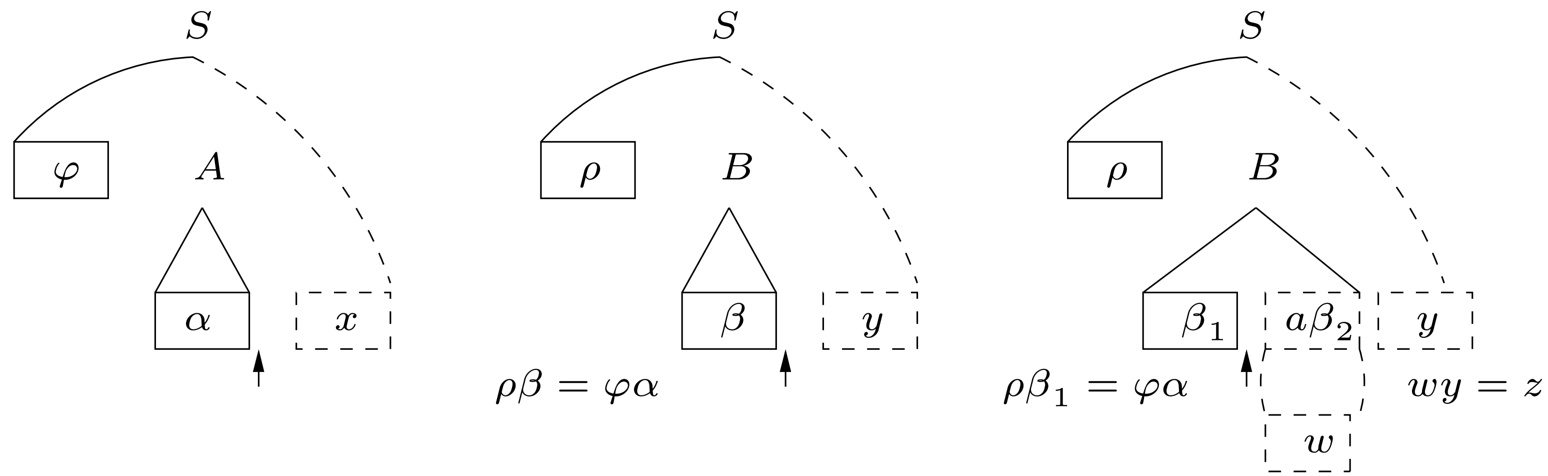
Grammaires LR-régulières (LRR)

Rappels sur les conflits LR(k)

– une grammaire est LR(k) si

$$\left.\begin{array}{l} S' \underset{\text{rm}}{\Rightarrow}^* \varphi A x \underset{\text{rm}}{\Rightarrow} \varphi \alpha x, \\ S' \underset{\text{rm}}{\Rightarrow}^* \rho B y \underset{\text{rm}}{\Rightarrow} \varphi \alpha z, \\ k : x = k : z \end{array}\right\} \text{implique que } \rho B y = \varphi A z.$$

– Exemples d'arbres en conflit



Grammaires LR-régulières

– K. Čulik & R. Cohen, LR-Regular Grammars – an Extension of LR(k) Grammars, J. of Computer and System Science **7**, 1973

– La couverture régulière d'un langage (algébrique) L_a est un langage régulier L_r tel que tout mot de L_a est un mot de L_r

  . par exemple, une couverture régulière de $a^n bc^n, n \geq 0$ est a^*bc^*, mais $b|abc|aabcc|...|a^{15}bc^{15}|a^{16}a^*bc^{16}c^*$ en est une plus *fine*

– Soit $\pi = \{R_1, R_2, \ldots, R_n\}$ une partition régulière de T^* : une grammaire est LR-régulière (on dit aussi LR-π) si

$$\left.\begin{array}{l} S' \underset{\mathrm{rm}}{\Rightarrow}^* \varphi A x \underset{\mathrm{rm}}{\Rightarrow} \varphi\alpha x, \\ S' \underset{\mathrm{rm}}{\Rightarrow}^* \rho B y \underset{\mathrm{rm}}{\Rightarrow} \varphi\alpha z, \\ x \equiv z \ (\mathbf{mod}\ \pi) \end{array}\right\} \text{ implique que } \rho B y = \varphi A z.$$

Autrement dit, il y a conflit si les couvertures régulières des contextes droits sont dans une même partition

Exemple de grammaire LR-régulière (1)

– Soit la grammaire $\mathcal{G}_1$

$S \rightarrow D \qquad S \rightarrow SsD$

$D \rightarrow T = V$

$T \rightarrow l \qquad T \rightarrow Tl$

$V \rightarrow \varepsilon \qquad \mathrm{V} \rightarrow Vl \qquad V \rightarrow Vs$

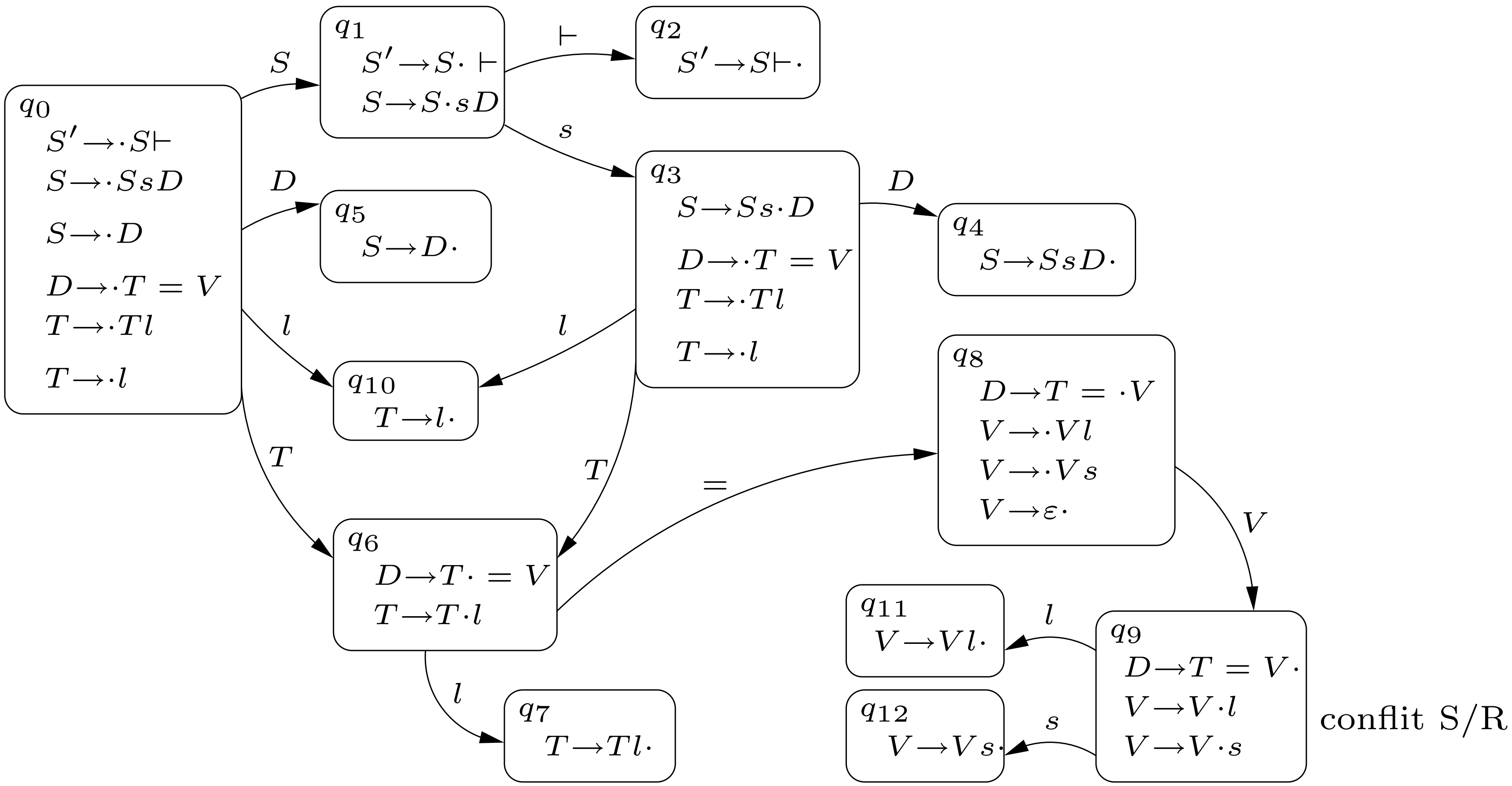
– Exemple de phrase (en simplifiant avec l= lettre, s = fin de ligne ou espace)

```
nom=farre prenom=jacques surnom=
institution=universite de nice sophia antipolis
          et cnrs
```

– Cette grammaire n'est pas LR(k) quel que soit k

Exemple de grammaire LR-régulière (2)

– Automate LR(0) de la grammaire $\mathcal{G}_1$



Exemple de grammaire LR-régulière (3)

– Conflit LR(0) entre le décalage de l ou s dans $(V \rightarrow V\,l, V \rightarrow V\,s)$, et la réduction $D \rightarrow T = V$

– Grammaire (augmentée par S') en forme EBNF

$$S' \rightarrow S\vdash \qquad S \rightarrow D(sD)^* \qquad D \rightarrow T = V \qquad T \rightarrow l^+ \qquad V \rightarrow (l|s)^*$$

. On voit que le langage à droite des décalages est $(l|s)^+\ (s\ \mathcal{L}(D))^*\vdash$ i.e., $R_1 = (l|s)^+\ (s\ l^+ = (l|s)^*)^*\vdash$

