III Parsing déterministe

Parsing déterministe : pour des langues acceptées par des automates push-down déterministes.

Plutôt intéressant pour la construction des compilateurs mais pas pour des langages naturels qui ne permettent pas de parsing déterministe.

L'interêt pour les langues naturels vient du fait que, basé sur l'algorithme LR, un algorithme de parsing déterministe, on peut developper un algorithme de parsing pour des langues hors contexte en général, comme p.e. celui de M. Tomita.

7 Parsing LR

Le parsing LR est un parsing bottom-up shift-reduce déterministe. LR = lire la phrase de gauche (Left) à droite et générer une dérivation droite (Right).

On distingue deux procédures :

1. `reduce` : le contenu de la pile représentant les catégories déjà reconnues est réduit par l'application d'une production
2. `shift` : le mot suivant est mis sur la pile

Les grammaire auxquelles un parsing LR peut être appliqué : Le parsing est déterministe en utilisant un Look-ahead de longuer k, ($k = 0, 1, \ldots$). Si une grammaire permet un parsing LR déterministe avec Look-ahead de longueur k, on l'appelle une *grammaire LR(k)*. D'habitude on considère des grammaires LR(0) ou LR(1).

Dans un parsing LR on construit l'inverse d'une dérivation droite.

Chaque étape de la dérivation à la forme γx avec $\gamma \in (V \cup T)^*, x \in T^*$ avec γ étant les catégories déjà reconnues et x correspondant à la partie de la phrase qui n'est pas encore lue. On appelle γ la *partie ouverte* et x la *partie fermée.*

Exemple 27

G_1: Productions NP $\rightarrow$ Det N, NP $\rightarrow$ N, Det $\rightarrow$ une, N $\rightarrow$ pomme, N $\rightarrow$ Jean, symbole initial NP

partie ouverte	*partie fermée*	*action*
	une pomme	
une	*pomme*	*shift*
Det	*pomme*	*reduce*
Det pomme		*shift*
Det N		*reduce*
NP		*reduce*

Grammaire LR(0) : il suffit de considérer la partie ouverte pour décider d'une manière déterministe s'il faut appliquer `shift` ou `reduce` et en plus, dans le dernier cas, pour décider même quelle production appliquer.

G_1 est LR(0).

Pour formaliser la propriété de LR(0), on a besoin

- du contexte gauche d'un $A \in V$ (ce qui peut apparaître à gauche de A dans une étape d'une dérivation droite).
  Dans G_1 le contexte gauche de N est {Det, ϵ}, celui de Det est {ϵ} et celui de NP est {ϵ} aussi)
- du contexte gauche d'une règle $A \rightarrow \alpha$ (le contexte gauche de A suivi de α).
  Dans G_1 le contexte gauche de NP $\rightarrow$ Det N est {Det N}, celui de NP $\rightarrow$ N est {N}, celui de Det $\rightarrow$ une est {une}, celui de N $\rightarrow$ pomme est {Det pomme, pomme} et celui de N $\rightarrow$ Jean est {Det Jean, Jean}.

LR(0): aucun contexte gauche d'une production est préfixe du contexte gauche d'une autre production.

Exemple 28

G_2: $S \rightarrow a\ S$, $S \rightarrow a$

Cette grammaire n'est pas LR(0): a fait partie du contexte gauche de $S \rightarrow a$ et a S fait partie du contexte gauche de $S \rightarrow a\ S$.

Non-déterminisme quand on ne considère que la partie ouverte :

$$\begin{array}{r|l|l} & a\ a & \\ a & a & shift \end{array}$$

Puis deux continuations possibles :

$$\begin{array}{l|c|l} S & a & reduce \\ \multicolumn{3}{c}{ou} \\ a\ a & & shift \\ a\ S & & reduce \\ S & & reduce \end{array}$$

La première est à choisir si la partie fermée est déjà vide, la deuxième est à choisir si la partie fermée contient encore des a.

