Analyse LR généralisée (GLR)

Analyse non déterministe des grammaires non contextuelles

- Il existe de nombreuses grammaires non contextuelles pour lesquelles on ne sait pas construire un analyseur déterministe, les grammaires ambiguës notamment
- Les grammaires des langues naturelles, en particulier, sont ambiguës (en réalité, elles sont même non contextuelles, mais on peut s'arranger pour avoir un "squelette" non contextuel de la langue)
- Quand, dans un état donné, l'automate a le choix entre 2 (ou plus) actions, il existe deux possibilités
  1. Effectuer une des actions et continuer l'analyse : si l'automate tombe en erreur, revenir en arrière et essayer avec une autre action
  2. Effectuer toutes les actions en parallèle
- La deuxième solution permet d'explorer toutes les possibilités données par la grammaire

Une mini grammaire de langue naturelle

– Soit la grammaire G_{ln} modélisant un sous-ensemble de l'anglais

1:	$S \rightarrow NP\ VP$	6:	$NP \rightarrow NP\ PP$
2:	$S \rightarrow S\ PP$	7:	$NP \rightarrow NP$ and NP
3:	$S \rightarrow S$ and S	8:	$VP \rightarrow$ v NP
4:	$NP \rightarrow$ n	9:	$VP \rightarrow$ v S
5:	$NP \rightarrow$ d n	10:	$PP \rightarrow$ with NP

où n représente un nom ou un pronom, d un article, v un verbe

– Exemple de phrase

I saw a man : $S \underset{\text{rm}}{\Rightarrow} NP\ VP \underset{\text{rm}}{\Rightarrow} NP\ v(\text{saw})\ NP \underset{\text{rm}}{\Rightarrow}$
$NP\ v(\text{saw})\ d(\text{a})\ n(\text{man}) \underset{\text{rm}}{\Rightarrow} n(\text{I})\ v(\text{saw})\ d(\text{a})\ n(\text{man})$

Phrases ambiguës

– La grammaire G_{ln} est ambiguë

– I saw a man with a telescope : soit

. [I saw a man] [with a telescope] : $S \underset{\text{rm}}{\Rightarrow} S\ PP$

. [I saw] [a man with a telescope] : $S \underset{\text{rm}}{\Rightarrow} NP\ VP \underset{\text{rm}}{\Rightarrow} NP \text{ v } NP \underset{\text{rm}}{\Rightarrow} NP \text{ v } NP\ PP$

– I know Jane and Jack knew it :

. [I know] [Jane and Jack knew it] : $S \underset{\text{rm}}{\Rightarrow} NP\ VP \underset{\text{rm}}{\Rightarrow} NP \text{ v } S \underset{\text{rm}}{\Rightarrow} NP \text{ v } NP\ VP \Rightarrow^* NP \text{ v } NP \text{ and } NP\ VP$

. [I know Jane] and [Jack knew it] : $S \underset{\text{rm}}{\Rightarrow} S \text{ and } S$

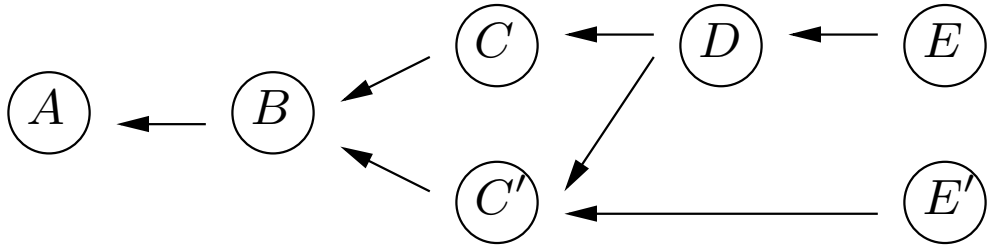
– On peut donc avoir plusieurs analyses d'une même phrase

Analyse non déterministe reposant sur un automate déterministe

- On construit un automate (LALR(1), ...) dans lequel un état peut avoir plusieurs actions en cas de conflit (non déterminisme)
- Quand l'automate atteint un état non déterministe, il se duplique (avec sa pile) : chaque copie fait une des actions sur sa pile
- On peut donc avoir, en général, plusieurs automates qui s'exécutent en parallèle, et qui peuvent éventuellement se dupliquer à leur tour
- Ces automates sont synchronisés sur les décalages
- Un automate est tué s'il détecte une erreur, car, si on suppose l'entrée correcte, son analyse est incompatible avec l'entrée
- Les automates allant jusqu'au bout donnent les analyses possibles de l'entrée

Pile implémentée par un graphe

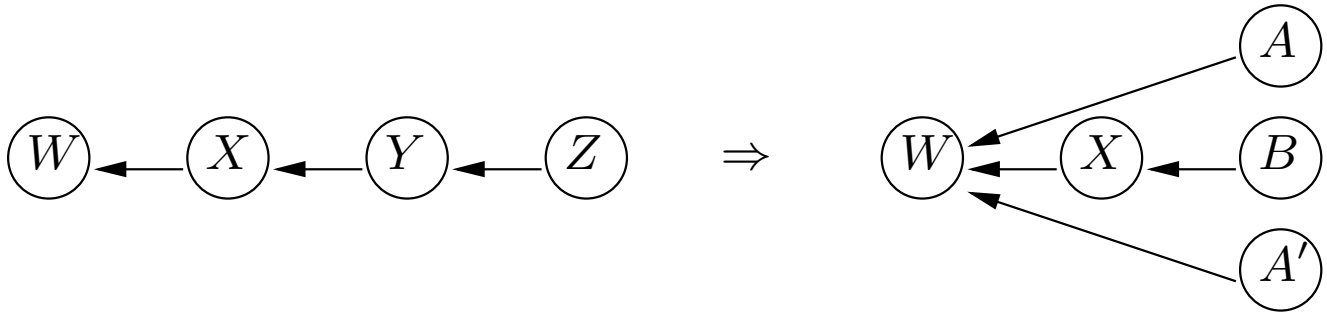
- Les piles des automates parallèles ont de nombreuses parties communes
- Ces piles seront représentées par un graphe (sans cycles) (*graph-structured stack* – GSS)
- Exemple pour les piles $ABCDE$, $ABC'DE$ er $ABC'E'$