. Le langage à droite de la réduction est $(s\ \mathcal{L}(D)^*)^*\vdash$, i.e., $R_2 = (s\ l^+ = (l|s)^*)^*\vdash$

. Ces langages sont disjoints : $\pi = \{R_1, R_2\}$

. Ils ont un préfixe commun sl^+, ce qui montre bien qu'une fenêtre de longueur bornée ne peut décider

Principe de construction

– De même que chaque état d'un automate LR définit une classe d'équivalence pour le contexte gauche, les états de l'automate de la couverture régulière vont définir des classes d'équivalence pour le contexte droit

– L'idée est de calculer une couverture régulière pour le langage miroir : en voici une vision (très) simplifiée

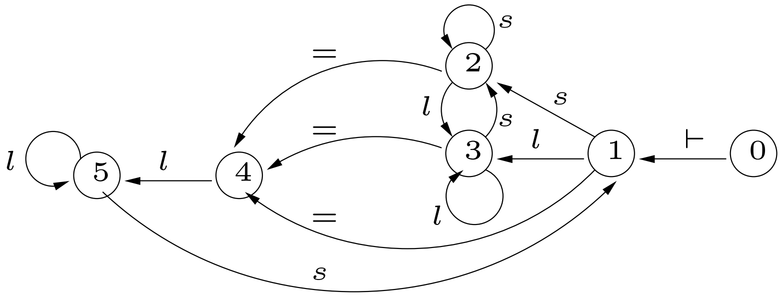
  . lors de l'analyse, une première passe lit l'entrée de **droite à gauche**, et remplace chaque terminal t par (q_i, t, q_j) si t fait passer de q_i à q_j (i.e. $q_i \xrightarrow{t} q_j$) dans l'automate de la couverture régulière inverse

  . La grammaire initiale G est transformée en G_π, où le vocabulaire (terminal et non terminal) est remplacé par des triplets (q_i, X, q_j) tel que $q_i \xrightarrow{w} q_j, X \Rightarrow^* w$: on peut démontrer que G_π est LR(0)

– Donc LRR a un temps d'analyse linéaire sur la longueur de l'entrée

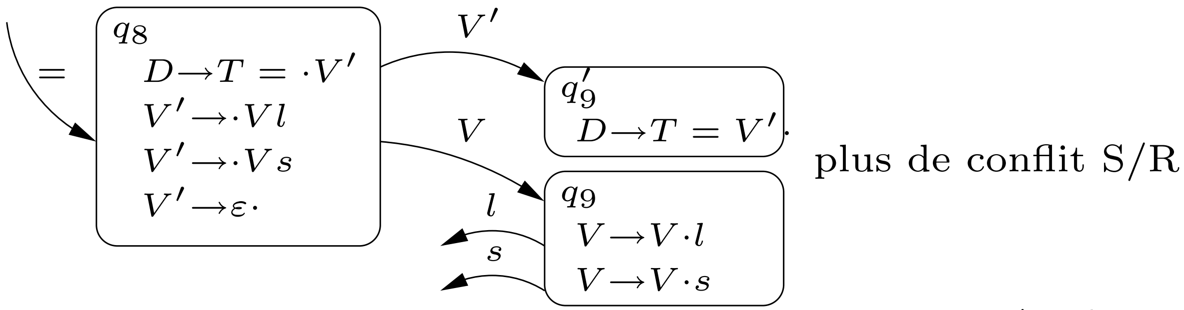
Exemple de construction (très simplifiée) pour G_1

– Automate de la couverture régulière du langage miroir de $\mathcal{L}(G_1)$:



– Les l de T ne donnent pas le même état (5) que ceux de V (3) : ce sont donc des terminaux différents dans G_π (idem pour les s de D et ceux dans V); enfin, il y a plusieurs non-terminaux isssus de V

– L'automate de la grammaire G_π ressemble donc à:



Classe LRR

– L'ensemble des langages LRR

. inclut strictement celui des langages déterministes

. et il est strictement inclus dans celui des langages non ambigus

– Cette définition est malheureusement théorique

. car il n'existe pas de couverture régulière minimale d'un langage non régulier

∗ par exemple, pour le langage $a^n bc^n, n \geq 0$, les couvertures régulières possibles sont $b + abc + a^2bc^2 + \cdots + a^{p-1}bc^{p-1} + a^p a^* bc^p c^*, p > 0$

∗ Il existe toujours $p' > p$ qui donne une couverture régulière plus précise

. si une grammaire est LRR pour une partition π de T^*, elle est aussi LRR pour une partition plus *fine*; l'inverse n'est pas vrai

. par conséquent, le problème de savoir si une grammaire est LRR est indécidable en général

Problèmes de la construction originale

– La lecture de l'entrée de droite à gauche : il faut donc avoir toute l'entrée (pas incrémental)

– La grammaire G_π est nettement plus grosse que G : et son automate LR(0) est très volumineux

Principe d'une construction alternative

– On construit un automate LR(0)

– Les conflits LR(0) peuvent être résolus s'il existe, pour chaque état avec un conflit, des couvertures régulières disjointes des langages à droite de chaque action en conflit dans cet état

. Il suffit alors de construire des automates finis qui reconnaissent des préfixes de ces couvertures régulières pour choisir l'action à effectuer

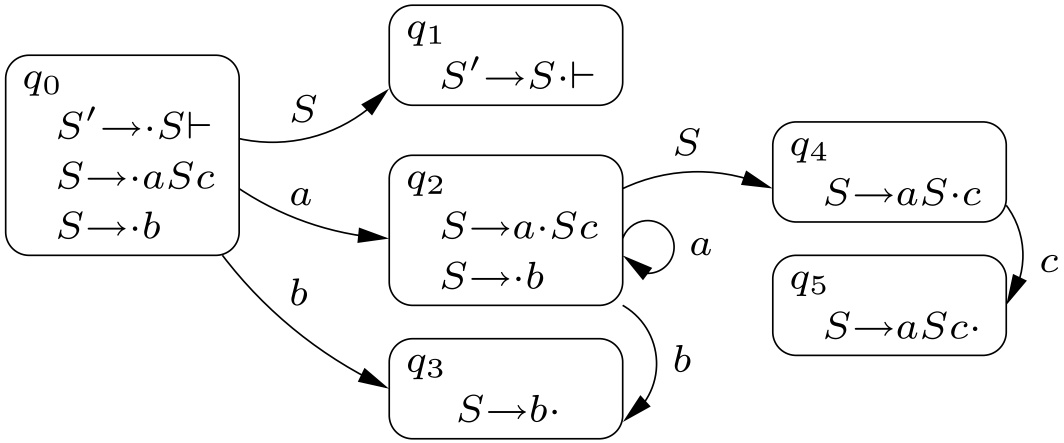
Sous-classes des grammaires LRR

- Construire des analyseurs pour un sous-ensemble des grammaires LRR présente un intérêt pratique certain, car ils permettent, pour un coût en général assez faible
  - . d'accepter certaines grammaires LR(k), $k \gg 1$, et des grammaires non LR
  - . ou de réduire significativement le nombre de conflits, ceux restant étant souvent dus à une ambiguïté
  - . défaut : on peut parcourir plusieurs fois une portion non bornée du contexte à droite des conflits, et la linéarité n'est pas garantie
- Le problème est de calculer des couvertures régulières aussi précises que possible dans un temps fini
- LR(k), LALR(k) et SLR(k) sont des cas particuliers de LRR

Calcul simple d'une couverture régulière - I

– T. Baker, Extending lookahead for LR parsers, J. of Computer and System Science **22**, 1981

– Soit la grammaire $\mathcal{G}_2$: $S \to aSc \mid b$, de $\mathcal{L}_2 = a^n b c^n$, et son automate LR(0)

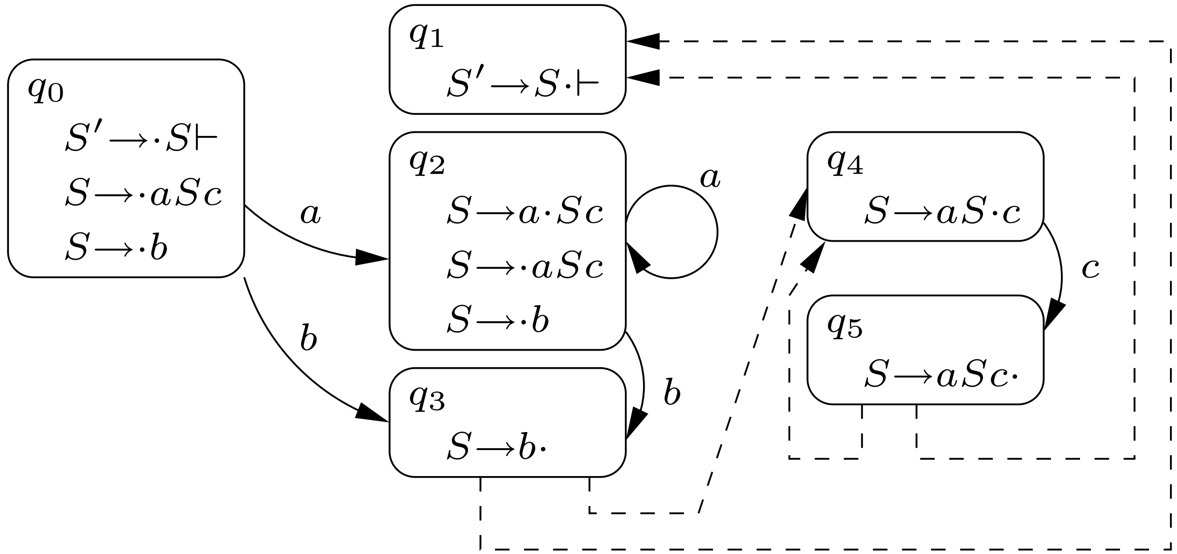


– On supprime les transitions sur les non-terminaux

– On ajoute des transitions sur ε de q contenant $A \to \alpha \cdot$ vers q' contenant $B \to \beta A \cdot \gamma$ tels que $q'' \xrightarrow{\alpha} q$ et $q'' \xrightarrow{A} q'$

