

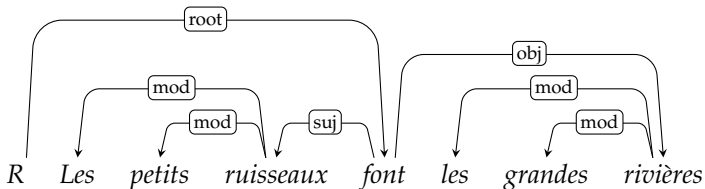
Analyse en transitions

Analyse en transitions

- L'analyse d'une phrase $S = w_1 \dots w_n$ est vue comme une séquence d'opérations (appelées transitions).
- L'exécution de ces opérations construit l'arbre de dépendance correspondant à S .
- Une opération s'applique sur une **configuration** pour produire une nouvelle configuration.
- Une configuration $C = (\sigma, \beta, \Delta)$ où
 - β est une file qui contient initialement tous les mots de S . β_0 est le premier mot dans la file
 - σ est une pile contenant des mots de S au fur et à mesure de leur lecture. σ_0 est le mot au sommet de la pile
 - Δ est l'ensemble des dépendances construites (l'arbre de dépendances)
- Configuration initiale : $C_0 = ([R], [w_1 \dots w_n], \{\})$
- Configuration d'acceptation : $(\sigma, [], \Delta)$

Example

stack	buffer	depset
R	les petits ruisseaux font les grandes rivières	
R les	petits ruisseaux font les grandes rivières	
R les petits	ruisseaux font les grandes rivières	(ruisseaux, mod, petits)
R les	ruisseaux font les grandes rivières	(ruisseaux, mod, les)
R	ruisseaux font les grandes rivières	
R ruisseaux	font les grandes rivières	(font, suj, ruisseaux)
R	font les grandes rivières	(R, root, font)
R font	les grandes rivières	
R font les	grandes rivières	
R font les grandes	rivières	(rivières, mod, grandes)
R font les	rivières	(rivières, mod, les)
R font	rivières	(font, obj, rivières)
R font rivières		
R font		
R		



Les transitions (arc standard)

Arc-gauche_r construit une dépendance étiquetée r ayant pour gouverneur le premier mot de la file (β_0) et pour dépendant le mot en sommet de pile (σ_0)

$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma, w_j|\beta, \Delta \cup \{(w_j, r, w_i)\})$$

Arc-droit_r construit une dépendance étiquetée r ayant pour gouverneur σ_0 et pour dépendant β_0

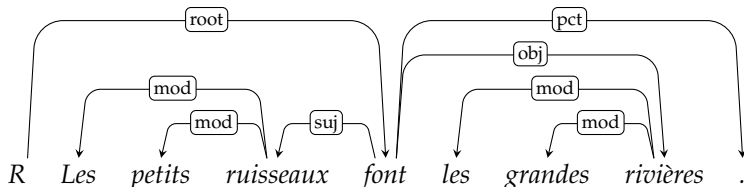
$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma, w_i|\beta, \Delta \cup \{(w_i, r, w_j)\})$$

Shift enlève β_0 de la file pour le mettre dans la pile

$$(\sigma, w_i|\beta, \Delta) \Rightarrow (\sigma|w_i, \beta, \Delta)$$

Exemple (arc standard)

stack	buffer	operation	depset
R	les petits ruisseaux font les grandes rivières .	SHIFT	
R les	petits ruisseaux font les grandes rivières .	SHIFT	
R les petits	ruisseaux font les grandes rivières .	LEFT-mod	(ruisseaux, mod, petits)
R les	ruisseaux font les grandes rivières .	LEFT-mod	(ruisseaux, mod, les)
R	ruisseaux font les grandes rivières .	SHIFT	
R ruisseaux	font les grandes rivières .	LEFT-suj	(font, suj, ruisseaux)
R	font les grandes rivières .	SHIFT	
R font	les grandes rivières .	SHIFT	
R font les	grandes rivières .	SHIFT	
R font les grandes	rivières .	LEFT-mod	(rivières, mod, grandes)
R font les	rivières .	LEFT-mod	(rivières, mod, les)
R font	rivières .	RIGHT-obj	(font, obj, rivières)
R	font .	SHIFT	
R font	.	RIGHT-pct	(font, pct, .)
R	font	RIGHT-root	(R, root, font)
R	R	SHIFT	



Prédiction des transitions

- Un classifieur prédit pour toute configuration la transition la plus vraisemblable.
- Une configuration est représentée sous la forme d'un vecteur de features.
- Modèle simple : les décisions sont (presque) indépendantes les unes des autres.
- La structure syntaxique produite (Δ) constitue une image des transitions effectuées jusque là.

Les transitions (arc eager)

Arc-gauche_r construit une dépendance étiquetée r ayant pour gouverneur le premier mot de la file (β_0) et pour dépendant le mot en sommet de pile (σ_0)

$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma, w_j|\beta, \Delta \cup \{(w_j, r, w_i)\})$$

Arc-droit_r construit une dépendance étiquetée r ayant pour gouverneur σ_0 et pour dépendant β_0

$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma|w_i|w_j, \beta, \Delta \cup \{(w_i, r, w_j)\})$$

