Analyse syntaxique et application aux langues naturelles

Lexiques et pré-traitements syntaxiques

Jacques Farré

et

Sylvain Schmitz

Rôle du lexique

- Connaître l'ensemble des mots d'une langue et leurs informations associées (catégorie, genre, temps…)
- Éventuellement permettre de procéder à des corrections orthographiques
- Fournir les entrées de l'analyseur syntaxique (un peu comme le couple lex-yacc pour les langages de programmation)
- Quelques lexiques du français :
  - http://atilf.atilf.fr/tlf.htm
  - http://actarus.atilf.fr/lexiques/morphalou/
  - **http://www.lefff.net/**

Terminologie (1)

- *Token* : séquence de caractères encadrée par des séparateurs tels que espace, signe de ponctuation,…
- *Forme* : une unité élémentaire du point de vue syntaxique
- *Mot* : forme simple ou composée
- Exemples
  - *123* = 1 token, 1 mot, 1 forme (*_NOMBRE)*
  - *a priori* = 2 tokens, 1 forme, 2 mots
  - *aux* = 1 token, 1 mot, 2 formes (*à les*)
  - *donnez-moi* = 1 token, 2 mots, 2 formes

Terminologie (2)

- *Stemme* = racine d'un mot
- *Lemme* = classe d'équivalence de formes ayant des significations communes
- Les formes d'une même classe (lemme) diffèrent par leurs *traits morphologiques* (genre, nombre, temps,…)
  - *Forme décorée* = forme accompagnée de son lemme et de ses traits morphologiques
- Exemple de forme décorée :

écoutions : lemme = *écouter*
traits = verbe, indicatif, imparfait, 1ère personne, pluriel

Traitement du texte d'entrée : que donner à l'analyseur

- Qu'attend l'analyseur syntaxique ?
  - Des tokens ? Pas une bonne solution, difficile d'énumérer tous les tokens (nombres, adresses,…)
  - Des mots ? Un peu mieux, mais il y a des quantités de mots syntaxiquement équivalents (singulier/pluriel, masculin/féminin, verbes conjugués,…)
  - Dans les 2 cas, on perd les informations qu'on peut déduire des traits morphologiques, sauf à les intégrer dans les règles de la grammaire (qui devient d'une taille monstrueuse)
- Bon compromis :
  - lui fournir des catégories de mots/formes (verbe, nom,…) : lemme et traits sont alors des informations associées au mot, qui permettront de vérifier la correction fonctionnelle de la phrase

Traitement du texte d'entrée : mots artificiels

- Un texte ne comprend pas que des mots (au sens courant) et des signes typographiques, mais aussi des
  - sigles, abbréviations, balises : S.N.C.F., par ex., <h3>…</h3>
  - nombres, dates, horaires : 30, trente, 15 novembre 2006, 22h12
  - adresses (postales, email, URL), n° de téléphone, … : Jacques.Farre@unice.fr
  - etc.
- Des grammaires (régulières) permettent de reconnaître ces séquences de tokens, et le les transformer en mot « artificiel » : _nombre, _adresse, etc
- Comme ces mots peuvent contenir des signes de ponctuation, leur reconnaissance doit être faite avant le découpage du texte en phrases

Traitement du texte d'entrée : correction des fautes

- Un texte comprend aussi des fautes d'orthographe ou des erreurs typographiques
  - rares sur les documents littéraires, plutôt nombreuses sur les supports électroniques
- Il existe diverses techniques de correction, tenant compte ou non du contexte, notamment
  - Distance minimum de correction (distance de Levensthein) calcul du nombre d'insertions, suppression, remplacement pour passer d'un mot (erroné) à un mot connu
  - Par la composition de transducteurs définissant des règles élémentaires de correction (plus puissant que le calcul de distance)