Calcul simple d'une couverture régulière - II

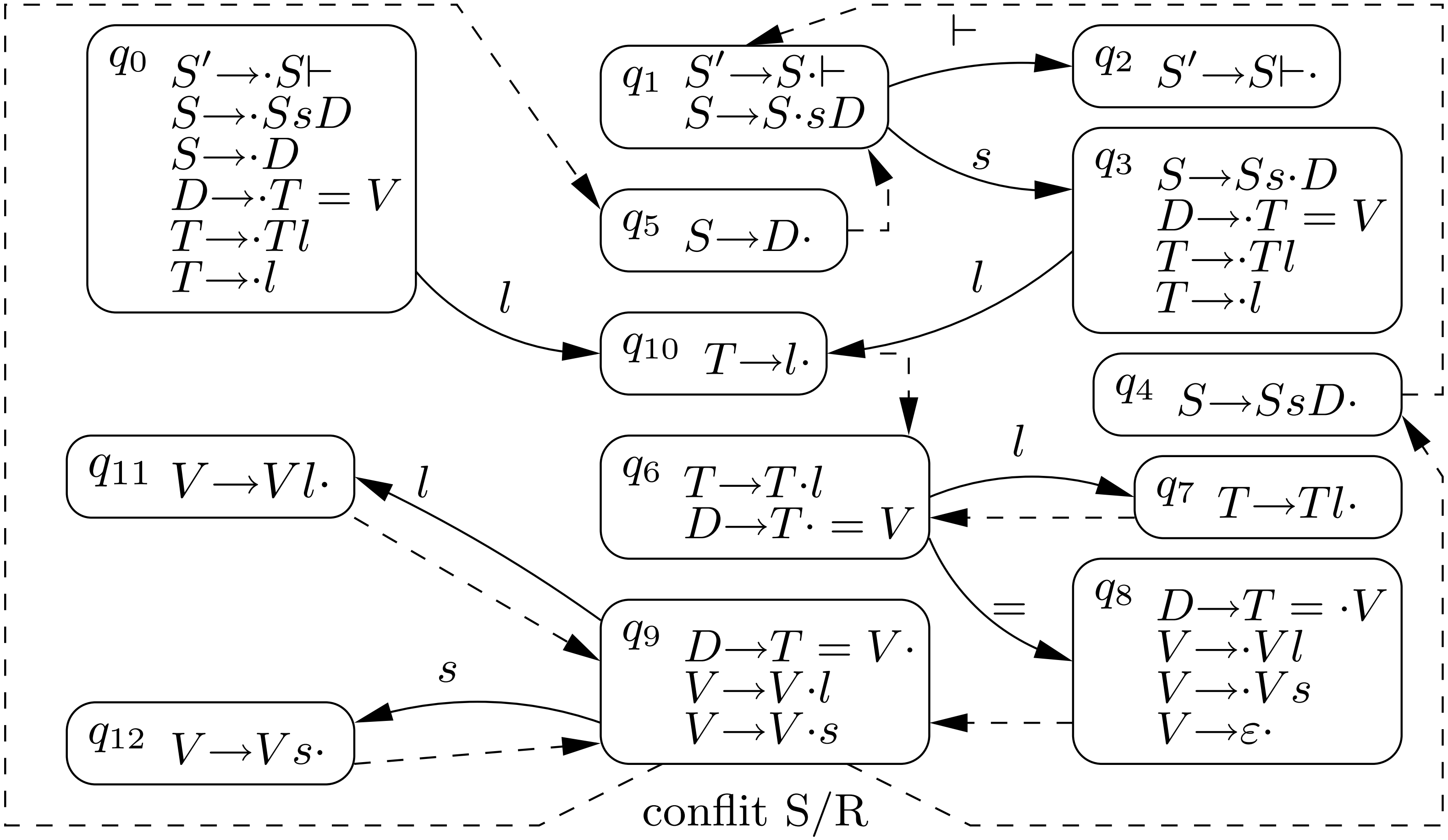
– Donc, l'automate LR(0) modifié de $\mathcal{G}_2$ donne



– On peut déterminiser cet automate

– Langage reconnu : $a^*bc^*\vdash$

Calcul simple d'une couverture régulière pour $\mathcal{G}_1$



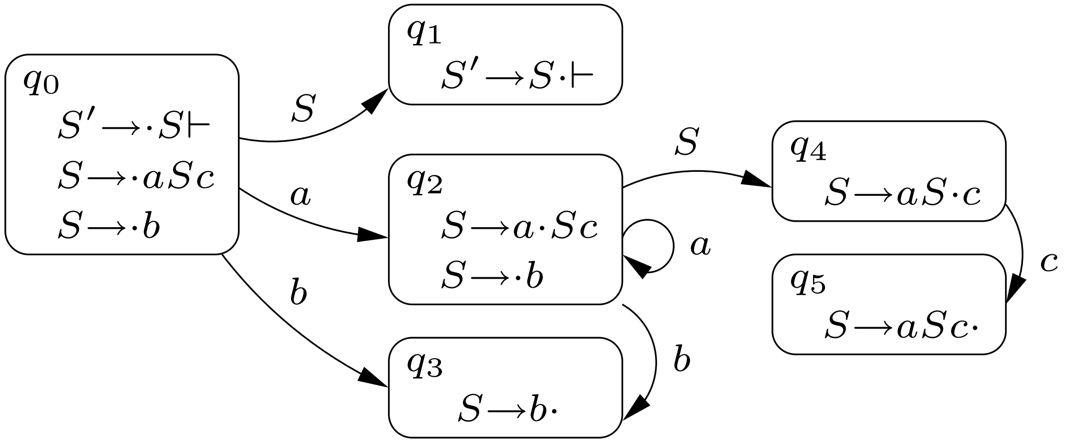
Langages reconnus à partir de q_9 (état en conflit) :

à droite des décalages $(l+s)^+ \; (s \; l^+ = (l+s)^*)^* \vdash$

à droite de la réduction $(s \; l^+ = (l+s)^*)^* \vdash$

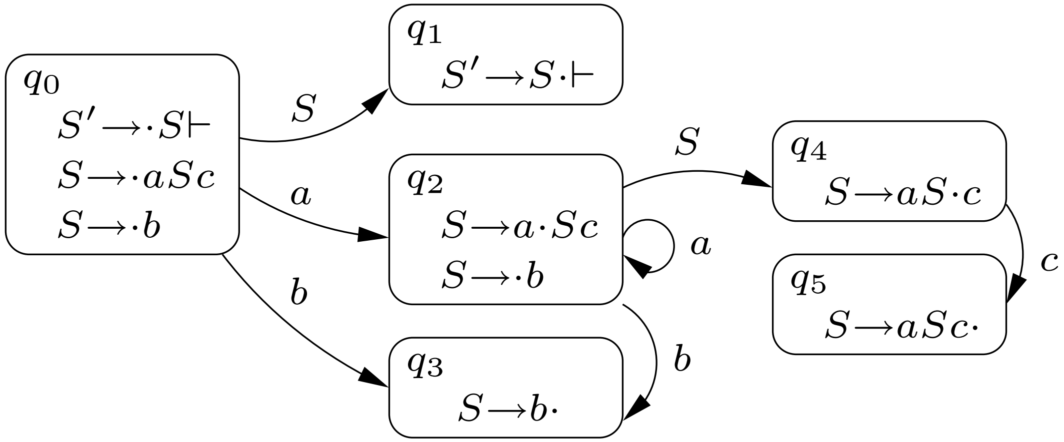
Calcul plus précis d'une couverture régulière - I

- P. Boullier, Thèse d'État Orléans, 1984, et M. E. Bermudez & K. M. Schimpf, Practical Arbitrary Look-ahead LR Parsing, J. of Computer and System Science **41**-2, 1990
- Dans l'automate LR(0) original
  . on suit les transitions sur les terminaux à partir de q_0
  . quand on atteint un état q avec $A \to \alpha \cdot$, on continue à partir de q' tel que $q'' \xrightarrow{\alpha} q, q'' \xrightarrow{A} q'$
  . exemple avec $a^n bc^n$: $q_0 \xrightarrow{a} q_2 \xrightarrow{a} q_2 \ldots q_2 \xrightarrow{b} q_3, (q_2 \xrightarrow{S} q_4) \xrightarrow{c} q_5,$ $(q_2 \xrightarrow{S} q_4) \xrightarrow{c} q_5, \ldots (q_2 \xrightarrow{S} q_4) \xrightarrow{c} q_5, (q_0 \xrightarrow{S} q_1)$ donne la couverture $a^+ bc^+$



Calcul plus précis d'une couverture régulière - II

– Mais pour les transitions sur a ou b, il y a un problème



– $q_0 \xrightarrow{a} q_2 \xrightarrow{a} q_2 \ldots \xrightarrow{b} q_3$

– Retour : $(q_2 \xrightarrow{S} q_4)$, ou $(q_0 \xrightarrow{S} q_1)$?