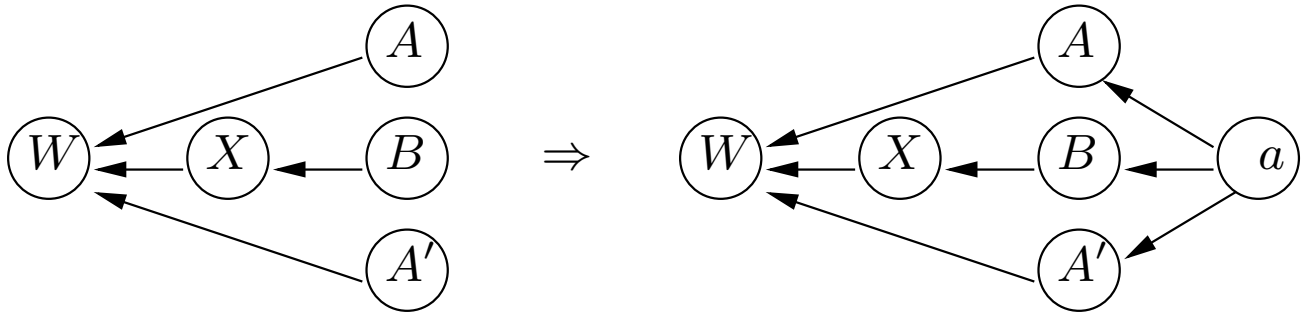
- Les sommets de pile sont les nœud sans prédécesseurs

Principales opérations sur le GSS - I

– Éclatement : quand un automate a le choix entre plusieurs actions par exemple, réduction de XYZ à A ou A', réduction de YZ à B