Correction du texte d'entrée : distance de Levensthein

```
LevenshteinDistance (char str1[1..lg1], char str2[1..lg2]) : int
// d est une matrice lg1+1 × lg2+1
int d[0..lg1, 0..lg2] , i, j, cost
for i := 0 to lg1 do d[i, 0] := i
for j := 0 to lg2 do d[0, j] := j
for i := 1 to lg1 do
  for j := 1 to lg2 do
    if str1[i] = str2[j] then cost := 0 else cost := 1
    d[i, j] := minimum  (
                    d[i-1, j] + 1, // suppression
                    d[i, j-1] + 1, // insertion
                    d[i-1, j-1] + cost) // substitution )

return d[lg1, lg2]
```

		s	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
s	1	0	1	2					
u	2								
n	3								
d	4								
a	5								
y	6								⊠

Correction du texte d'entrée : exemple de calcul de la distance de Levensthein

		s	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
s	1	*0*	*1*	*2*	3	4	5	6	7
u	2	1	1	2	*2*	3	4	5	6
n	3	2	2	2	3	*3*	4	5	6
d	4	3	3	3	3	4	*3*	4	5
a	5	4	3	4	4	4	4	*3*	4
y	6	5	4	4	5	5	5	4	*3*

```
s a t u r d a y
s     u n d a y
```

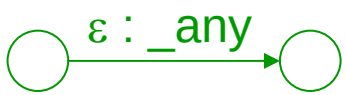
On peut de plus tenir compte de l'inversion de lettres
(distance de Damerau-Levenshtein)

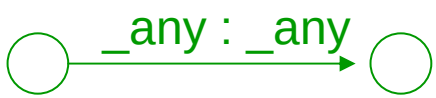
```
i n v r e s e r
i n v e r s e r
```

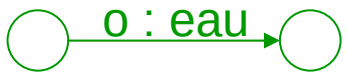
pour réduire le coût

Correction du texte d'entrée : utilisation de règles et de transducteurs

- Les règles sont exprimées par des transducteurs T_i
  - insertion :
  - suppression :
  - substitution :
  - correction phonétique :
  - …



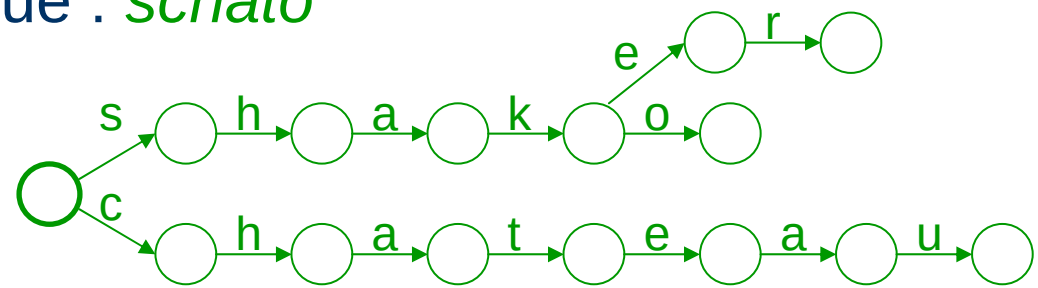




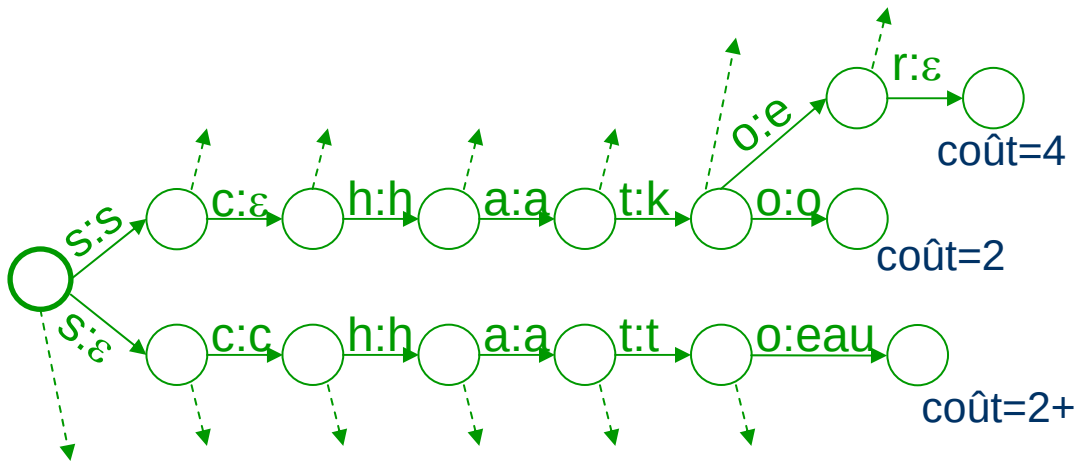
- Le lexique est vu comme un automate fini *A*
  - On calcule l'intersection de *A* avec une composition (finie) des T_i prenant le mot à corriger en entrée