– Quelle transition sur b doit-on remonter en q_3 ? Idem pour les transition sur a en q_2 ? Si on ne sait pas, on accepte a^+b et bc^+

– Dépend de l'état par lequel on est venu en q_3 ou q_2

Calcul plus précis d'une couverture régulière - III

– On simule le fonctionnement d'un analyseur LR : on mémorise les états par lesquels on est passé dans une pile $q_{i_1} q_{i_2} \cdots q_{i_n}$

. suivre une transition sur un terminal $a \approx$ décaler : empiler l'état $q_{i_{n+1}}$ tel que, si la pile est $q_{i_1} \cdots q_{i_n}$, $q_{i_n} \xrightarrow{a} q_{i_{n+1}}$

. Quand on atteint un état avec $A \rightarrow \alpha \cdot$, on est donc aussi à droite de A

. quand on atteint un état avec $A \rightarrow \alpha \cdot \approx$ "réduire" selon $A \rightarrow \alpha$: dépiler $|\alpha|$ états, empiler q'_{i_n} tel que, si la pile est $q_{i_1} \cdots q_{i_n} \cdots q_{i_{n+|\alpha|}}$, $q_{i_n} \xrightarrow{A} q'_{i_n}$

. Quand on "réduit" à S', on a calculé la couverture régulière

Exemple

pile	action	langage
q_0	suivre a	a
q_0q_2	suivre a	a^2
$q_0q_2q_2$	suivre b	a^2b
$q_0q_2q_2q_3$	depiler(1)	a^2b
$q_0q_2q_2$	suivre S	a^2b
$q_0q_2q_2q_4$	suivre c	a^2bc
$q_0q_2q_2q_4q_5$	depiler(3)	a^2bc
q_0q_2	suivre S	a^2bc
$q_0q_2q_4$	suivre c	a^2bc^2
$q_0q_2q_4q_5$	depiler(3)	a^2bc^2
q_0	suivre S	a^2bc^2
q_0q_1	reduire à S'	a^2bc^2

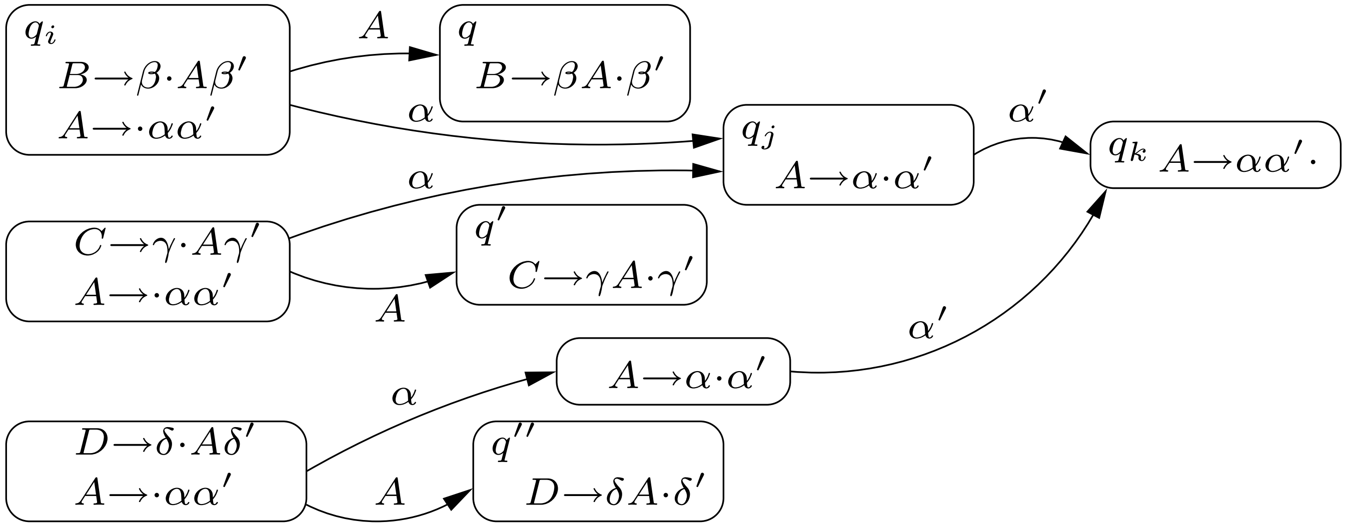
Calcul plus précis d'une couverture régulière - IV

- Grosse différence avec un analyseur : on n'a pas de texte d'entrée
  $\Rightarrow$ envisager toutes les possibilités
- À un état r de l'automate fini de la couverture régulière est associé un ensemble de piles d'états de l'automate LR(0)
- $r_0 : \{q_0\}$
- r_1 tel que $r_0 \xrightarrow{a} r_1 : \{q_0q_2\}$
  r_2 tel que $r_0 \xrightarrow{b} r_2 : \{q_0q_3, q_0q_4\}$
- r_3 tel que $r_1 \xrightarrow{a} r_3 : \{q_0q_2q_2\}$
  r_4 tel que $r_1 \xrightarrow{b} r_4 : \{q_0q_2q_3, q_0q_2q_4\}$
- r_5 tel que $r_3 \xrightarrow{a} r_5 : \{q_0q_2q_2q_2\}$
  r_6 tel que $r_5 \xrightarrow{a} r_6 : \{q_0q_2q_2q_2q_2\}$
- Quand arrête-t-on ?

Calcul plus précis d'une couverture régulière - V

- Chaque état r_i de l'automate de reconnaissance de la couverture régulière comprend un ensemble de piles d'états LR(0)
- Pour arrêter le calcul d'une couverture régulière, on borne la hauteur des piles à une constante h
  - . empiler dans une pile de hauteur h supprime l'élément de fond de pile : noté $\pi : h$
- Le nombre de piles différentes est donc fini (puisque le nombre d'états de l'automate LR(0) est fini)
  - . donc le nombre d'ensemble différents de piles est fini
  - . donc le nombre d'états r de l'automate de la couverture régulière est fini : ceci introduira des boucles dans cet automate, c'est à dire des $*$
- Les piles contrôlent (partiellement) les remontées lors des "réductions"

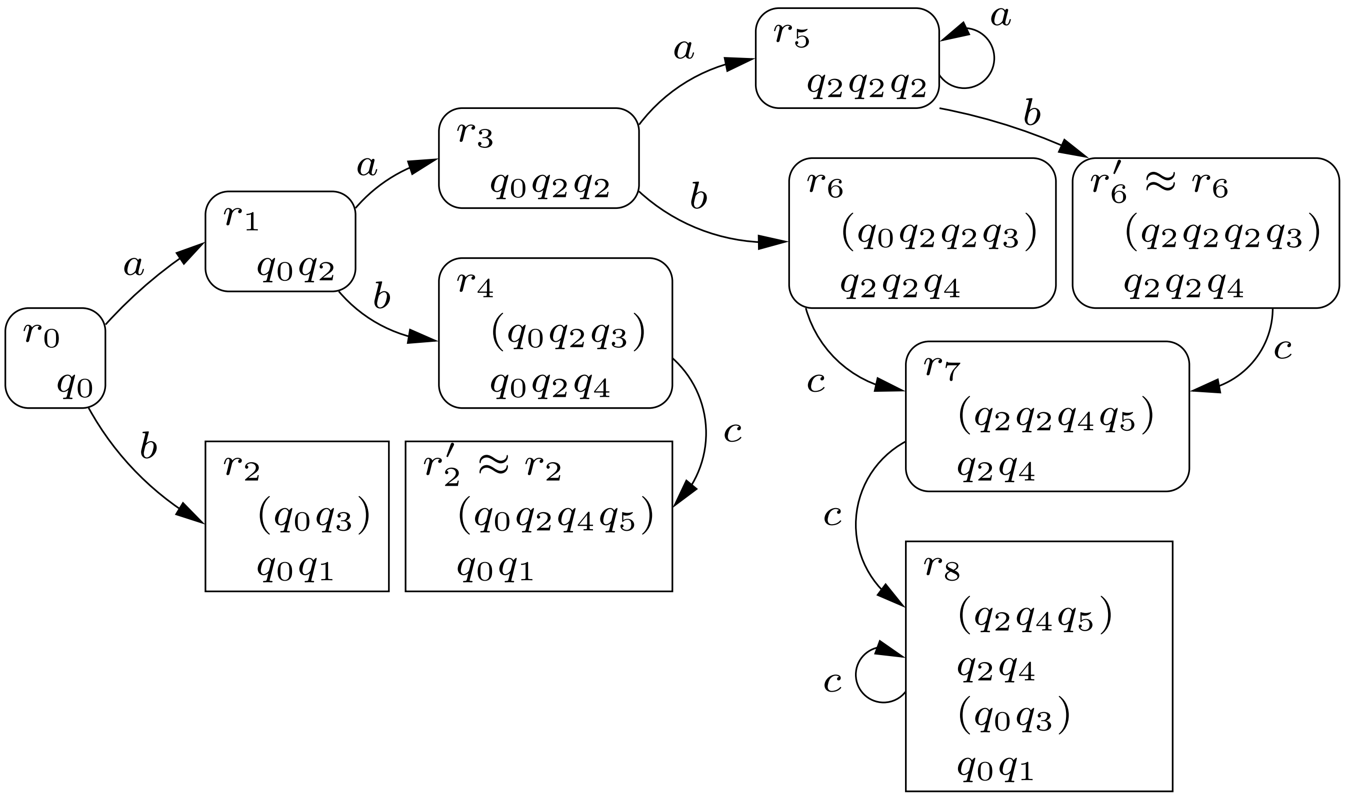
Exemple de remontée contrôlée (partiellement) par la pile



- En suivant $\alpha\alpha'$, $q_i \cdots q_j$ donnera, par exemple, $q_j \cdots q_k$ si $|\alpha'| = h$, qui donnera les piles q et q', mais pas q''.