Définition 8 (Grammaire LR(0))

$G = \langle V, T, P, S\rangle$ *soit une grammaire hors contexte.*

1. *Pour chaque* $A \in (V \cup T)^*$*, le contexte gauche de* A *est* $CG(A) := \{\gamma \,|\, S \overset{*}{\Rightarrow}_d \gamma A\alpha$ *avec* $\gamma, \alpha \in (V \cup T)^*\}$*. (*$\Rightarrow_d$ *= dérivation droite)*
2. *Pour chaque* $A \to \alpha \in P*$*, le contexte gauche de* $A \to \alpha$ *est* $CG(A \to \alpha) := \{\gamma\alpha \,|\, \gamma \in CG(A)\}$*.*
3. G *est un syntaxe LR(0) ssi pour toutes les productions* $X \to \alpha, X' \to \alpha' \in P, X \to \alpha \neq X' \to \alpha'$*, aucun* $\gamma \in CG(X \to \alpha)$ *est préfixe d'un* $\gamma' \in CG(X' \to \alpha')$*.*

Grammaire LR(k) : grammaire qui permet un parsing déterministe en considérant la partie ouverte et les k premiers symboles de la partie fermée de chaque étape de dérivation.

G_1 est LR(0), G_2 est LR(1) : si le premier symbole de la partie fermée est un a, il faut faire un shift. Si la partie fermée est ϵ et la partie ouverte n'est pas S, il faut faire des réductions, soit avec S $\to$ a (si le dernier symbole de la partie ouverte est a), soit avec S $\to$ a S (si le dernier symbole de la partie ouverte est S).

Définition 9 (Grammaire LR(k))

$G = \langle V, T, P, S\rangle$ *soit une grammaire hors contexte. G est LR(k) (*$k \geq 0$*) ssi*

pour toutes deux dérivations droites de la forme

$S \overset{*}{\Rightarrow}_d \alpha X w \Rightarrow_d \alpha\beta w$ *et*

$S \overset{*}{\Rightarrow}_d \gamma Y x \Rightarrow_d \alpha\beta y$

avec $w, x, y \in T^*$*,* $X, Y \in V$ *:*

Si $init_k(w) = init_k(y)$*,* $\alpha X y = \gamma Y x$ *c'est-à-dire* $\alpha = \gamma, X = Y, x = y$*.*

($init_k$ *= premiers* k *symboles)*

Autrement dit : si la partie ouverte $\alpha\beta$ permet une réduction, elle et les k permiers symboles de la partie fermée w déterminent d'une façon unique la production $A \to \beta$ avec laquelle il faut faire la réduction et on ne peut pas arriver à une autre réduction possible en faisant d'abord des shifts.

Exemple 29

1. G_2 *:* $S \to a$*,* $S \to a\ S$
   - $k = 0$*:*
     $S \Rightarrow_d a$ *et* $S \Rightarrow_d a\ S \Rightarrow_d a\ a$
     Ne satisfaisont pas la condition avec $X = S$*,* $\alpha = \epsilon$*,* $\beta = a$*,* $w = \epsilon$*,* $\gamma = a$*,* $Y = S$*,* $x = \epsilon$*,* $y = a$*.*

- $k = 1$:
  Chaque γ avec $S \to \gamma$ qui contient des symboles non-terminaux a la forme $\gamma = a^n S$ pour un $n \geq 0$.
  La grammaire satisfait alors trivialement la condition pour LR(k) pour tous les $k \geq 1$ parce que des pairs de dérivations droites comme dans la définition de LR(k) avec $w \neq \epsilon$ n'existent pas.