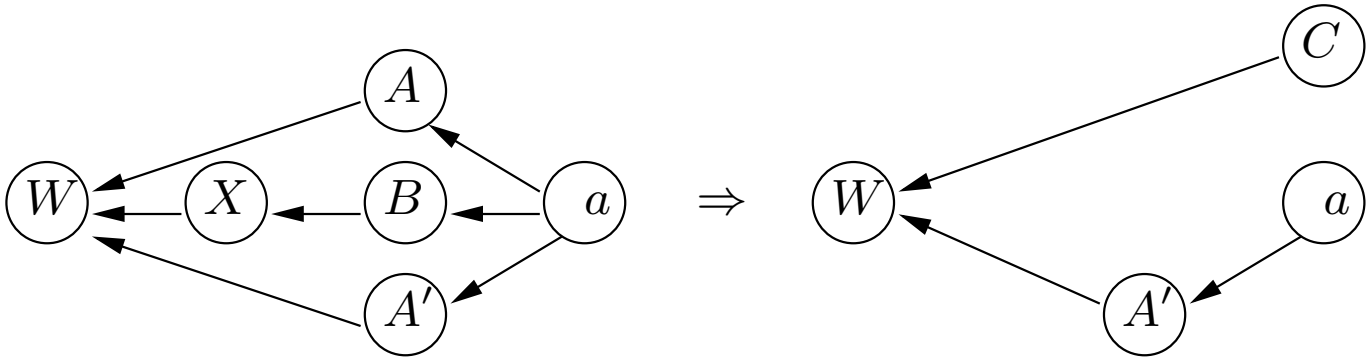


– Partage : quand on décale le prochain terminal

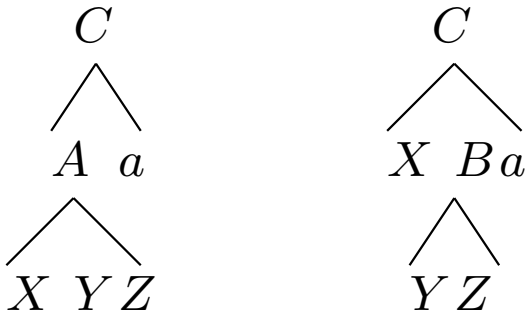


Principales opérations sur le GSS - II

– Compactage dû à une ambiguïté locale : si Aa et XBa se réduisent à C

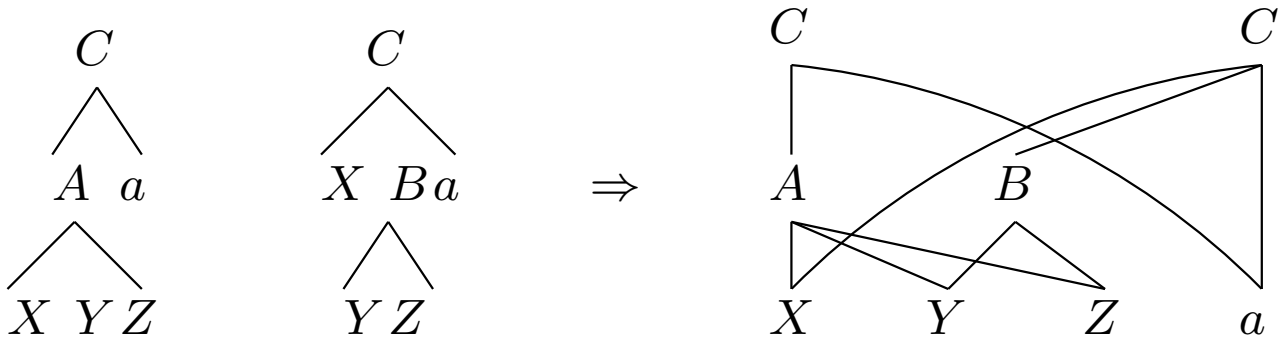


On a en ce cas les deux sous-arbres possibles de racine C



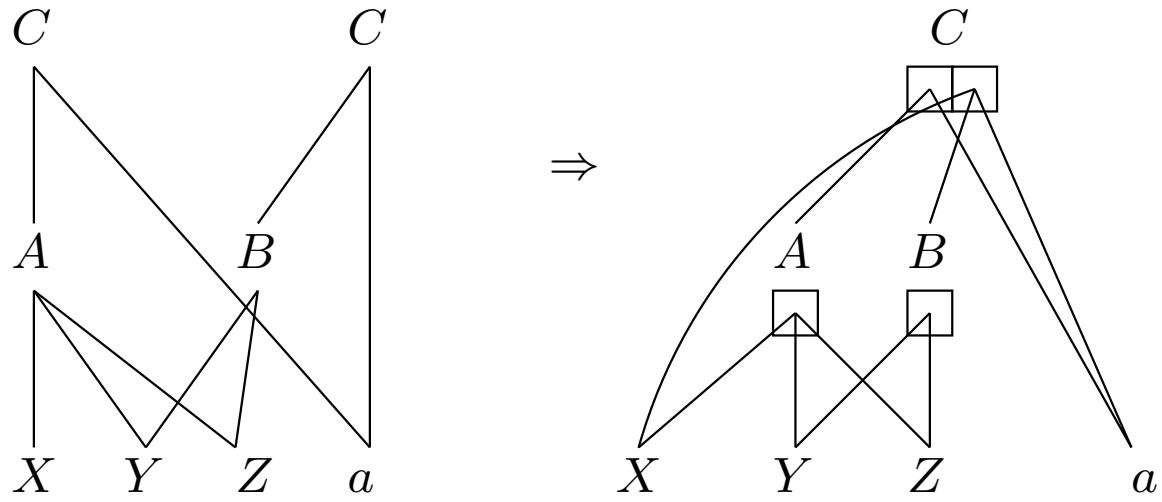
Forêt partagée

- Si la grammaire est ambiguë, les automates survivants ont chacun calculé un arbre de dérivation
- L'ensemble de ces arbres est une forêt
- Plusieurs arbres de la forêt ont des sous-arbres communs, que l'on partage (forêt partagée)



Forêt partagée compactée

– Des sous-arbres couvrant la même portion de la phrase d'entrée et dont la racine est un même symbole (à cause d'une ambiguïté locale) peuvent être regroupés sous une même racine (forêt partagée compactée)



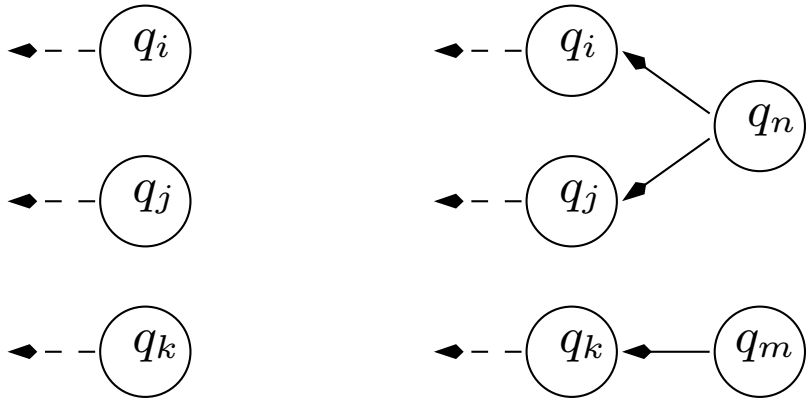
– Un symbole et les sous-arbres dont il est la racine sera appelé un *vertex*

Analyse LR généralisée

- Masaru Tomita, Efficient Parsing for Natural Language, Kluwer Academic Publishers, 1986
- La méthode repose sur un automate de la famille LR (SLR(1) ou LALR(1) principalement)
- Donc, les nœud du GSS doivent contenir des états LR, et pas des symboles de la grammaire
- Pour pouvoir construire la forêt partagée compactée, les nœuds du GSS doivent aussi contenir les vertex correspondant
- Ces caractéristiques des nœuds diminuent les possibilités de partage, car
  - . des états différents peuvent correspondrent à un même symbole
  - . on ne peut pas partager deux nœuds avec des vertex différents