- On perd donc de la précision dans le calcul de la couverture du contexte droit de l'état q_i (qui ne contient pas $\mathcal{L}(\gamma')$), , mais on a quand même évité d'inclure $\mathcal{L}(\delta')$

Exemple avec $h = 3$ pour $a^n b c^n$



– Couverture régulière $= b + abc + a^2 a^* b c^* c^2$

Construction formelle - Fonctions utilitaires

– États prédécesseurs ($q' \in Pred(q, \alpha)$ sera noté $q \xrightarrow{\alpha} q'$) :

$$Pred(q, \alpha) = \begin{cases} \{q\} & \text{si } \alpha = \varepsilon \\ \displaystyle\bigcup_{q' \in Pred(\alpha')} \{q'' \mid q'' \xrightarrow{X} q'\} & \text{si } \alpha = X\alpha' \end{cases}$$

– Retour en arrière :

$$Ante(\pi, \alpha) = \begin{cases} \{\pi_1 \mid \pi = \pi_1\pi_2, |\pi_2| = |\alpha|\} & \text{si } |\pi| > |\alpha| \\ \{q' \mid q' \in Pred(1 : \pi, \alpha_1), & \\ \quad \alpha = \alpha_1\alpha_2, |\alpha_2| = |\pi| - 1 & \text{si } |\pi| \leq |\alpha| \end{cases}$$

On remonte les transitions de l'automate en se guidant autant que possible avec les états contenus dans la pile ($\equiv$ *lookback* de LALR, mais restreint)

Construction formelle - Fonctions de déplacement dans l'automate

– Saut des non-terminaux se dérivant dans le vide

$$
\begin{aligned}
Next(\pi) = \{\pi : h\} \\
\cup \{\pi'' \mid \pi'' \in Next(\pi'), \pi = \pi_1 q, \pi' = \pi q' : h, q \xrightarrow{A} q', \\
B \rightarrow \beta \cdot A\gamma \in q, A \Rightarrow^+ \varepsilon\}
\end{aligned}
$$

$\equiv$ *reads* de LALR

– "Réductions"

$$
\begin{aligned}
Prev(B \rightarrow \alpha \cdot \beta, \pi) = \\
\{\pi'' \mid \pi'' \in Next(\pi'), \pi = \pi_1 q, \pi' = \pi q', q \xrightarrow{A} q', \beta = A\beta_1\} \\
\cup \{\ \pi'' \mid \pi'' \in Prev(C \rightarrow \gamma \cdot B\varphi, \pi'), \pi' \in Ante(\pi, \alpha), \\
C \rightarrow \gamma \cdot B\varphi \in q', \pi' = \pi'_1 q', (\beta = \varepsilon \text{ ou } \beta = A\beta_1 \text{ et } \beta_1 \Rightarrow^* \varepsilon)\}
\end{aligned}
$$

$\equiv$ *lookback* $\circ$ *includes* de LALR, mais restreints par la pile

Application à la construction LRR - I

- À chaque état r_i^q de l'automate de reconnaissance du contexte droit du conflit en q, on associe un ensemble de couples (π, i) où i est une action en conflit et π une pile d'états LR(0)
- État initial : on part d'un état en conflit q

$$r_0^q = \{(q,0) \,|\, \exists A{\rightarrow}\alpha{\cdot}a\beta \in q\}$$
$$\cup\ \{(\pi, i) \,|\, A\overset{i}{\longrightarrow}\alpha \in P, A{\rightarrow}\alpha{\cdot} \in q, \pi \in Prev(A{\rightarrow}\alpha{\cdot}, q), A \neq S'\}$$

- Transitions sur un terminal a

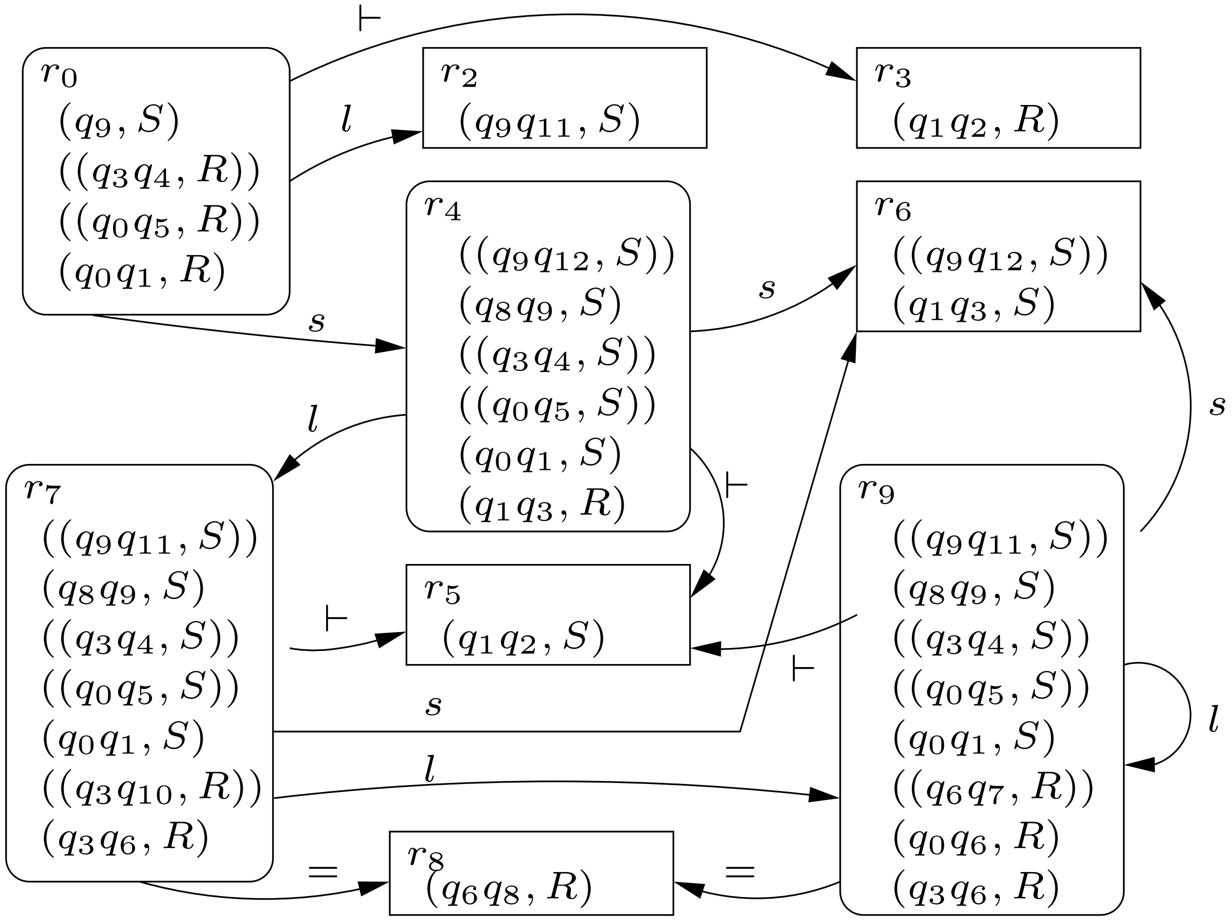
$$\Theta(r,a) = \{(\pi', i) \,|\, (\pi, i) \in r, \pi = \pi_1 q, A{\rightarrow}\alpha{\cdot}a\beta \in q,$$
$$\pi' \in Next(\pi''), \pi'' = \pi q', q\overset{a}{\rightarrow}q'$$
$$\cup\ \{(\pi', i) \,|\, (\pi, i) \in r, \pi = \pi_1 q, A{\rightarrow}\alpha{\cdot}a\beta \in q, \beta{\Rightarrow}^*\varepsilon, A \neq S',$$
$$\pi'' \in Ante(\pi, \alpha), \pi'' = \pi_1'' q'', C{\rightarrow}\gamma{\cdot}A\varphi \in q'',$$
$$\pi' \in Prev(C{\rightarrow}\gamma{\cdot}A\varphi, \pi''\}$$

Application à la construction LRR - II

- Il suffit de discriminer sur des **préfixes** des couvertures régulières des langages à droite d'un conflit
  $\Rightarrow$ On peut résoudre le conflit en faveur de l'action i si, dans un état r_k^q, tous les couples (π, i) ont le même i
- On sait qu'on ne pourra pas résoudre le conflit si les suffixes des langages droits restant à examiner ne sont pas disjoints
  $\Rightarrow$ On rejette la grammaire si, dans un état r_k^q, il existe (π, i) et (π, i') avec $i \neq i'$

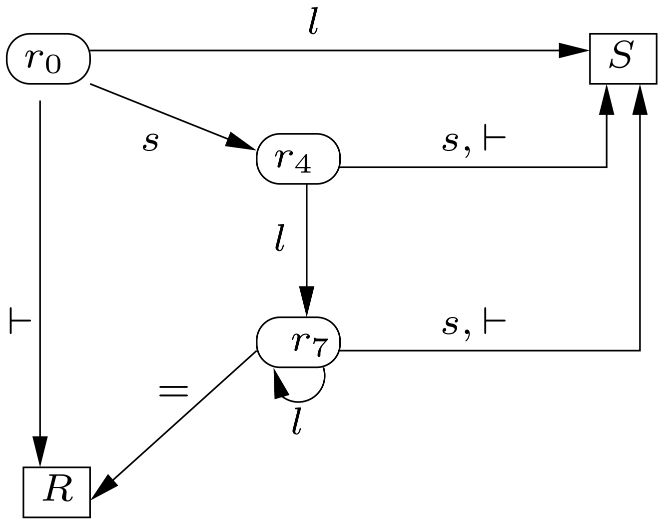
- Avec une pile de longueur bornée à k, si on n'a pas sauté de non-terminaux se dérivant dans le vide, on a exploré au moins les k premiers terminaux du contexte droit exact du conflit
  $\Rightarrow$ **garantit l'acceptation de toute grammaire LALR(k) sans ε-productions**