Correction du texte d'entrée : exemple d'utilisation de règles et de transducteurs

- Mot inconnu du lexique : *schato*
- Une partie de *A* :



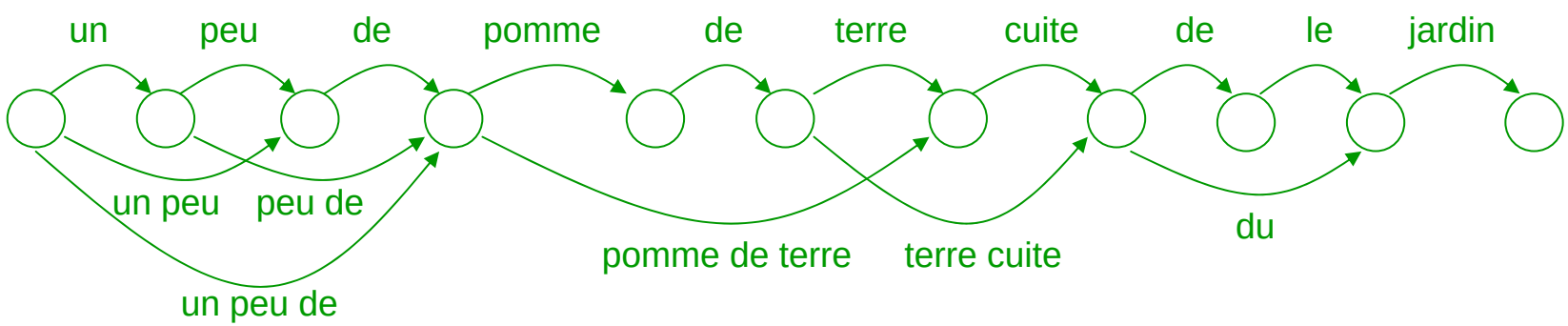
- Application des règles T_i et intersection avec *A* (vue partielle)