Analyse GLR et partage des nœuds du GSS - I

- Les nœuds représentant les sommets de pile avant le décalage d'un terminal t doivent être partitionnés selon les états où mène t
- Il faut créer autant de nœuds ν_s correspondant à t qu'il existe de sous-ensembles π_s dans la partition, et établir un arc de ν_s vers tout nœud de π_s.
- Le GSS ci-dessous, et l'extrait de table (LA)LR(1) qui définit un partitionnement $\Pi = \{\{q_i, q_j\}, \{q_k\}, \cdots\})$, donnent



avant décalage — après décalage

	$\cdots$	t	$\cdots$
$\cdots$			
q_i		sh,q_n	
$\cdots$			
q_j		sh,q_n	
$\cdots$			
q_k		sh,q_m	
$\cdots$			

table (LA)LR(1)

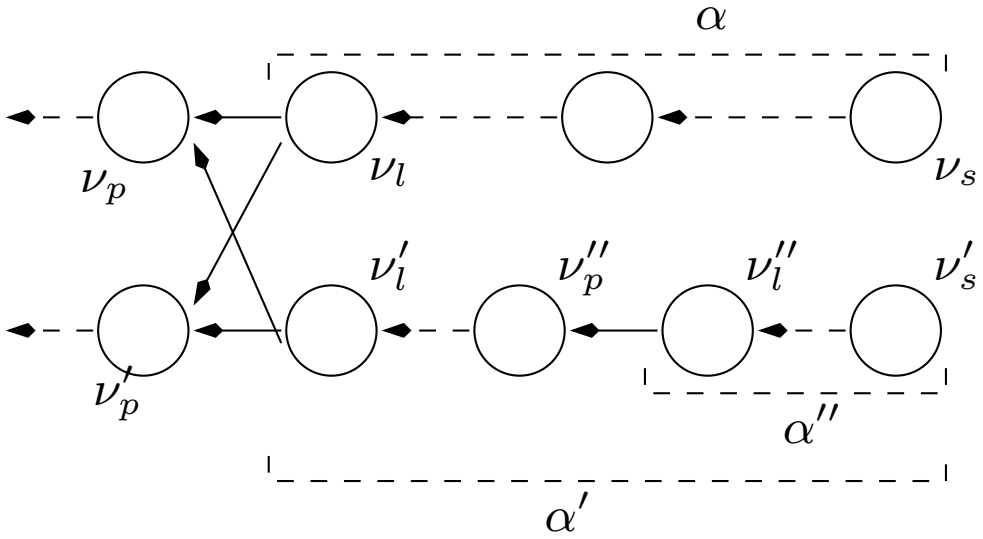
Analyse GLR et partage des nœuds du GSS - II

– Ce problème de partitionnement est analogue pour les réductions, avec la table *goto*

– De plus, pour les réductions, comme les nœuds du GSS sont aussi utilisés pour représenter la forêt partagée compactée, seuls ceux

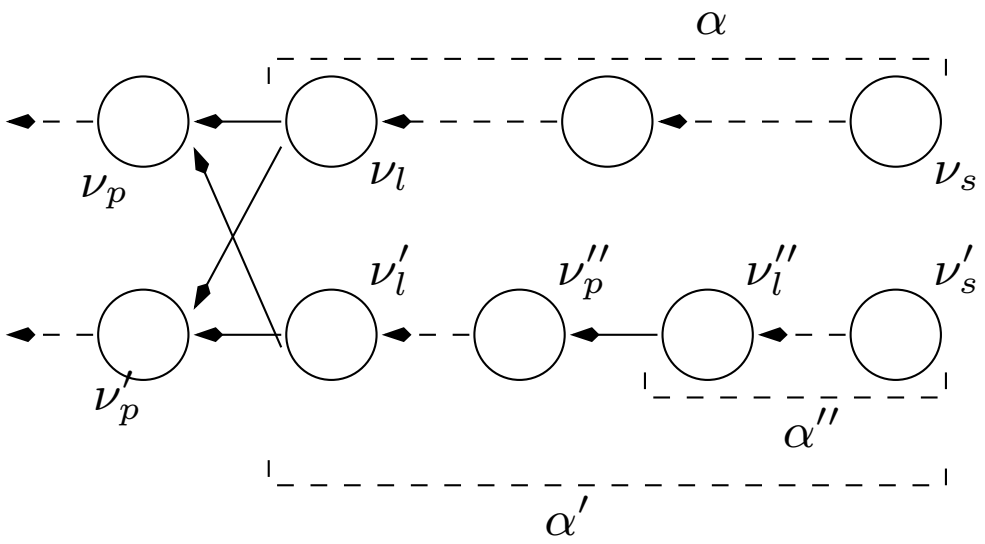
. correspondant au même état (donc au même symbole),

. et ayant les mêmes contextes gauches (c'est-à-dire ayant les mêmes ensembles de prédécesseurs) peuvent être fusionnés