Application à la construction LRR (avec $h = 2$) pour $\mathcal{G}_1$ - I



Application à la construction LRR (avec $h = 2$) pour $\mathcal{G}_1$ - II

– Soit sous forme réduite (les états carrés permettent de décider l'action en conflit –S pour le *shift*, R pour le *reduce*)



– Préfixes décidant d'un décalage : $l + sl^*(s + \vdash)$

– Préfixes décidant de la réduction : $\vdash + sll^* =$

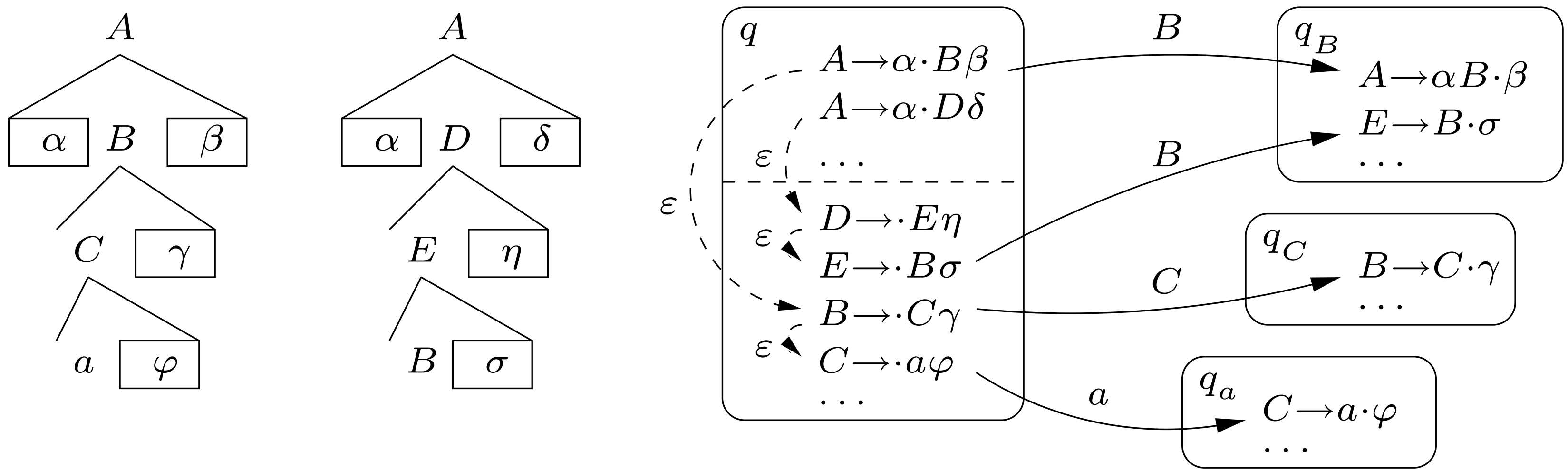
Construction LRR avec graphes d'items

– J. Farré & J. Fortes Gálvez, A Bounded Graph-Connect Construction for LR-regular Parsers, Compiler Construction, LNCS 2027, Springer Verlag 2001

– Plus puissante que la construction avec pile d'états

 . Notamment pour les grammaires avec ε-productions

– L'automate LR(0) définit un graphe d'items du noyau des états LR(0)



Définition du graphe d'items

– Un item $A \to \alpha X \cdot \beta$ du **noyau** d'un état q est le nœud $\nu = [A \to \alpha X \cdot \beta]_q$ du graphe

– Les transitions sur les nœuds sont définies par :

$$\begin{aligned}
\delta(\nu, X) \quad = \quad & \{[A \to \alpha X \cdot \beta]_{q'} \mid q \xrightarrow{X} q', \nu = [A \to \alpha \cdot X \beta]_q\} \\
& \cup \{[C \to X \cdot \gamma]_{q'} \mid q \xrightarrow{X} q', \nu = [A \to \alpha \cdot B \beta]_q, \\
& \qquad B \underset{\mathrm{rm}}{\Rightarrow}^* C x \Rightarrow X \gamma x\}.
\end{aligned}$$

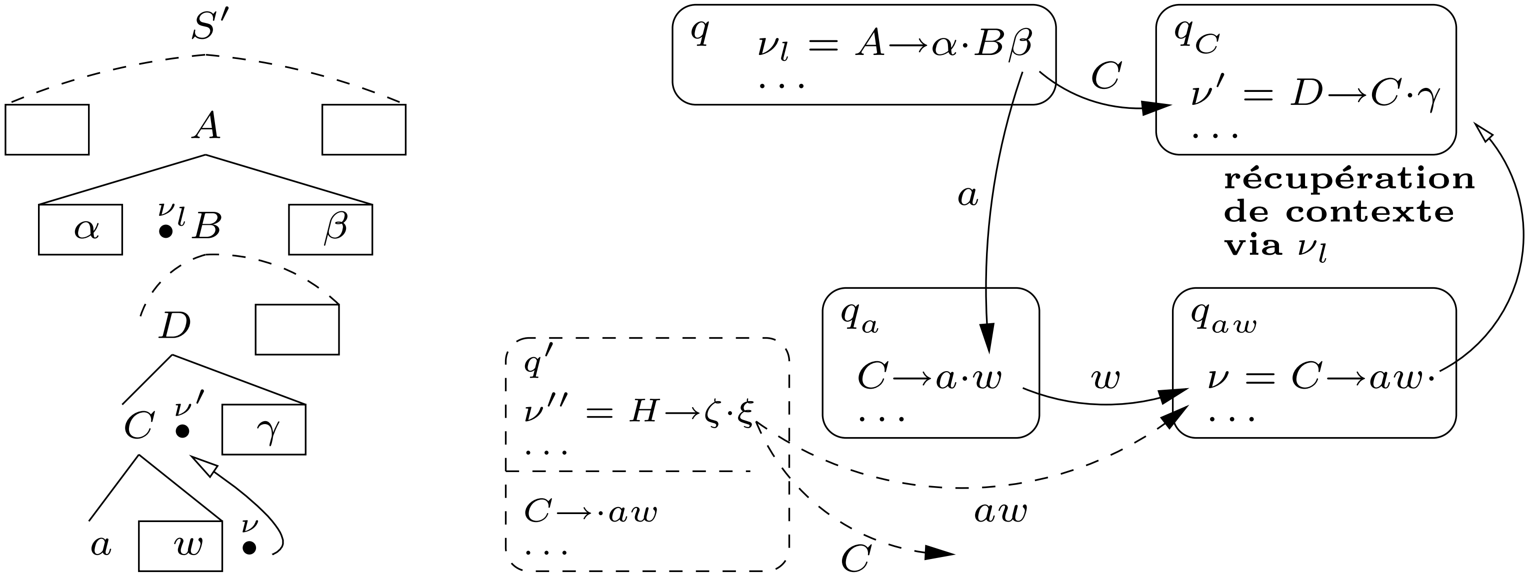
et

$$\delta^*(\nu, \varepsilon) = \{\nu\}, \qquad \delta^*(\nu, X\alpha) = \bigcup_{\nu' \in \delta(\nu, X)} \delta^*(\nu', \alpha).$$

On écrira $\nu \xrightarrow{\alpha} \nu'$ pour $\nu' \in \delta^*(\nu, \alpha)$

Récupération de contexte

– Un couple (ν_l, ν) permet de remonter les chemins du graphe



– Arrivé sur $\nu = [A\to X\alpha\cdot]_q$, on continue avec $\nu' = [B\to A\cdot\beta]_{q'}$ tel que $\nu_l \xrightarrow{B} \nu'$, sauf si $\nu_l = [A\to X\cdot\alpha]_{q_l} \xrightarrow{\alpha} \nu$, en quel cas, on continue avec tous les ν' autorisés par le graphe ($\equiv$ *lookback*, mais restreint par ν_l)

Connexion de sous-graphes

– Suivre une transition qui correspond à une descente dans un sous-arbre demande de faire une connexion du sous-graphe correspondant au sous-arbre

– Au lieu d'une pile d'états, on utilise une pile de couples

$$\kappa = (\nu_c^{(1)}, \nu_t^{(1)}) \cdots (\nu_c^{(m)}, \nu_t^{(m)}), m \geq 1.$$

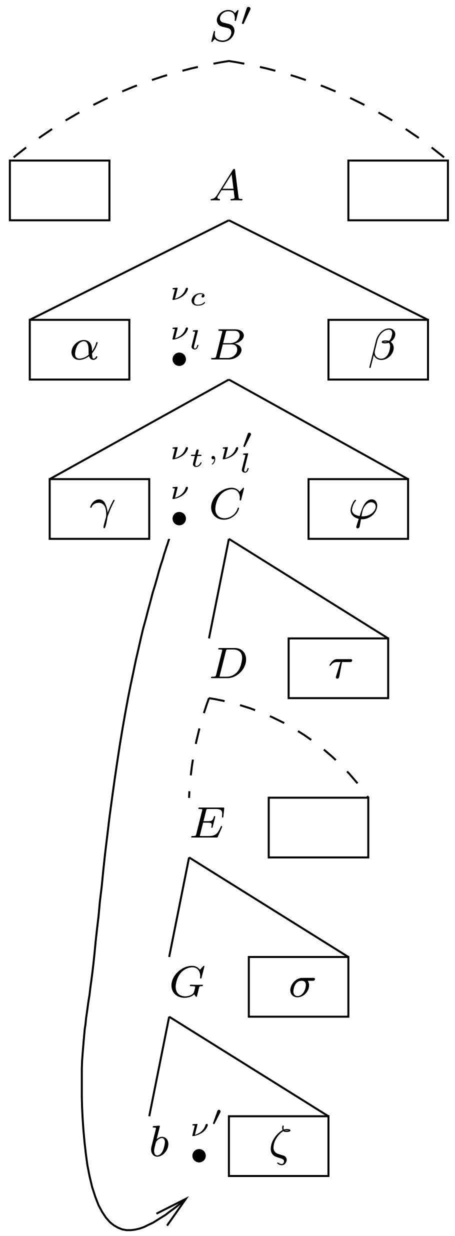
En général, $\nu_t^{(i)} \xrightarrow{\eta} \nu_c^{(i+1)}, \eta \Rightarrow^* \varepsilon$