2. G_3: $S \to A\ B$, $A \to a$, $B \to C\ D$, $B \to a\ E$, $C \to a\ b$, $D \to b\ b$, $E \to b\ b\ a$
   - $k = 0$:
     G_3 n'est pas LR(0) : $a \in CG(A \to a)$, $a\ C\ D \in CG(B \to C\ D)$.
   - $k = 1$:
     G_3 n'est pas LR(1) :
     $S \Rightarrow A\ B \Rightarrow A\ C\ D \Rightarrow A\ C\ b\ b \Rightarrow A\ a\ b\ b\ b$
     $S \Rightarrow A\ B \Rightarrow A\ a\ E \Rightarrow A\ a\ b\ b\ a$
     avec $\alpha = A$, $X = C$, $\beta = a\ b$, $w = b\ b$, $y = b\ a$: $ACba \neq AaE$
     Ceci montre que dans une situation

partie ouverte	partie fermée
$A\ a\ b$	$b \ldots$

     ou on ne connaît pas ce qui suit le premier b dans la partie fermée, il y a deux possibilités qui pourraient mener vers une analyse :

partie ouverte	partie fermée	action
$A\ a\ b$	$b\ b$	
$A\ C$	$b\ b$	reduce
$A\ C\ b$	b	shift
$A\ C\ b\ b$		shift
$A\ C\ D$		reduce
$A\ B$		reduce
S		reduce

partie ouverte	partie fermée	action
$A\ a\ b$	$b\ a$	
$A\ a\ b\ b$	a	shift
$A\ a\ b\ b\ a$		shift
$A\ a\ E$		reduce
$A\ B$		reduce
S		reduce

   - $k = 2$
     G_3 est LR(2) :
     Les $\alpha X w$ possibles avec $|w| \geq 2$ sont :
     (a) $A\ C\ b\ b$: $S \stackrel{*}{\Rightarrow} A\ C\ b\ b \Rightarrow A\ a\ b\ b\ b$ ($w = bb$)
         d'autres possibilités de dériver un $A\ a\ b\ b\ b\ \gamma$ avec $\gamma \in T^*$ n'existent pas
     (b) $A\ a\ b\ b\ b$: $S \stackrel{*}{\Rightarrow} A\ a\ b\ b\ b \Rightarrow a\ a\ b\ b\ b$ ($w = abbb$)
         autres $a\ a\ b\ \gamma$, $\gamma \in T^*$ dérivable : seul $a\ a\ b\ b\ a$:
         $S \stackrel{*}{\Rightarrow} A\ a\ b\ b\ a \Rightarrow a\ a\ b\ b\ a$. Satisfait la condition parce que $A\ a\ b\ b\ a = A\ a\ b\ b\ a$.
     (c) $A\ a\ b\ b\ a$: voir $A\ a\ b\ b\ b$. Satisfait aussi la condition parce que $A\ a\ b\ b\ b = A\ a\ b\ b\ b$.

3. G_4: $S \to A\ X$, $S \to Y\ C$, $X \to B\ C$, $Y \to A\ B$,
   $A \to a\ A$, $A \to a$, $B \to b\ B$, $B \to b$, $C \to c\ C$, $C \to c$
   Pour tous les k, $k \geq 0$, cette grammaire n'est pas LR(k) :
   k soit fixe.
   Il existe des dérivations droites
   $S \Rightarrow Y\ C \stackrel{k}{\Rightarrow} Y\ c^k \Rightarrow A\ B\ c^k$
   et
   $S \Rightarrow A\ X \Rightarrow A\ B\ C \stackrel{k-1}{\Rightarrow} A\ B\ c^{k-1}\ C \Rightarrow A\ B\ c^k$

Le parser LR consiste d'une table de parsing et d'une procédure de contrôle pour évaluer cette table. La table contient deux parties qui definissent les fonctions *action* et *goto*.

La fonction *action* prend un état s et un symbole terminal comme arguments et retourne une des valeurs suivantes (On ne considère que LR(k), $k \leq 1$. Pour d'autres k il faudrait plus qu'un seul symbole terminal):

- `shift <nouvel état>`
- `reduce <production>`
- `accept`
- `error`

La fonction *goto* prend un état s et un symbole non-terminal comme arguments et retourne un nouvel état s'. (*goto* calcule le nouvel état après l'application d'un `reduce`.)