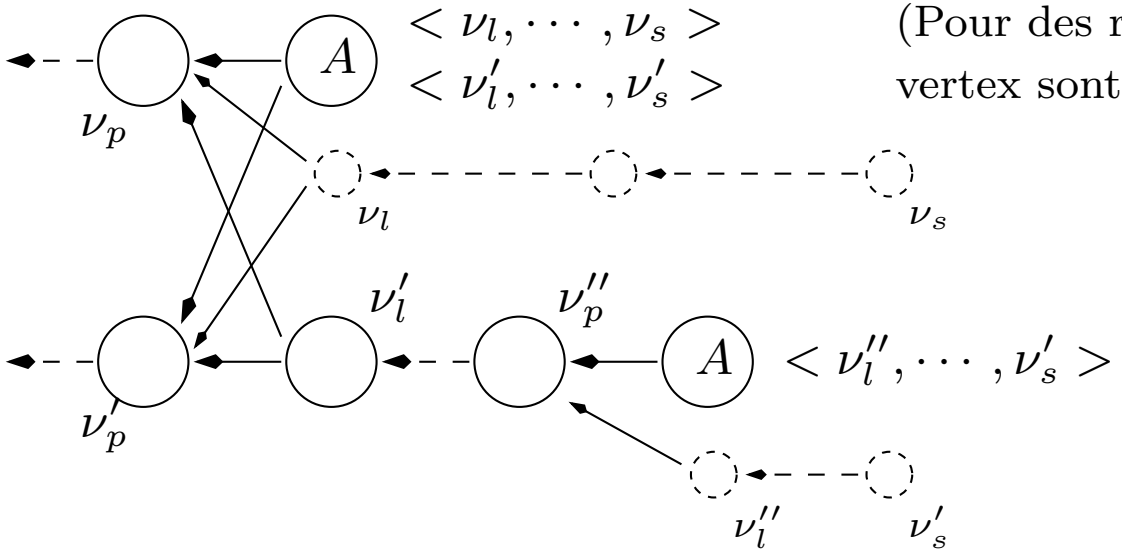


avant réductions
$A \rightarrow \alpha, A \rightarrow \alpha', A \rightarrow \alpha''$

Analyse GLR et partage des nœuds du GSS - III



avant réductions
$A \rightarrow \alpha, A \rightarrow \alpha', A \rightarrow \alpha''$



(Pour des raisons de présentation, les vertex sont dessinés à coté du nœud)

après réductions
$A \rightarrow \alpha, A \rightarrow \alpha', A \rightarrow \alpha''$

Principes de l'algorithme GLR original - I

- On notera respectivement $State(\nu)$ et $Vertex(\nu)$ l'état et le vertex du nœud ν du GSS
- On notera $Symb(V)$ le symbole du vertex (i.e., la racine de l'ensemble des sous-arbres du vertex) d'un nœud
- On notera $Pred(\nu)$ l'ensemble des prédécesseurs de ν dans le GSS
- L'action à effectuer dans un état s sur une fenêtre w sera notée $Action(s, w)$ (on ne s'intéresse qu'aux entrées non "vides")
- Un ensemble $Frontier$de couples $\eta = (\nu, i)$ donne les actions i à effectuer sur les sommets $State(\nu)$ des piles courantes
- L'ensemble Top donne l'ensemble des nouveaux nœuds créés entre deux décalages

Principes de l'algorithme GLR original - II

– Initialement, on crée ν_0 tel que $State(\nu_0) = q_0, Vertex(\nu_0) = \emptyset$, et $Frontier = \{(\nu_0, i) \mid i = Action(s_0, w)\}$

– On répète ensuite

1. $Top = \emptyset$
2. tant qu'il existe $\eta = (\nu, i) \in Frontier$, $A \xrightarrow{i} \alpha$, ôter η de $Frontier$, et effectuer la réduction
3. soit S l'ensemble des ν tels que $(\nu, 0) \in Frontier$
   . ôter tous les $(\nu, 0)$ de $Frontier$
   . effectuer le décalage

jusqu'à

. $Frontier = \{(\nu, \text{accept})\}$ (entrée reconnue, forêt partagée dans $Vertex(\nu)$),

. ou $Frontier = \emptyset$ (entrée incorrecte)

Principes de l'algorithme GLR original - III

– Pour effectuer la réduction $A \xrightarrow{i} \alpha$ commandée par $\eta = (\nu, i)$

. pour chaque chemin $p = \nu_1 \cdots \nu_k$, où $\nu_k = \nu$ et $k = |\alpha|$

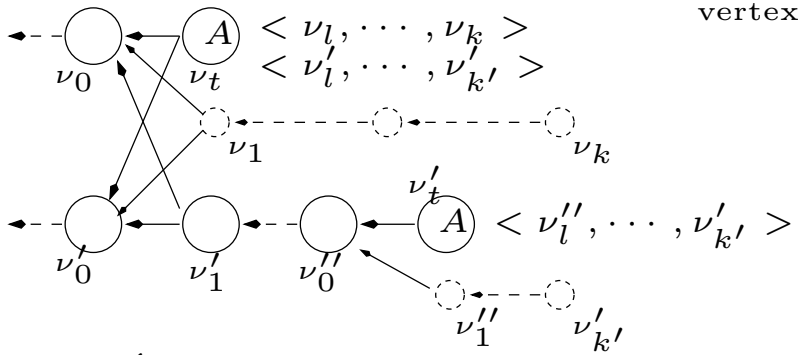
∗ partitionner l'ensemble $Pred(\nu_1)$: ν_0 et ν'_0 sont dans la même partition π_q si $Goto(State(\nu_0), A) = Goto(State(\nu'_0), A) = q$

∗ pour chaque ν_0 de π_q,

· s'il existe $\nu_t \in Top$ avec $Pred(\nu_t) = \pi_q$ et $Symb[\nu_t) = A$, ajouter le sous-arbre correspondant à p au vertex de ν_t

· sinon, créer ν_t tel que $State(\nu_t) = q$, $Vertex(\nu_t)$ avec A pour racine et le sous-arbre correspondant à p, établir les arcs tels que $Pred(\nu_t) = \pi_q$, ajouter ν_t à Top, et ajouter $\eta' = (\nu_t, j), j = Action(q, w)$ à $Frontier$

(Pour des raisons de présentation, les vertex sont dessinés à coté du nœud)



réductions
$A \rightarrow \alpha = \nu_1 \ldots \nu_k$
$A \rightarrow \alpha' = \nu'_1 \ldots \nu'_{k'}$
$A \rightarrow \alpha'' = \nu''_1 \ldots \nu'_{k'}$

