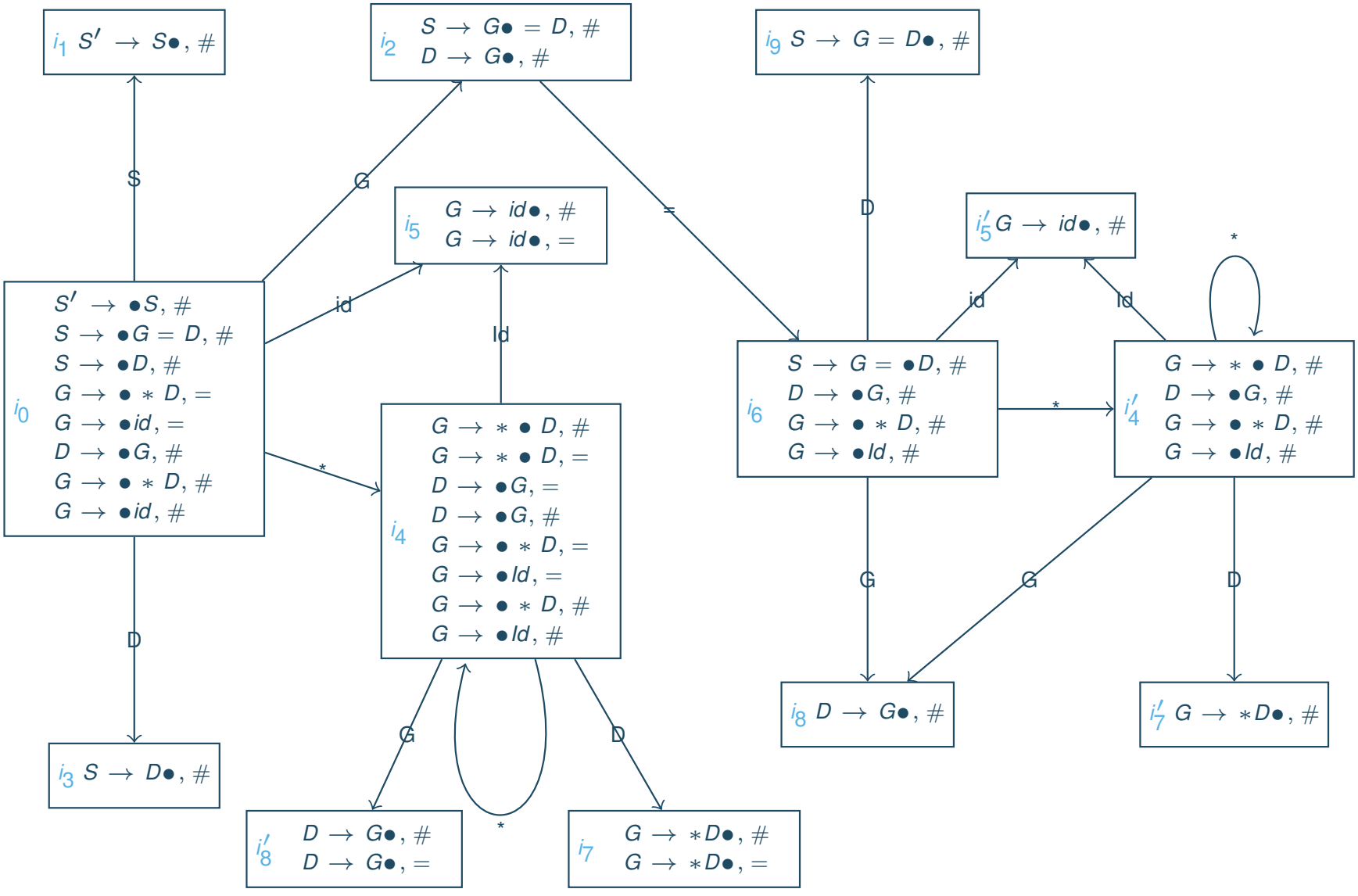


Table des actions

Soit I un ensemble d'1−items, $x \in \Sigma$, et $y \in \Sigma \cup \{\#\}$

- $action(I, x) = shift$ si $\exists [T \to \alpha \bullet x\beta, y] \in I$
- $action(I, y) = reduce_{T \to \alpha}$ si $[T \to \alpha\bullet, y] \in I$ et $T \neq S'$
- $action(I, \#) = success$ si $[S' \to S\bullet, \#] \in I$
- $action(I, u) = fail$ sinon