Une configuration du parser est une pair avec le contenu de la pile (la partie ouverte) et le reste de la phrase (la partie fermée). Chaque partie ouverte correpond à un état (l'inverse n'est pas nécessairement le cas, c'est-à-dire plusieurs parties ouvertes peuvent correspondre au même état). Dans une configuration les états auxquels correpondent des préfixes de la partie ouverte sont marqués entre les symboles de la partie ouverte:

$$< s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m, w_i w_{i+1} \ldots w_n \$ >$$

Le symbole \$ marque la fin de la phrase. La partie ouverte ϵ correspond à l'état s_0 et $X_1 \ldots X_j, j \leq m$ correpond à l'état s_j.

Pour une configuration actuelle $< s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m, w_i w_{i+1} \ldots w_n \$ >$, la configuration suivante est calculée en utilisant $action(s_m, w_i)$:

- si $action(s_m, w_i) =$`shift` s, la nouvelle configuration est
  $< s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m w_i s, w_{i+1} \ldots w_n \$ >$
- si $action(s_m, w_i) =$`reduce` $A \rightarrow \alpha$ avec $r = |\alpha|$, la nouvelle configuration est
  $< s_0 X_1 s_1 X_2 s_2 \ldots X_{m-r} s_{m-r} A s, w_i w_{i+1} \ldots w_n \$ >$ avec $s = goto(s_{m-r}, A)$
- si $action(s_m, w_i) =$`accept`, la phrase est acceptée et l'algoritme s'arrête
- si $action(s_m, w_i) =$`error`, la phrase n' est pas acceptée et l'algoritme s'arrête

Exemple 30

G_1:

1. NP → Det N   2. NP → N
3. Det → une   4. N → pomme   5. N → Jean

Table de parsing :

état	*action*				*goto*		
	une	*pomme*	*Jean*	*\$*	*Det*	*N*	*NP*
0	*s1*		*s2*		*3*	*4*	*7*
1		*r3*					
2				*r5*			
3		*s5*				*6*	
4				*r2*			
5				*r4*			
6				*r1*			
7				*acc*			

Correspondance parties ouv. – états :

partie ouv.	*état*
	0
une	*1*
Jean	*2*
Det	*3*
N	*4*
Det pomme	*5*
Det N	*6*
NP	*7*

champ vide = error

(si = `shift` *i, ri =* `reduce` *i)*

Parsing de "une pomme" :

partie ouverte	*partie vermée*	*action*
0	*une pomme* $	
0 une 1	*pomme* $	*s1*
0 Det 3	*pomme* $	*r3 avec goto(0,Det)*
0 Det 3 pomme 5	$	*s5*
0 Det 3 N 6	$	*r4 avec goto(3,N)*
0 NP 7	$	*r1 avec goto(0,NP)*
0 NP 7	$	*accept*

La procédure de contrôle ne pose pas de problème, seul la construction de la table de parsing est complexe. Plusieurs méthodes, la méthode considérée ici s'appelle la méthode LR canonique (CLR).

On calcule d'abord une séquence d'ensembles d'arcs LR(1). Chacun de ces ensembles représente un état. Puis, dans une deuxième étape, on calcule la table.

Un arc LR(1) est une pair qui se compose

- d'une production divisée $A \rightarrow \alpha.\beta$ $(A \rightarrow \alpha\beta \in P)$ et
- d'un Look-ahead $x \in T \cup \{\$\}$

Intuition : une pair $\langle A \rightarrow \alpha.\beta, x\rangle$ correspond à une situation ou α est déjà reconnu, β est encore à reconnaître, et une fois qu'on aura reconnu $\alpha\beta$, le premier symbole de la partie fermée sera x.

Une production $A \rightarrow \alpha$ peut être utilisée pour une réduction ssi il existe un $a \in T$ tel que a est le premier symbole de la partie fermée et $\langle A \rightarrow \alpha., a\rangle$ fait partie de l'état actuel.