Principes de l'algorithme GLR original - IV

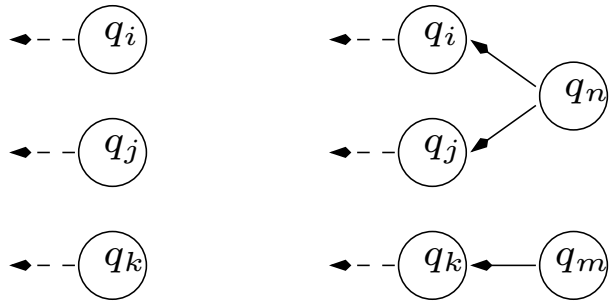
– Pour effectuer le décalage

. rappel transp. 16 : S est l'ensemble des nœuds de $Frontier$ pour un décalage; $Frontier$ est maintenant vide (mais pas S)

. Partitionner S : ν et ν' sont dans la partition π_{qa} si $Goto(State(\nu), a) = Goto(State(\nu'), a) = q$, pour une fenêtre a

. Décaler, la fenêtre devenant b

. Pour chaque π_{qa}, créer un nœud ν' tel que $State(\nu') = q$ et $Vertex(\nu')$ est le terminal a, et $Pred(\nu') = \pi_{qa}$, et ajouter (ν', i) à $Frontier$, avec $i = Action(q, b)$



avant décalage — après décalage

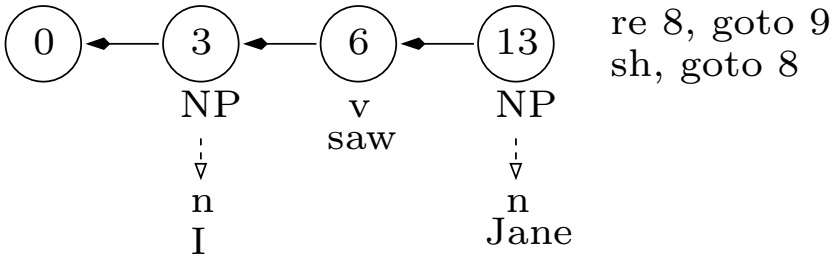
	$\cdots$	a	$\cdots$
$\cdots$			
q_i		sh,q_n	
$\cdots$			
q_j		sh,q_n	
$\cdots$			
q_k		sh,q_m	
$\cdots$			

table (LA)LR(1)

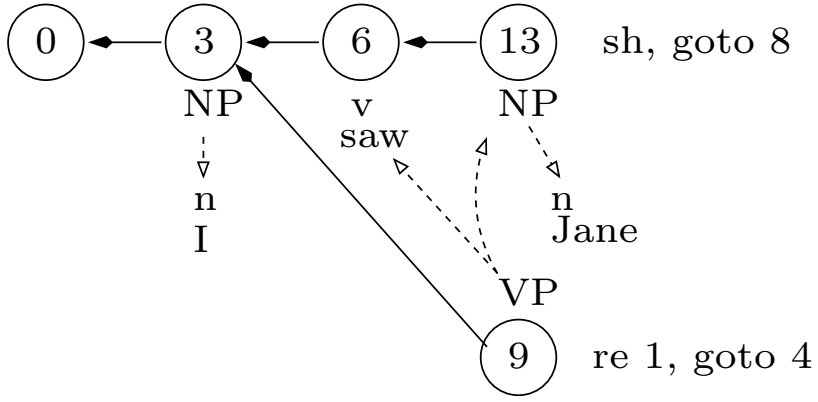
Exemple de fonctionnement de l'algorithme GLR - I

– On prendra la phrase *I saw Jane and Jack hit the man*

– Pas de problèmes pour analyser *I saw Jane*

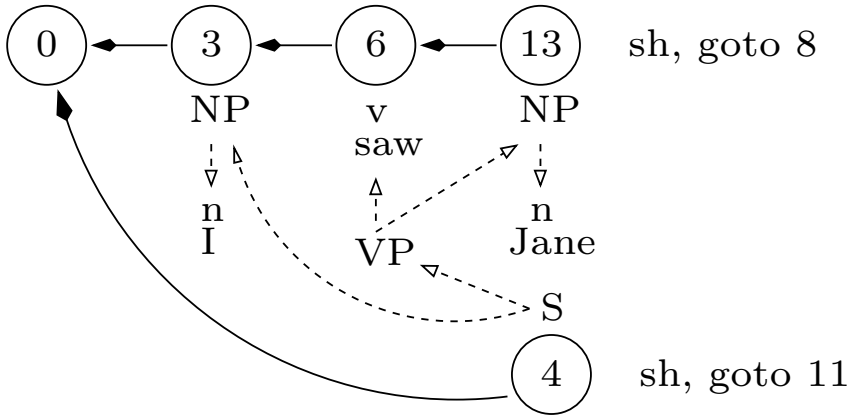


– Dans l'état 13 sur la fenêtre *and*, conflit entre VP $\xrightarrow{8} v$ NP et décalage, donc après la réduction