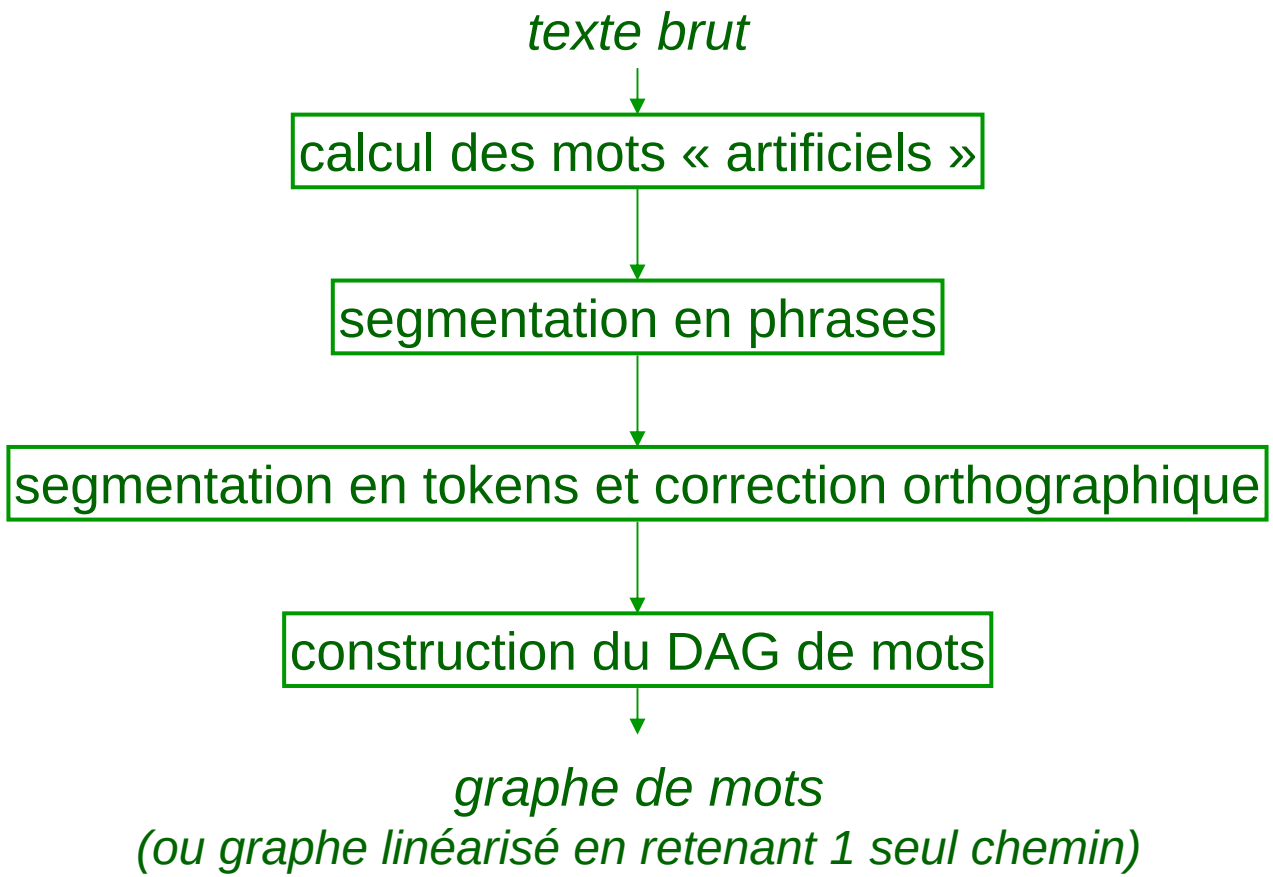
Résultat du texte d'entrée : DAG de mots

- Ambiguïté des mots du lexique (*couvent = nom ou verbe*), mots agglutinés (*aux → à les*), mots composés (grand rue) : pas de correspondance simple entre tokens et formes/mots
  - Transformation de l'entrée en graphe orienté acyclique (DAG) :

    un peu de pomme de terre cuite du jardin



Une architecture possible de « lexeur »



Mécanismes morphologiques

- En général, les langues ont des mécanismes relativement homogènes pour construire les formes d'un lemme
  - Par exemple en français, le *s* est souvent une marque de pluriel, et les verbes en *–er* dérivent fréquemment un nom en *–eur* : *chanter→chanteur, râler→râleur*
  - Ces mécanismes sont des *paradigmes morphologiques*
  - On peut les utiliser pour construire le lexique à partir d'un lemme
  - Ou inversement, pour calculer les formes possibles correspondant à un mot (lemmes et traits)

Construction/utilisation du lexique

- Comment associer un mot à ses formes ?
- En utilisant les paradigmes morphologiques
  - Soit pour une *analyse morphologique*, c-à-d retrouver les composants des mots pour retrouver leur forme et leurs traits : anti-ségrégation-iste
  - Soit en utilisant un lexique morphologique, c-à-d qui contient tous les mots de la langue (on peut rêver) avec leur lemme et leurs traits :
    - En ce cas, on construit (plus ou moins manuellement) un lexique des lemmes, et on utilise les paradigmes morphologiques pour construire le lexique final : *chant* $\rightarrow$ *chanter* $\rightarrow$ *chanteur, chanteuse* $\rightarrow$ *chante,chantera,…*
  - Dans les 2 cas, besoin d'une *description morphologique* de la langue