La grammaire $\mathcal{G}$ est $LR(1)$ si et seulement si il n'y a pas de conflit dans la table action de son analyseur $LR(1)$

Table des actions - Exemple

i_2 | $S \to G\bullet = D, \#$
$D \to G\bullet, \#$

Comme $S \to G\bullet = D, x \in I_2$ avec $x = \#$

$$shift \in action(I_2, =)$$

Est-ce qu'une réduction s'applique ?

La définition indique

$$action(I_2, =) = reduce_{T \to \alpha} \text{ si } [T \to \alpha\bullet, =] \in I, \text{ et } T \neq S'$$

ce qui n'est pas le cas ici puisque l'item qui contient $D \to G\bullet$ en première composante contient $\#$ en deuxième composante

➥ Le conflit est donc levé

Remarques

Comparaison des analyseurs $LR(0)$ et $LR(1)$

- Les actions *shift* sont les mêmes dans les deux cas
- Les actions *reduce* de l'analyseur $LR(1)$ sont des actions *reduce* de l'analyseur $LR(0)$
- Si $[T \to \alpha \bullet \beta, x] \in I_1(\gamma)$ alors $T \to \alpha \bullet \beta \in I_0(\gamma)$ et $x \in \mathit{Follow}(T)$
- Si $T \to \alpha \bullet \beta \in I_0(\gamma)$ alors $\exists x \in \Sigma \cup \{\#\}$ tel que $[T \to \alpha \bullet \beta, x] \in I_1(\gamma)$

Problèmes

La classe *SLR* est intéressante mais trop faible pour des cas qui se rencontrent dans les langages de programmation

Les classes $LR(k)$ avec $k \geq 2$ vont conduire à une taille de table d'analyse beaucoup trop importante en raison des lookahead. Même la table $LR(1)$ aurait en pratique plusieurs milliers d'états pour les langages de programmation classiques