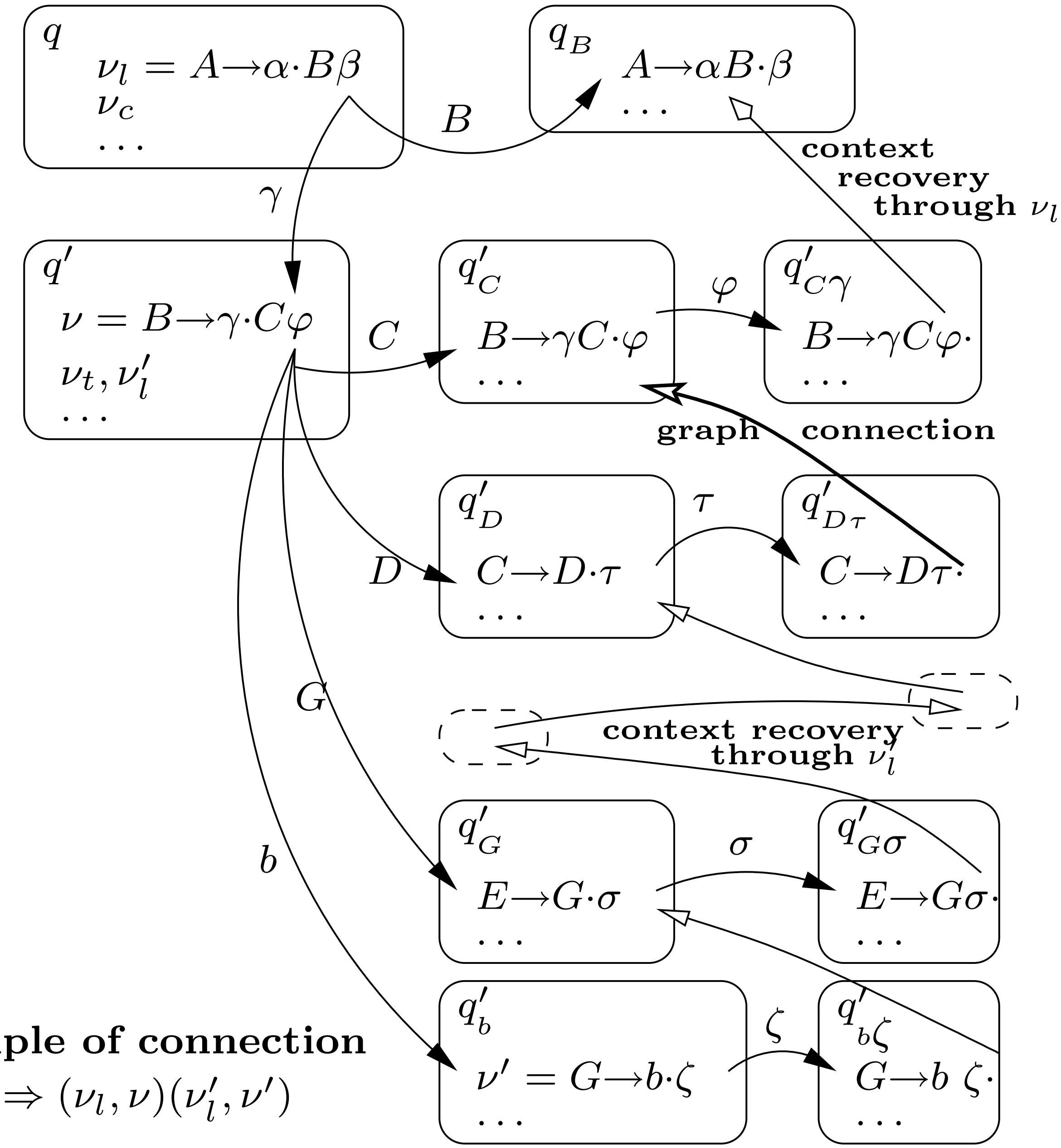
– $(\nu_c^{(m)}, \nu_t^{(m)})$ sera aussi noté (ν_l, ν)

– Faire une connexion revient à ajouter un couple (ν_l', ν') à la pile

– La hauteur de la pile est bornée à h couples

– Exemple :





Example of connection

$(\nu_l, \nu) \Rightarrow (\nu_l, \nu)(\nu'_l, \nu')$

Saut des non-terminaux se dérivant dans le vide

– On doit toujours être positionné sur un nœud de la forme $[A \rightarrow X\alpha{\cdot}a\beta]_q$, ce qui implique

. de "réduire" quand on atteint nœud de la forme

$$[B \rightarrow X\beta{\cdot}\gamma]_q, \gamma \Rightarrow^* \varepsilon,$$

ce qui demande de remonter les chemins dans les sous-graphes, et éventuellement les connexions

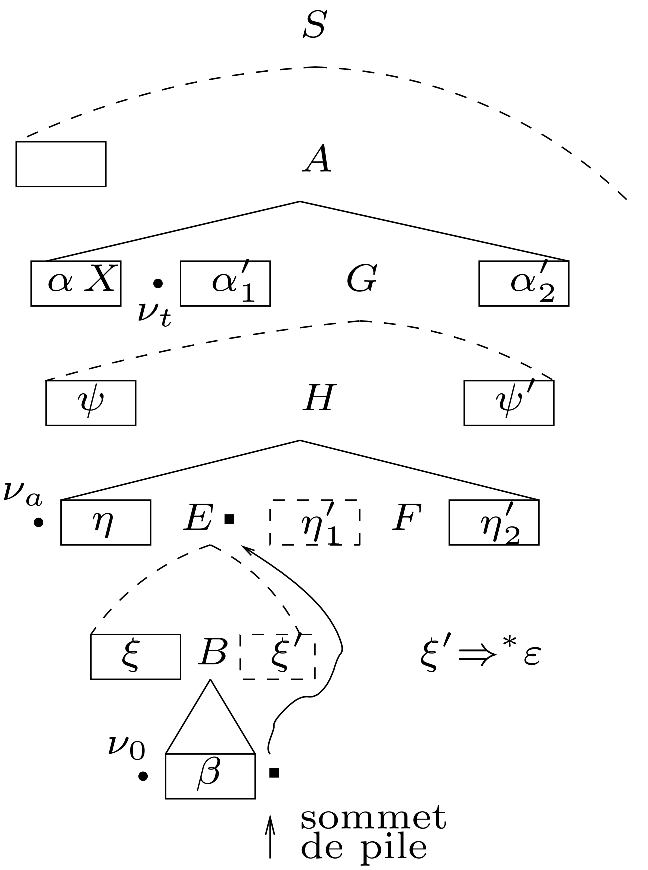
. de suivre les transitions sur γ quand on atteint nœud de la forme

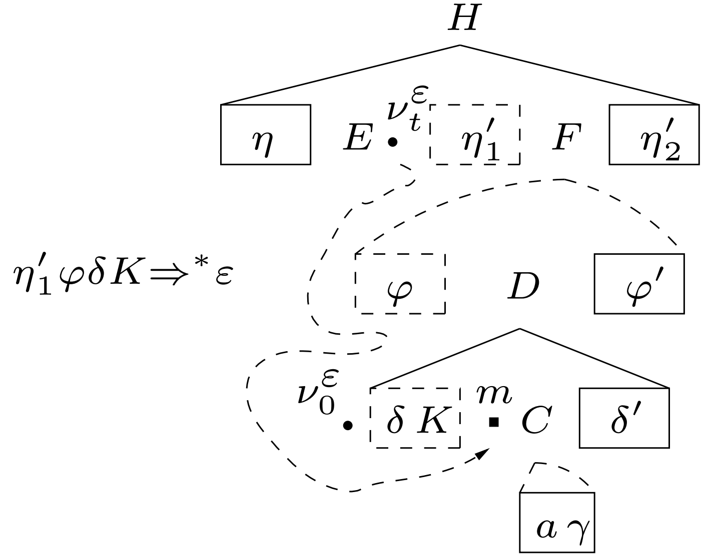
$$[B \rightarrow X\beta{\cdot}\gamma Y\varphi]_q, \gamma \Rightarrow^* \varepsilon, Y \Rightarrow^* by,$$

ce qui peut correspondre éventuellement à une descente dans le sous-arbre de Y si $Y \rightarrow \eta EZ\sigma, \eta E \Rightarrow^* \varepsilon, Z \Rightarrow^* bz$