Descriptions morphologiques

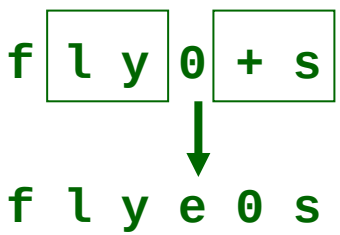
- Morphologie à 2 niveaux
  - Associe une forme de surface (celle qu'a le mot dans le texte) à une forme profonde (sa racine –ou stemme) à l'aide de différentes règles pour dériver les mots

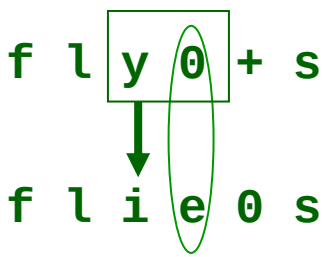
- Morphologies flexionnelle et dérivationnelle
  - Permet de passer d'un lemme à ses formes grâce à la classe morphologique à laquelle appartient le lemme

    La morphologie dérivationnelle permet en plus de prendre en compte des mots nouveaux (néologismes) à partir de mots existants : *chanter→chantage,* mais aussi *changer→changeage*

Morphologie à deux niveaux

- Le modèle comprend
  - Des règles, représentées par des transducteurs, qui sont appliquées en parallèle
  - Un lexique qui contient les mots sous leur forme profonde et code des contraintes
- Par ex. le nom commun anglais *fly* et son pluriel *flies* :
  - insérer un *e* entre un *y* précédé d'une ***C***onsonne et une frontière de morphème suivie d'un *s :* **0 : e ⇔ *C* y : _ + s**
  - un *y* devient un *i* devant un *e* : **y : i ⇔ _ 0 : e**





Morphologies flexionnelle

- Classes décrivant les flexions des lemmes
  - Possibilité d'héritage entre classe
    - sauf éventuellement pour certaines flexions
    - avec redéfinition possible de ces flexions
  - Exemple d'héritage de morphologie flexionnelle : le verbe *haïr* il se conjugue comme les verbes du 2ème groupe (**`v-ir2`**), sauf pour certaines conjugaisons (**`PJ`**) où il n'y a pas de ¨ sur le *i*

    Description en XML (= veut dire ôter le ¨ du i) :

```
<table name="v-haïr" rads="haï">
    <like name="v-ir2" except="PJ"/>
        <form target="=s" tag="P12S"/> je/tu hais
        <form target="=t" tag="P3S"/>  il hait
        ...
```

Morphologies dérivationnelle

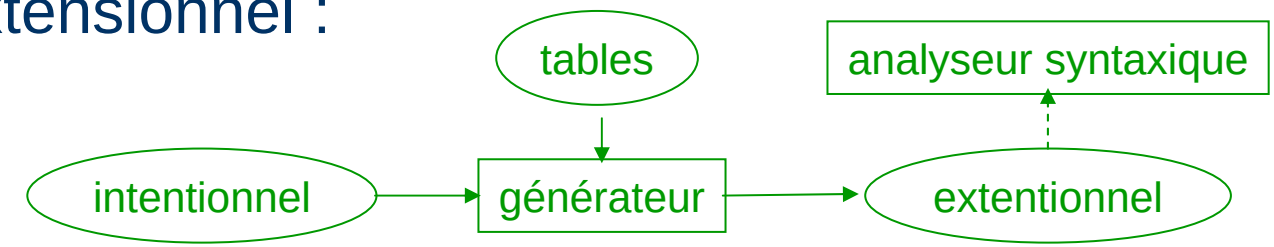
- Des tables indiquent les suffixes à ajouter au verbe
- Par ex., pour les dérivations des verbes du 1er groupe :

```
<table name = "v-er" rads="*">
 <derivation target="age" table="nc-2"/>    chantage
 <derivation target="at" table="nc-teur"/> examinateur
 …
```