On utilise en pratique un compromis entre $LR(0)$ et $LR(1)$: *LALR*

Analyse syntaxique

Analyse *LALR*

Principe le l'analyse *LALR*

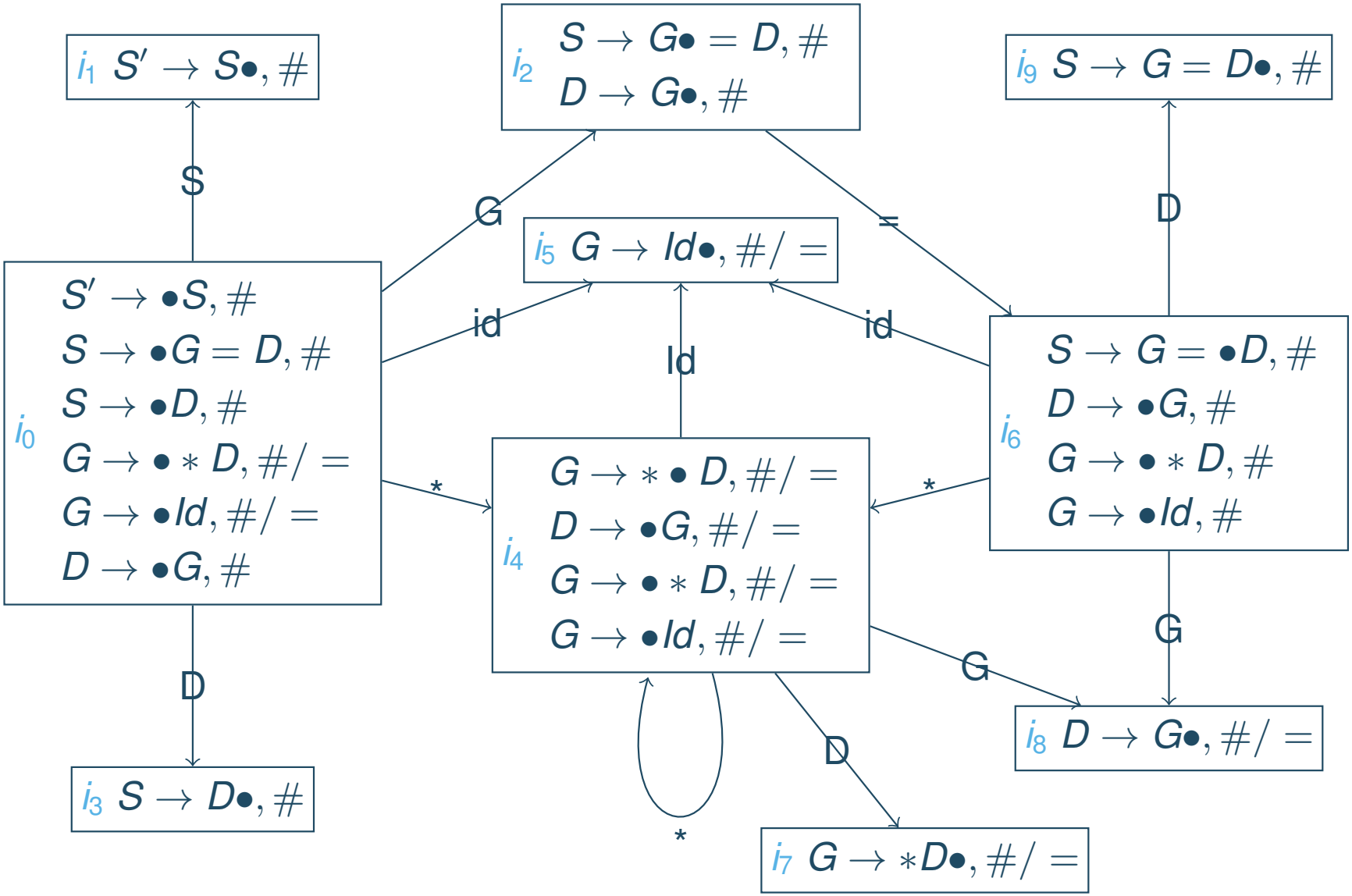
Look-Ahead LR est une classe d'analyseurs dont l'automate est obtenu en fusionnant les états de l'automate $LR(1)$ qui diffèrent uniquement par leur deuxième composante

L'automate va donc avoir autant d'états que l'automate $LR(0)$ (puisque les premières composantes des $1-$items sont les $0-$ *items*)

On conserve néanmoins l'information de look-ahead. Quand il n'y a pas de conflit dans la table $LR(1)$

- on peut montrer que l'on introduit pas de conflit shift/reduce
- on peut néanmois introduire des conflits reduce/reduce