Chaque fois qu'on ajoute un nouvel arc dans un ensemble, il faut faire la clôture de cet ensemble en faisant toutes les expansions.

Procédure closure

```
Données globales :  CFG G = ⟨V,T,P,S⟩
Input :  ensemble I d'arcs LR(1)
Output :  ensemble d'arcs LR(1)
Variable locale :  changed, valeur initiale true
Méthode :
   while changed
      changed := false
      for all ⟨A → α.Bβ,a⟩ from I:
         for all B → γ from P:
            for all b from FIRST(βa):
              if ⟨B → .γ,b⟩ ∉ I
              then
                 changed := true
                 I := I ∪ {⟨B → .γ,b⟩}
   return(I)
```

Intuition derrière `closure` : La tache de reconnaître une catégorie B dans un état s (p.e. NP dans l'état 0 dans l'exemple précédante avec $ comme reste de la phrase) comprend la tache de reconnaître un γ avec $B \rightarrow \gamma$. Si B est suivi d'un $\beta \neq \epsilon$ (p.e. reconnaître Det suivi de N dans l'état 0), cela veut dire qu'il faut dans l'état s reconnaître un γ avec $B \rightarrow \gamma$ (p.e. *une* pour Det) avec un $b \in FIRST(\beta a)$ (p.e. *pomme* $\in FIRST$(N)) comme look-ahead.

Une procédure `goto-states` calcule l'état nouveau pour un état (ensemble d'arcs I) donné et un symbole non-terminal ou terminal X. Idée : X était premier symbole de la partie encore à reconnaître. On vient de trouver un X, dans l'état nouveau X est alors dernier symbole de la partie déjà reconnue d'une production divisée.

Procédure goto-states

```
Données globales :  -
Input :  ensemble I d'arcs LR(1), X ∈ V ∪ T
Output :  ensemble d'arcs LR(1)
Variable locale :  ensemble d'arcs LR(1) J
Méthode :
   J := {⟨A → αX.β, a⟩ | ⟨A → α.Xβ, a⟩ ∈ I}
   return(closure(J))
```

Dans l'algorithme principal, on ajoute une production $S' \to S$ à la grammaire et la séquence C d'états est initialisée avec [closure($\{S' \to .S, \$\}$)]. Puis on calcule à partir de cette valeur initiale tous les autres états. Ceci est fait par `compute-states`:

Procédure compute-states

```
Données globales :  CFG G = ⟨V,T,P,S⟩
Input :  séquence C d'ensembles d'arcs LR(1)
Output :  séquence d'ensembles d'arcs LR(1)
Variable locale :  changed, valeur initiale true
Méthode :
   while changed
      changed := false
      for all I from C:
         for all X from V:
            if goto-states(I, X) ≠ ∅ ∧ goto-states(I, X) ∉ C
            then
              changed := true
              C := C + [goto-states(I, X)]
   return(C)
```

L'algorithme principal qui calcule d'abord la séquence d'états et puis la table de parsing :

Procédure CLR(1)

```
Données globales :  CFG G = ⟨V,T,P,S⟩ avec S → S' ajouté à P
Input :  CFG G = ⟨V,T,P,S⟩ avec S → S' ajouté à P
Output :  table de parsing pour G/false
Variable locale :  C, valeur initiale true
Méthode :
   C := [closure({⟨S' → .S, $⟩})]
   C := compute-states(C)
   for all i from {0,...,|C| − 1}:
      if ⟨A → α.aβ, b⟩ ∈ C_i ∧ goto-states(C_i, a) = C_j ∧ a ∈ T
      then action(i, a) = shift j
      if ⟨A → α., a⟩ ∈ C_i
      then action(i, a) = reduce A → α
      if ⟨S' → S., $⟩ ∈ C_i
      then action(i, $) = accept
   for all i from {0,...,|C| − 1}:
      for all X from V:
         if goto-states(C_i, X) = C_j
         then goto(i, X) = j
```

L'état initial du parser est l'état construit à partir de $\langle S' \rightarrow S., \$\rangle$.