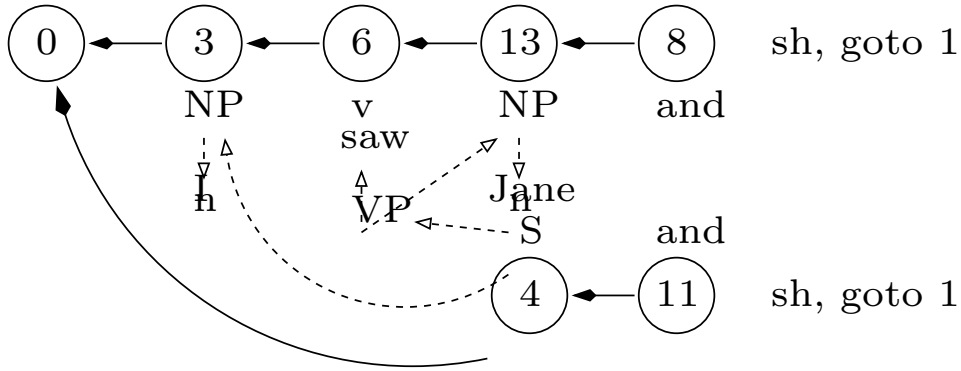


Exemple de fonctionnement de l'algorithme GLR - II

– Après la réduction dans l'état 9

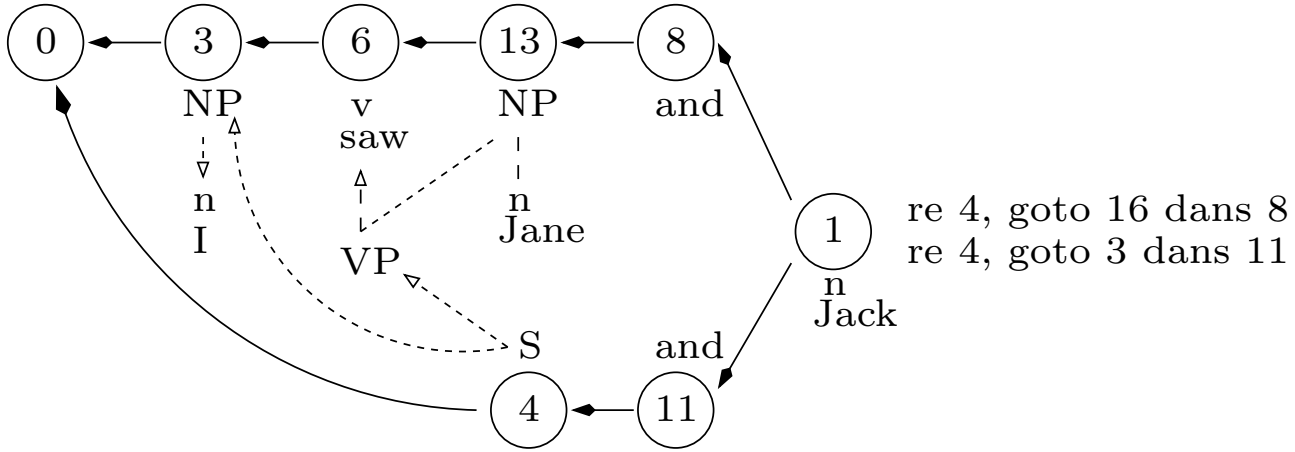


– Puis, après le décalage de *and*

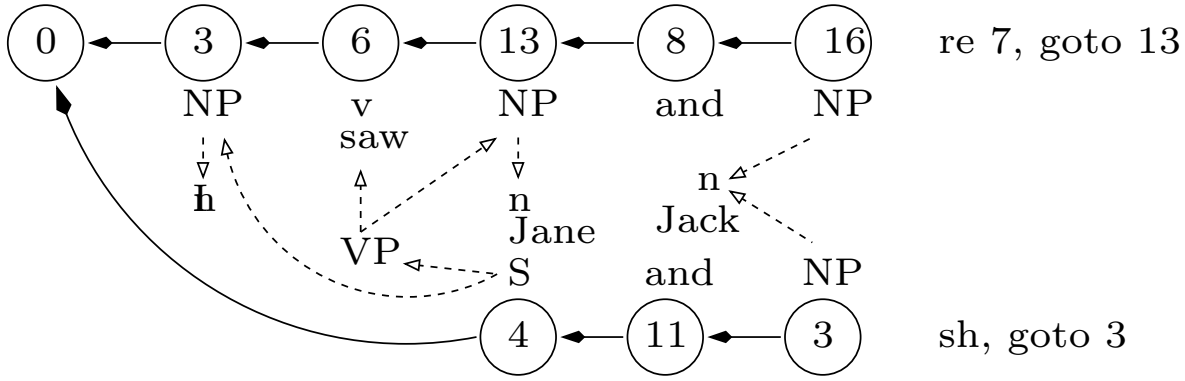


Exemple de fonctionnement de l'algorithme GLR - III

– Puis, après le décalage de *Jack*

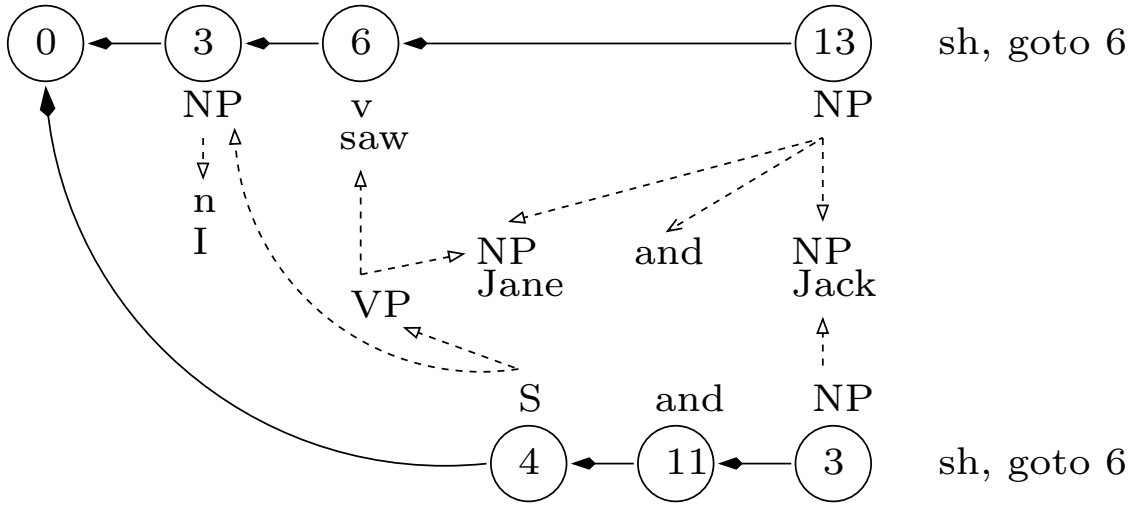


– Puis, après la réduction 4



Exemple de fonctionnement de l'algorithme GLR - IV

– Puis, après la réduction NP $\xrightarrow{7}$ NP and NP



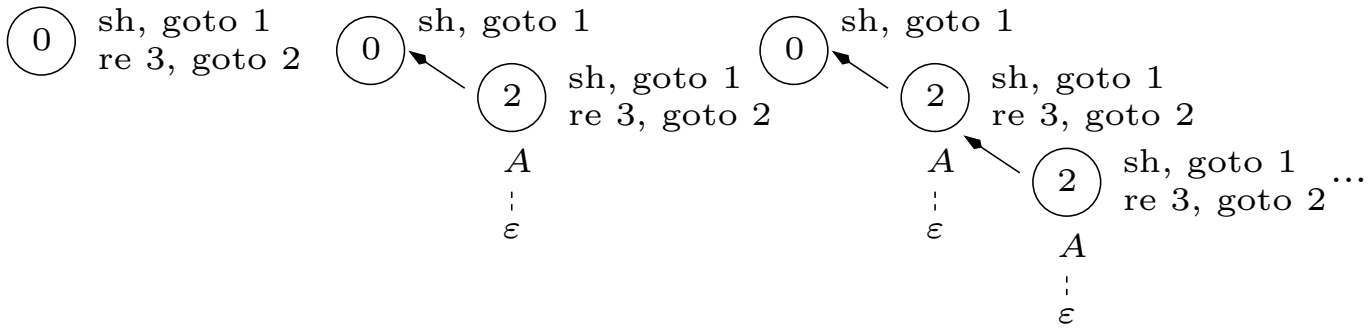
– Puis décalage de *hit*

– *hit the man* est analysé comme *v NP*

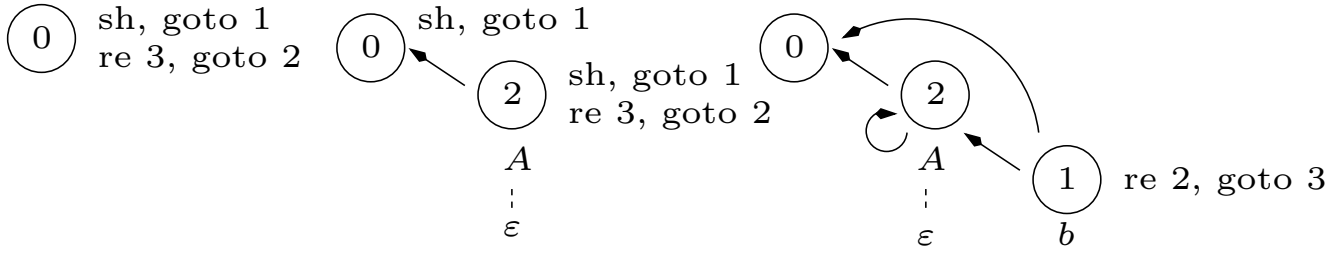
– On voit que l'analyse donnera *(I saw Jane) and (Jack hit the man)* d'une part, *I saw (Jane and Jack) hit the man* d'autre part

Problèmes avec les ε-productions

- L'algorithme boucle sur certaines grammaires avec ε-productions
- Par exemple, avec $G_{hlr} = \{S \rightarrow ASa,\ S \rightarrow b,\ A \rightarrow \varepsilon\}$, on ne sait pas combien de fois réduire le vide à A (récursivité gauche cachée)



- L'algorithme a été amélioré (Nozohoor-Farshi, 1990) pour éviter ce genre de problème en introduisant des boucles dans le graphe



Complexité en temps de GLR

- $O(n^m)$ pour la grammaire
  $G = \{S \rightarrow a,\ S \rightarrow S\,S,\ S \rightarrow S^{m+2}\}$ pour $m > 0$
- $O(n^{\sqrt{|G|}})$ pour la grammaire
  $G' = \{S \rightarrow A_i,\ A_i \rightarrow B_j A_i\ \ A_i \rightarrow B_j,\ B_j \rightarrow a\}$ pour $i \neq j$
- En fait, l'algorithme est quasi-linéaire sur la longueur de l'entrée pour des grammaires plus naturelles
- Des améliorations de l'algorithme permettent des performances au pire en $O(n^3)$
- Il est encore possible d'améliorer ces résultats pour des grammaires sans récursivités gauches cachées ni récursivités droites