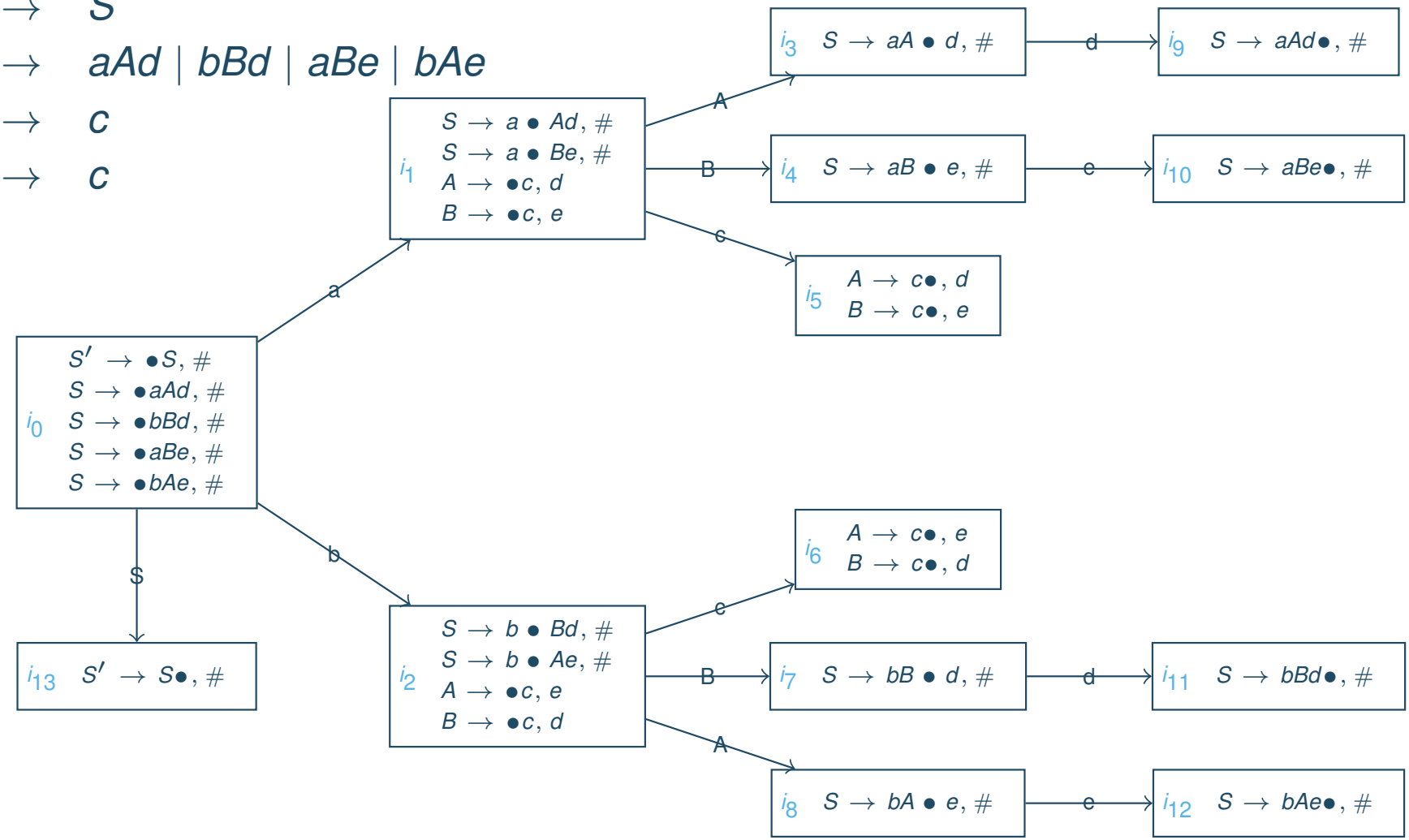
Exemple



➥ L'automate a la même structure l'automate $LR(0)$, la table des actions évite le conflit en i_2 comme l'analyse $LR(1)$

Exemple - Automate $LR(1)$

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\
A & \rightarrow & c \\
B & \rightarrow & c
\end{array}
$$