Shift enlève β_0 de la file pour le mettre dans la pile

$$(\sigma, w_i|\beta, \Delta) \Rightarrow (\sigma|w_i, \beta, \Delta)$$

Reduce enlève σ_0 de la pile (dépile)

$$(\sigma|w_i, \beta, \Delta) \Rightarrow (\sigma, \beta, \Delta)$$

Recherche de la solution optimale

- Dans la plupart des cas, plusieurs actions peuvent d'appliquer à une configuration.
- La probabilité d'une solution est le produit des probabilités des actions qui la composent.
- La solution optimale est la solution ayant la probabilité la plus élevée :

$$\hat{s} = \arg \max_{s \in \mathcal{S}} P(s)$$

$$\hat{s} = \arg \max_{s=a_1 \dots a_n \in \mathcal{S}} \prod_{i=1}^n P(a_i)$$

- Mais l'espace de recherche $\mathcal{S}$ est de taille exponentielle par rapport à la longueur de la phrase à analyser.
- On adopte un comportement glouton :

$$\tilde{s} = \prod_{i=1}^n \arg \max_{a_i \in \mathcal{A}(C_{i-1})} P(a_i)$$

- La recherche de la solution est réalisée en temps linéaire, mais on n'a pas la garantie qu'il s'agit de la solution optimale.

Classifieur I

- Il prend en entrée une configuration c et produit une distribution de probabilités sur l'ensemble de toutes les actions possibles.
- Le nombre de configurations est infini, c'est la raison pour laquelle on introduit une fonction de décomposition (feature function) qui produit pour toute configuration c un vecteur de dimension n :

$$f : \mathcal{C} \rightarrow \mathcal{X}$$

- On apprend ensuite un classifieur $g : \mathcal{X} \rightarrow \mathcal{Y}$ où $\mathcal{Y}$ est l'ensemble des actions possibles.
- Les paramètres du classifieur sont appris sur un treebank, constitué de paires (S, A) où S est une phrase, et A est l'arbre qui lui est associé.

Classifieur II

- Pour apprendre le classifieur, les données doivent se présenter sous la forme de paires $(c_i, a_i) \in \mathcal{C} \times \mathcal{A}$
- On transforme les arbres A_i sous la forme d'une séquence de transitions $C = c_0 \dots c_m$ avec
 - 1 $c_0 = C_0(A_i)$
 - 2 $c_m = (\sigma, [], A_i)$
- cette transformation est réalisée à l'aide d'une fonction oracle.

Oracle pour arc eager

- Etant donné $c = (\sigma, \beta, \Delta) \in \mathcal{C}$, on définit $O(c)$ de la manière suivante
 - LEFT- l si $(\beta[0], l, \sigma[0]) \in A$
 - RIGHT- l si $(\sigma[0], l, \beta[0]) \in A$
 - REDUCE si $\forall l' \in \mathcal{L} \quad (\sigma[0], l', \beta[0]) \in A \Rightarrow (\sigma[0], l', \beta[0]) \in \Delta$
(tous les dépendants de $\sigma[0]$ dans A sont dans Δ .)
 - SHIFT sinon

Conversion

Etant donné un oracle O , on définit simplement la conversion d'un arbre A en une séquence d'actions $a_0 \dots a_n$ et de configurations $c_0 \dots c_{n+1}$:

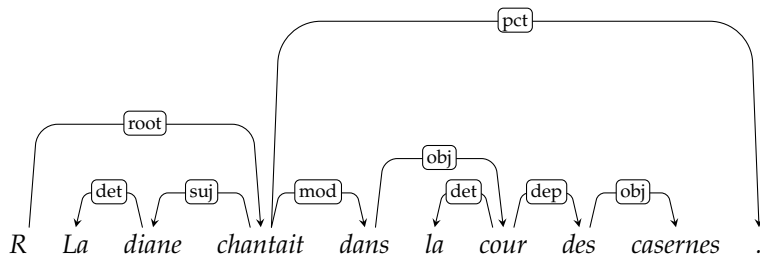
Algorithm 1 Conversion d'un arbre de dépendances en une séquence d'actions à l'aide d'un oracle

```
1:  $c_0 \leftarrow C_0(A)$   
2:  $i \leftarrow 0$   
3: while  $c_i$  n'est pas une configuration d'acceptation do  
4:    $a_i \leftarrow O(c_i)$   
5:    $c_{i+1} \leftarrow a_i(c_i)$   
6:    $i \leftarrow i + 1$   
7: end while
```

Oracle pour arc standard

- Etant donné $c = (\sigma, \beta, \Delta) \in \mathcal{C}$, on définit $O(c)$ de la manière suivante
 - LEFT- l si $(\beta[0], l, \sigma[0]) \in A$
 - RIGHT- l si $(\sigma[0], l, \beta[0]) \in A$ et si $(\sigma[0], l', \beta[0]) \in A \Rightarrow (\sigma[0], l', \beta[0]) \in \Delta$
(tous les dépendants de $\sigma[0]$ dans A sont dans Δ .)
 - SHIFT sinon

Exercice



- Produire la séquence de configurations et transitions correspondant à cette phrase avec un système arc eager et arc standard.

Fonction de décomposition (Feature function)

- Une feature représente un aspect d'une configuration qui semble intéressant pour prédire la transition correcte
- Exemples
 - la partie de discours du mot courant dans le buffer
 - la forme du mot se trouvant au sommet de la pile
 - le nombre de dépendants du mot se trouvant au sommet de la pile
 - la distance entre le mot courant et le mot se trouvant au sommet de la pile
 - ...
- On peut décomposer une feature en deux fonctions :
 - une fonction d'adresse qui identifie un mot particulier dans la configuration
 - une fonction d'attribut qui spécifie un attribut de ce mot
- Exemple

adresse	attribut
S[0]	FORM
S[0]	POS
B[0]	FORM
LDEP(S[0])	LABEL