- Formalisme assez concis, par exemple pour le français
  - 39 classes de verbes (3 pour 1er et 2ème groupes), 29 de noms, 18 d'adjectifs, 1 d'adverbes

Représentation intentionnelle/extentionnelle de lexiques (exemple du *Lefff*)

- Le lexique intensionnel, à l'aide des tables de flexion/dérivation, permet d'obtenir le lexique extensionnel :



  - Lexique intentionnel : chaque entrée est un triplet (racine, classe morphologique, classe syntaxique)

```
boire v-ir3 @verbe_standard
```

  - Lexique extentionnel : chaque entrée donne notamment: forme, catégorie, poids, informations syntaxiques

```
bois V [pred="boire<subj,(obj)>",…,@P12S]
```

    il peut y avoir plusieurs entrées pour une forme (ici bois nom commun)

Acquisition (automatique) de lexiques

- Créer un lexique à la main est fastidieux et source d'erreurs
  - Il en existe cependant (ATILF, MulText, ABU) qui peuvent être un point de départ
- Sinon, à partir d'un corpus relativement large, on peut construire le lexique en 3 phases
  - Génération de tous les lemmes possibles pouvant donner des formes du corpus
  - Classement des lemmes probables
  - Validation manuelle des lemmes les plus vraisemblables

Génération des lemmes possibles

1. Élimination des mots artificiels et segmentation du texte
2. Élimination de mots appartenant à des classes fermées (pronoms, prépositions,…)
3. Calcul du nombre d'occurrences de chaque forme
4. Calcul des lemmes en s'appuyant sur les tables de flexion/dérivation
   - Élimination de ceux qui ont été identifiés comme faux dans une itération précédente du processus, ou qui ne sont pas attestés par un nombre suffisant d'occurrences de leurs formes

Classement des lemmes probables

- Ce classement vise à ranger les lemmes les plus plausibles en tête
  - On part d'une estimation arbitraire de la plausabilité de chaque lemme
  - On itère jusqu'à stabilisation des plausabilités en calculant de nouvelles estimations à partir de la capacité des formes à dériver des lemmes
  - Exemple : on a 3 formes, *mangez*,*mange* et *mangerons* dont on déduit deux lemmes également plausibles *manger* et *mangerer*
    - itération 1 : *mangez* et *mange* augmentent la plausabilité de *manger* mais pas de *mangerer*;
    - itération 2 : *mangerons* confirme la plausabilité de *manger*

Calcul du classement des lemmes probables

- Pour un token (mot) *t* et forme une *f*
  - *P(f)* : probabilité que *t* soit une occurrence de *f*
    = $occ(f) / n_{tot}$ (nombre d'occurrence de f / nombre total de mots)
  - *P(l)* : probabilité que *t* soit une forme fléchie du lemme *l*
  - P(f/l) : probabilité conditionnelle d'avoir choisi une occurrence de *f* sachant qu'on a choisi une forme fléchie du lemme *l*
  - $\Pi(l)$ *:* probabilité que *l* soit un lemme correct
- Alors $P_{i+1}(f/l) = (P(f) - \Sigma_{l \neq l'} P_i(l')P_i(f/l')) / P_i(l)$

  $P_{i+1}(l) = occ(l).\ \Pi_{i+1}(l) / n_{tot}$ avec $occ(l)=\Sigma_f P_i(l).P_i(f/l)$

  $\Pi_{i+1}(l) = O_{i+1}(l) / (1+ O_{i+1}(l/f))$

  $O_{i+1}(l) = \Pi_f O_{i+1}(l/f)$, avec $O_{i+1}(l/f) = \Sigma_{l'} P_i(f/l') / \Sigma_{l' \neq l} P_i(f/l')$

(ces formules sont juste indicatives de la méthode, inutile de les apprendre)

Correction de lexiques

Lexiques syntaxiques (et sémantiques)

- http://framenet.icsi.berkeley.edu/