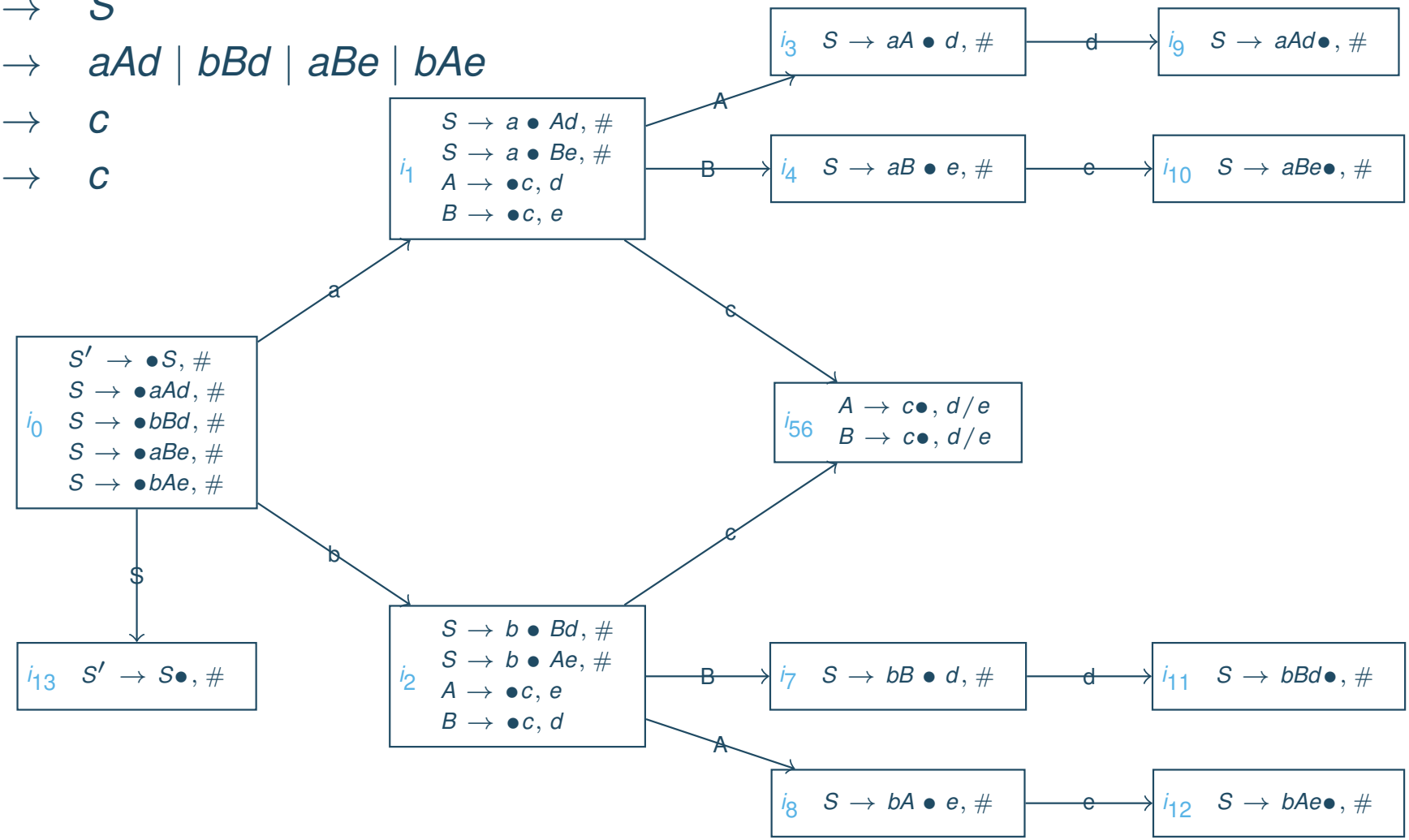
Exemple - Table d'actions

	a	b	c	d	e	$\#$
i_0	s	s				
i_1			s			
i_2			s			
i_3				s		
i_4					s	
i_5				$r_{A\to c}$	$r_{B\to c}$	
i_6				$r_{B\to c}$	$r_{A\to c}$	
i_7				s		
i_8					s	
i_9						$r_{S\to aAd}$
i_{10}						$r_{S\to aBe}$
i_{11}						$r_{S\to bBd}$
i_{12}						$r_{S\to bAe}$
i_{13}						*success*

➥ La grammaire est $LR(1)$

Exemple - Automate *LALR*

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\
A & \rightarrow & c \\
B & \rightarrow & c
\end{array}
$$



Exemple - Conflit dans l'état i_{56}

$$i_{56} \quad \begin{array}{l} A \to c\bullet, d/e \\ B \to c\bullet, d/e \end{array}$$

On modifie la table d'actions :

- comme $[A \to c\bullet, d] \in i_{56}$, $reduce_{A \to c} \in action(i_{56}, d)$
- comme $[B \to c\bullet, d] \in i_{56}$, $reduce_{B \to c} \in action(i_{56}, d)$

  ➥ conflit reduce/reduce
- comme $[A \to c\bullet, e] \in i_{56}$, $reduce_{A \to c} \in action(i_{56}, e)$
- comme $[B \to c\bullet, e] \in i_{56}$, $reduce_{B \to c} \in action(i_{56}, e)$

  ➥ conflit reduce/reduce

Cette grammaire (très simple) n'est pas *LALR*, cet exemple illustre l'introduction possible de conflits reduce-reduce lorsque l'on essaie de se ramener de $LR(1)$ à *LALR*

Analyse syntaxique

CYK - Thanks to Benoît Sagot

Pourquoi ?