Exemple 31

G_2: 1. $S \rightarrow a$ 2. $S \rightarrow a\ S$

Construction des états :

C_0 `closure`$(\{\langle S' \rightarrow .S, \$\rangle\})$
$S' \rightarrow .\ S, \$$
$S \rightarrow .\ a, \$$
$S \rightarrow .\ a\ S, \$$

C_1 `goto-states`(C_0, S)
$S' \rightarrow S\ ., \$$

C_2 `goto-states`(C_0, a)
$S \rightarrow a\ ., \$$
$S \rightarrow a\ .\ S, \$$
$S \rightarrow .\ a, \$$
$S \rightarrow .\ a\ S, \$$

C_3 `goto-states`(C_2, S)
$S \rightarrow a\ S\ ., \$$

`goto-states`$(C_1, a) = \emptyset$, `goto-states`$(C_1, S) = \emptyset$, `goto-states`$(C_2, a) = C_2$, `goto-states`$(C_3, a) = \emptyset$, `goto-states`$(C_3, S) = \emptyset$.

Table de parsing :

état	*action*		*goto*
	a	\$	*S*
0	*s2*		*1*
1		*acc*	
2	*s2*	*r1*	*3*
3		*r2*	

Parsing de a a a :

0	*a a a* \$	
0 a 2	*a a* \$	*s2*
0 a 2 a 2	*a* \$	*s2*
0 a 2 a 2 a 2	\$	*s2*
0 a 2 a 2 S 3	\$	*r1*
0 a 2 S 3	\$	*r2*
0 S 1	\$	*r2*

Parsing LR pour des grammaires hors contexte en général : Si on a une grammaire hors contexte sans contraintes, on peut toujours construire une table de parsing avec l'algorithme `CLR(1)` (sauf quelques modifications pour permettre plusieurs valeurs par champ), mais les entrées dans la table ne sont plus uniques. On peut avoir plusieurs entrées par champ.

Exemple 32 *Grammaire G :*

1	$S \rightarrow NP\ VP$	*2*	$S \rightarrow S\ PP$	*3*	$NP \rightarrow N$
4	$NP \rightarrow det\ n$	*5*	$NP \rightarrow NP\ PP$	*6*	$PP \rightarrow prep\ NP$
7	$VP \rightarrow v\ NP$				

Table de parsing :

état	*action*					*goto*			
	det	*n*	*v*	*prep*	$	*NP*	*PP*	*VP*	*S*
0	s3	s4				2			1
1				s6	acc		5		
2			s7	s6			9	8	
3		s10							
4			r3	r3	r3				
5				r2	r2				
6	s3	s4				11			
7	s3	s4				12			
8				r1	r1				
9			r5	r5	r5				
10			r4	r4	r4				
11			r6	r6,s6	r6		9		
12				r7,s6	r7		9		

Phrase : n v n prep det n (p.e. Jean regarde Marie avec un téléscope)

Deux analyses :

partie ouverte	*partie fermée*
0	n v n prep det n $
0 n 4	v n prep det n $
0 NP 2	v n prep det n $
0 NP 2 v 7	n prep det n $
0 NP 2 v 7 n 4	prep det n $
0 NP 2 v 7 NP 12	prep det n $
0 NP 2 VP 8	**prep det n $**
0 S 1	prep det n $
0 S 1 prep 6	det n $
0 S 1 prep 6 det 3	n $
0 S 1 prep 6 det 3 n 10	$
0 S 1 prep 6 NP 11	$
0 S 1 PP 5	$
0 S 1	$

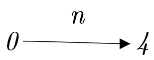
partie ouverte	*partie fermée*
...	...
0 NP 2 v 7 NP 12	prep det n $
0 NP 2 v 7 NP 12 prep 6	**det n $**
0 NP 2 v 7 NP 12 prep 6 det 3	n $
0 NP 2 v 7 NP 12 prep 6 det 3 n 10	$
0 NP 2 v 7 NP 12 prep 6 NP 11	$
0 NP 2 v 7 NP 12 PP 9	$
0 NP 2 v 7 NP 12	$
0 NP 2 VP 8	$
0 S 1	$

Idée de l'algorithme de Tomita : Pour éviter d'analyser une même partie plusieurs fois (p.e. le NP avec det n dans l'ex.) plusieurs parties ouvertes pour une même partie fermée sont mis dans un graphe acyclique directionné.