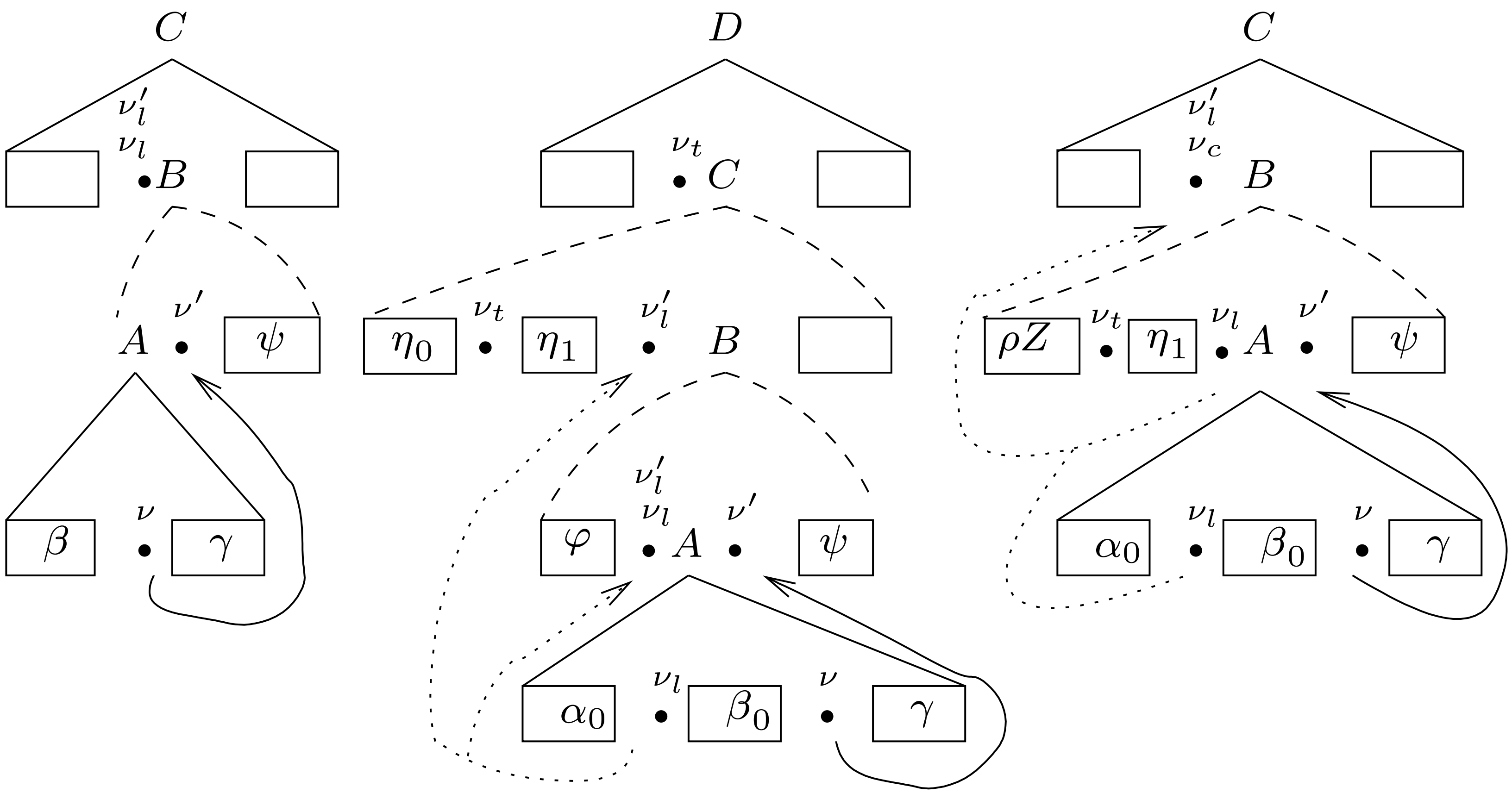
. $\equiv$ *includes* $\circ$ *reads*, mais contraint par le pile de couples de nœuds

Exemple : de sauts de non-terminaux se dérivant dans le vide





Récupération du contexte – exemple



ascent in same subtree ascent in same graph ascent to connected graph

Récupération du contexte – formules

– Récupération du contexte

$$\hat{\theta}(\kappa(\nu_l,\nu)) =$$

$$\begin{cases} \{\kappa_1(\nu_c,\nu') \mid \nu_t \overset{\eta}{\longleftarrow} \nu_l', \nu' = [B\to\rho Z\eta A\cdot\psi]_q, \\ \qquad \eta \Rightarrow^* \varepsilon, (\nu_l,\nu) \upuparrows (\nu_l',\nu')\} \\ \cup\ \{\kappa(\nu_l',\nu') \mid \nu_t \overset{\eta\varphi}{\longleftarrow} \nu_l', \nu' = [B\to\varphi A\cdot\psi]_q, \\ \qquad \eta\varphi \Rightarrow^* \varepsilon, (\nu_l,\nu) \upuparrows (\nu_l',\nu')\} & \text{si } \kappa = \kappa_1(\nu_c,\nu_t) \\ \\ \{(\nu_l',\nu') \mid (\nu_l,\nu) \upuparrows (\nu_l',\nu')\} & \text{autrement} \end{cases}$$

où $(\nu_l,\nu)\upuparrows(\nu_l',\nu')$ ssi $\nu_l' \overset{\alpha}{\longleftarrow} \nu_l \overset{\beta}{\longleftarrow} \nu = [A\to\alpha\beta\cdot\gamma]_q$, $\nu_l' \overset{A}{\longleftarrow} \nu'$, et $\gamma \Rightarrow^* \varepsilon$

– Saut des non-terminaux se dérivant dans le vide

$$\hat{\Theta}(r) = \{[j,\kappa'(\nu_l,\nu)] \mid \kappa'(\nu_l,\nu) \in \hat{\theta}(\kappa), [j,\kappa] \in r, \nu = [A\to\alpha\cdot\beta]_q, \beta \Rightarrow^* ax\}$$

Construction de l'automate discriminant

- On associe à chaque état des ensembles d'*items de fenêtre* $\mu = [i, \kappa]$, où i est une action en conflit, et κ une pile de couples (ν_l, ν)
- Fonction de transition :

$$
\begin{aligned}
&\Theta_h(r,a) = \\
&\qquad \hat{\Theta}(\{[j, \kappa(\nu_l, \nu')] \mid [j, \kappa(\nu_l, \nu)] \in r, \\
&\qquad\qquad \nu \xleftarrow{\beta Y} \nu' = [A \rightarrow \alpha X \beta Y \cdot \gamma]_q, \beta \Rightarrow^* \varepsilon, Y \Rightarrow^* a\} \\
&\qquad\ \cup \{[j, \kappa(\nu_c, \nu_t)(\nu_l, \nu) : h] \mid [j, \kappa(\nu_c, \nu_t)] \in r, \\
&\qquad\qquad\quad \nu_t \xleftarrow{\eta} \nu_l \xleftarrow{\beta Y} \nu = [A \rightarrow \beta Y \cdot \gamma]_q, \eta\beta \Rightarrow^* \varepsilon, Y \Rightarrow^* a, \gamma \Rightarrow^* bx\}).
\end{aligned}
$$

- État initial pour le conflit dans l'état q :

$$
\begin{aligned}
r_0^q = \{[0, (\nu, \nu)] \mid K_q \ni \nu \xleftarrow{a} \nu'\} \\
\cup\ \hat{\Theta}(\{[i, (\nu_l, \nu')] \mid A \xrightarrow{i} \alpha, \nu_l \xleftarrow{A} \nu', \nu_l \xleftarrow{\alpha} \nu \in K_q\}
\end{aligned}
$$

Grammaires rejetées

– Une grammaire est rejetée ssi $\exists\,[j,\kappa],[j',\kappa] \in r_i^q,\ j \neq j'$

Algorithme de construction de l'automate discriminant

DFA(h) GENERATOR:
  for each conflict-state q **do**
    initialize *states* with r_0^q
    repeat
      let r be next DFA state from *states*
      $Ps := \{(j,a) \mid [j, \kappa(\nu_l, [A{\rightarrow}\alpha{\cdot}\beta]_q)] \in J_r, a \in \mathrm{first}(\beta)\}$
      for $a \in T \mid \exists(j,a) \in Ps$ **do**
        if $\nexists(j',a) \in Ps \mid j' \neq j$ **then** $action(r,a) := j$
        else $r' := \Theta_h(r,a)$; add r' to *states*; $action(r,a) :=$ goto r'
    until end of *states* or inadequacy (rejection)

Limitations de cette construction

– Pour une pile de hauteur maximum k, accepte toutes les grammaires LALR(k), y compris celles avec ε-productions

– Mais pas de garantie pour les grammaires LR non LALR, à cause de la confusion des contextes

– Exemple de grammaire LR(1) non LALR(1) qui est rejetée :

$$S \to aAa \quad S \to bAb \quad S \to aBb \quad S \to bBa$$

$$A \to aAa \quad A \to bAb \quad A \to c$$

$$B \to aBb \quad B \to bBa \quad B \to c$$

– Il est encore possible, dans certains cas, de résoudre le conflit en explorant la pile par une sorte d'automate discriminant rétrograde, et donc d'accepter une partie des grammaires LR (B. Séité, A yacc extension for LRR parsing, Theoretical Computer Science **52**, 1987)

Pourquoi la grammaire LR(1) est rejetée

– Avec un automate LR(0), on arrive par a et b dans un même état, qui contient un conflit sur les réductions $A \rightarrow c$ et $B \rightarrow c$

– On confond donc les contextes gauches, et les contextes droits sont $(a+b)^+$ pour les deux réductions en conflit

– Alors qu'avec un automate LR(1),

  . si le contexte gauche de c est $(a+b)^*a$,

    $*$ si son contexte droit est $a(a+b)^*$, alors $A \rightarrow c$,

    $*$ et si son contexte droit est $b(a+b)^*$, alors $B \rightarrow c$,

  . et si le contexte gauche de c est $(a+b)^*b$,

    $*$ si son contexte droit est $a(a+b)^*$, alors $B \rightarrow c$,

    $*$ et si son contexte droit est $b(a+b)^*$, alors $A \rightarrow c$