Les langages de programmation sont définis par des grammaires algébriques qui ne sont pas ambiguës

Les algorithmes pour les CFG non ambiguës (« déterministes ») sont relativement simples (LR, LALR, . . .)

Le langage naturel est ambigu, les algorithmes que nous avons vus ne conviennent pas, cela a abouti au développement d'algorithmes d'analyse des grammaires algébriques générales (c'est-à-dire pas seulement déterministes) :

- CYK ➥nous allons voir un exemple de celui-ci
- Earley
- GLR

Principes

- L'algorithme de base Cocke – Younger – Kasami, alias CYK, résout un problème d'appartenance : ce n'est pas un algorithme d'analyse en soi
- Une implémentation de l'algorithme CYK est un identificateur, pas un analyseur mais il peut facilement être adapté pour devenir un algorithme d'analyse
- Le problème d'appartenance est simple : soient une grammaire algébrique et une chaîne d'entrée, la chaîne d'entrée appartient-elle au langage défini par la grammaire ?
- La grammaire doit être en forme normale de Chomsky. Pour rappel, les règles doivent avoir l'une des formes suivantes :
  - $S \rightarrow TU$, avec T et U des non terminaux
  - $S \rightarrow x$, avec x un terminal
  - $S \rightarrow \varepsilon$
- C'est un algorithme ascendant basé sur la programmation dynamique

Algorithme

Entrée : mot u, $n = |u|$
S de taille $n, n, |V|$
S initialisé à 0
for $i = 1$ à n **do**
  for all $T \to x \in R$ **do**
    if u[i] = x **then**
      S[i,i,T] = 1
    end if
  end for
end for

for k de 1 à $n - 1$ **do**
  for i de 1 à $n - k$ **do**
    j = i + k
    for all $T \to UX \in R$ **do**
      for r de i à $j - 1$ **do**
        if S[i,r,U] = S[r+1,j,X] = 1 **then**
          S[i,j,T]=1
        end if
      end for
    end for
  end for
end for

Idée de l'algorithme

Soient $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire en forme normale de Chomsky et $u = x_1 x_2 \dots x_n$ un mot

L'algorithme est itératif et construit une table $\mathcal{S}$ de n sur n tel que $\mathcal{S}_{i,j}$ soit l'ensemble des variables qui produisent $x_i \dots x_j$

- Étape 1 : traitement des facteurs de longueur 1. Pour chaque x_i, $\mathcal{S}_{i-1,i} = \{T \in V \mid T \to x_i\}$
- Étape 2 : traitement des facteurs de longueur 2. Pour chaque $x_i x_{i+1}$, $\mathcal{S}_{i-1,i+1} = \{T \in V \mid T \to x_i x_{i+1}\}$
- ...
-
- Étape n : traitement des facteurs de longueur n. Pour $x_1 \dots x_n$, $\mathcal{S}_{0,n} = \{T \in V \mid T \to x_1 \dots x_n\}$

Alors $u \in \mathcal{L}(\mathcal{G})$ si et seulement si $\mathcal{S}_{0,n} = \{S\}$

Table

$\mathcal{S}_{0,4}$			
$\mathcal{S}_{0,3}$	$\mathcal{S}_{1,4}$		
$\mathcal{S}_{0,2}$	$\mathcal{S}_{1,3}$	$\mathcal{S}_{2,4}$	
$\mathcal{S}_{0,1}$	$\mathcal{S}_{1,2}$	$\mathcal{S}_{2,3}$	$\mathcal{S}_{3,4}$
$x1$	$x2$	$x3$	$x4$

Exemple (1/8)

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
$S_{0,2}$	$S_{1,3}$	$S_{2,4}$	
$S_{0,1}$	$S_{1,2}$	$S_{2,3}$	$S_{3,4}$
c	b	b	a

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
$S_{0,2}$	$S_{1,3}$	$S_{2,4}$	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (2/8)