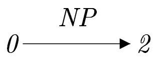
Exemple 33 *Prenons les deux analyses de n v n prep det n de l'exemple précédant.*

Evolution du graphe :

partie fermée v n prep det n $:

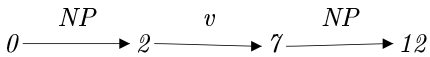


partie fermée v n prep det n \$:

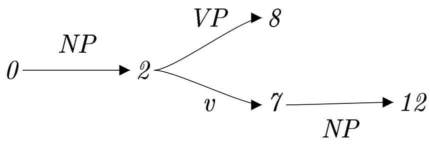


. . .

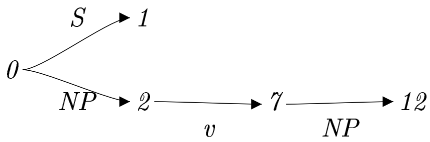
partie fermée prep det n \$:



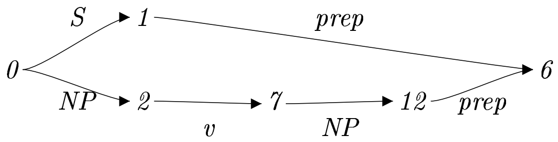
partie fermée prep det n \$:



partie fermée prep det n \$:

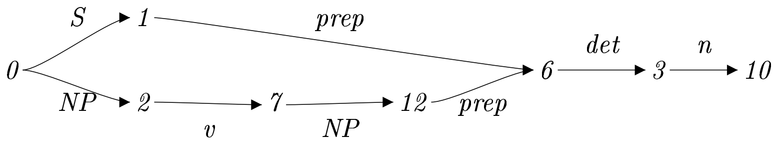


partie fermée det n \$:

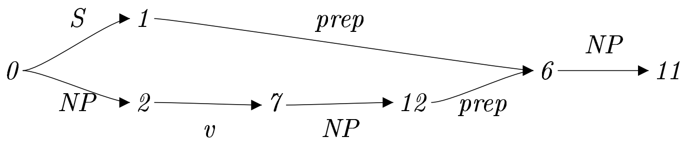


. . .

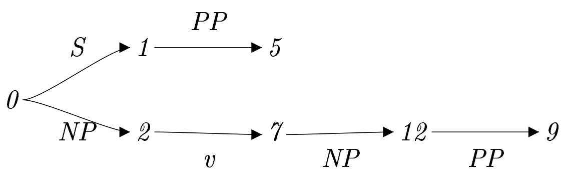
partie fermée \$:



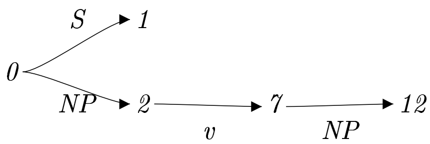
partie fermée \$:



partie fermée \$:



partie fermée \$:



partie fermée \$:

$$
\begin{tikzcd}
 & 1 & \\
0 \arrow[ur, "S"] \arrow[r, "NP"'] & 2 \arrow[r, "VP"'] & 8
\end{tikzcd}
$$

partie fermée \$:

$$
\begin{tikzcd}
0 \arrow[r, "S"] & 1
\end{tikzcd}
$$

On a trouvé une analyse quand on trouve un graphe avec partie fermée \$ qui contient un arc partant de l'état initial avec symbole S et qui mène vers un état duquel aucun autre arc part.