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
$S_{0,2}$	$S_{1,3}$	$S_{2,4}$	
$S_{0,1}$	$S_{1,2}$	$S_{2,3}$	$S_{3,4}$
c	b	b	a

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
{S,C}	$S_{1,3}$	$S_{2,4}$	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (3/8)

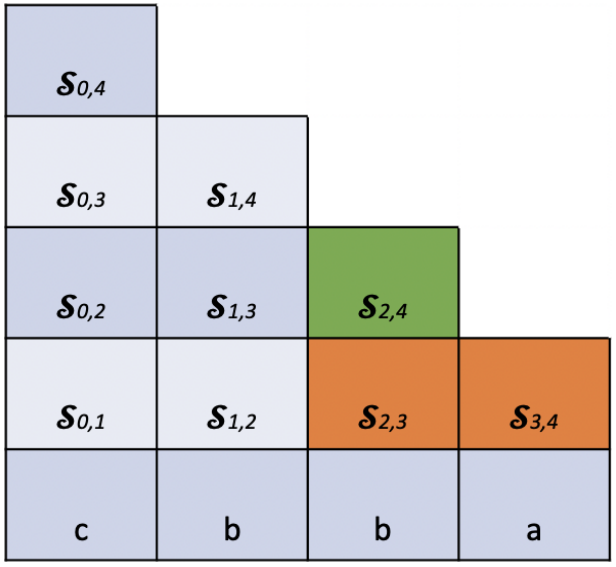
$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
$S_{0,2}$	$S_{1,3}$	$S_{2,4}$	
$S_{0,1}$	$S_{1,2}$	$S_{2,3}$	$S_{3,4}$
c	b	b	a

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
{S,C}	∅	$S_{2,4}$	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (4/8)

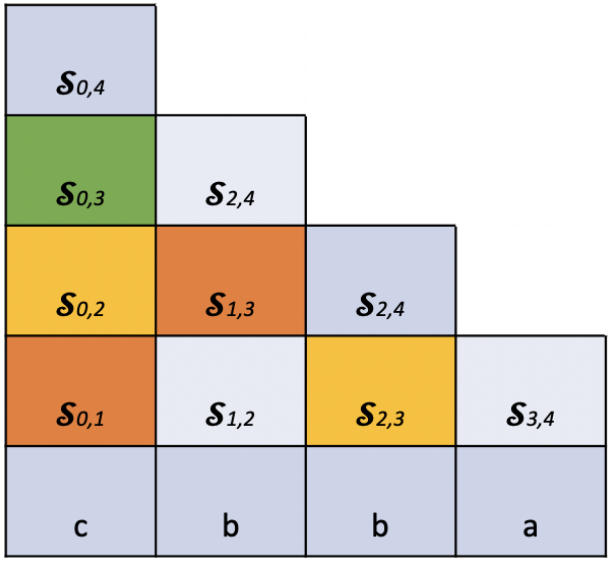
$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$

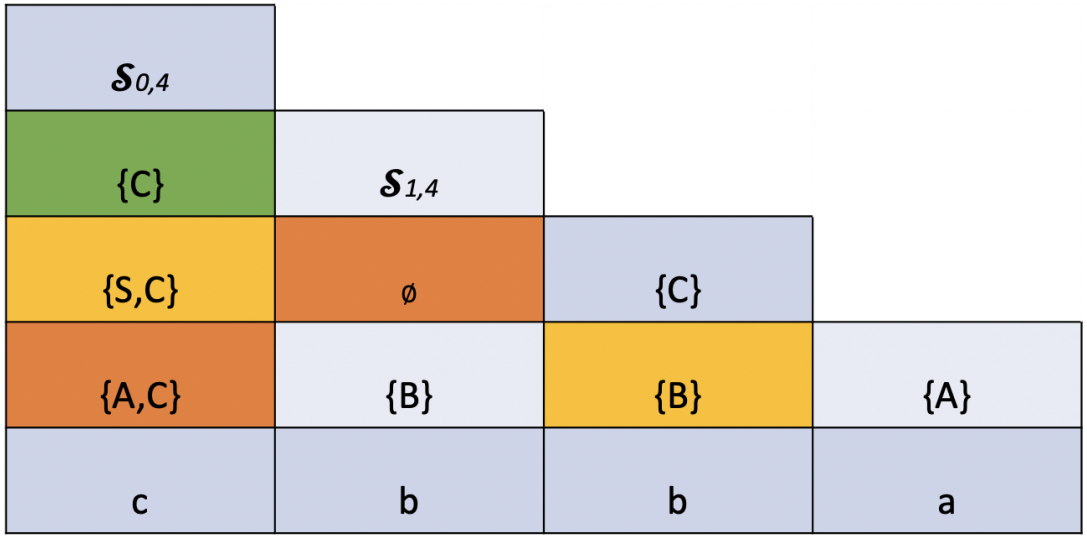


$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
{S,C}	∅	{C}	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (5/8)

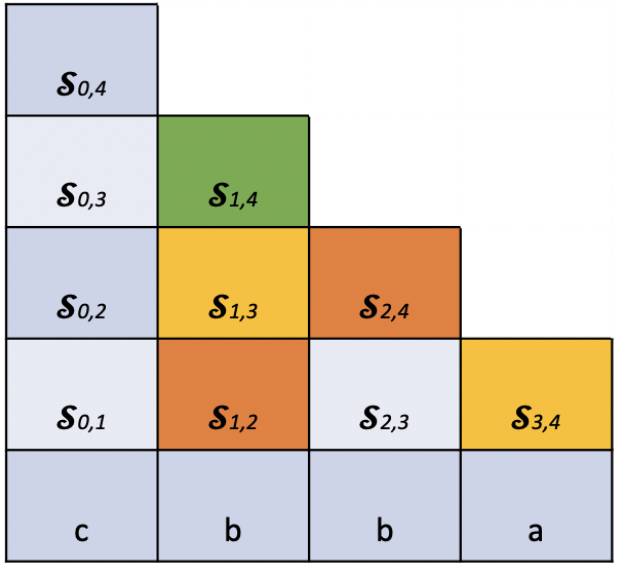
$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$





Exemple (6/8)

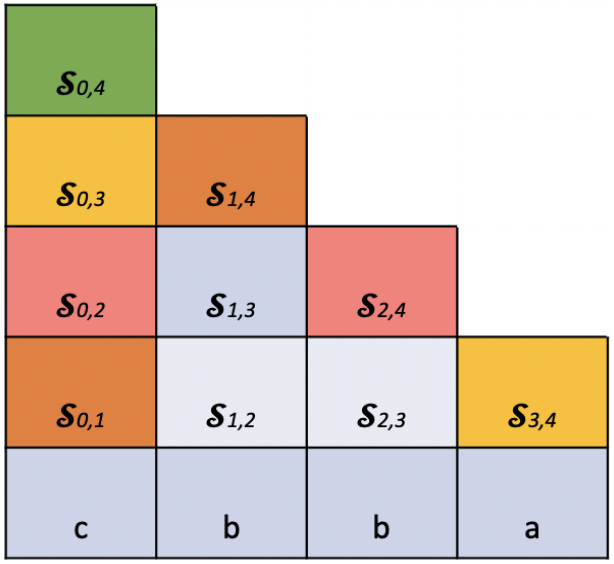
$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$



$S_{0,4}$			
{C}	{B}		
{S,C}	∅	{C}	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (7/8)

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$



{S,A,C}			
{C}	{B}		
{S,C}	∅	{C}	
{A,C}	{B}	{B}	{A}
c	b	b	a

Exemple (8/8)

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CC \mid a \mid c \\
B &\rightarrow BC \mid b \\
C &\rightarrow CB \mid BA \mid c
\end{aligned}
$$

$S_{0,4}$			
$S_{0,3}$	$S_{1,4}$		
$S_{0,2}$	$S_{1,3}$	$S_{2,4}$	
$S_{0,1}$	$S_{1,2}$	$S_{2,3}$	$S_{3,4}$
c	b	b	a

{S,A,C}			
{C}	{B}		
{S,C}	∅	{C}	
{A,C}	{B}	{B}	{A}
c	b	b	a