

Unitex2SxPipe

Antoine Désir, Benoît Sagot et Laurence Danlos
ALPAGE – INRIA/Paris 7

Séminaire d'informatique linguistique de l'IGM
26 novembre 2007

Unitex2SxPipe

Pour Ilimp... et plus si affinités

- Introduction
- SxPipe
- Ilimp
- Unitex2SxPipe

Antoine Désir – novembre 2007

Introduction (1)

- **Unitex** est un bon outil pour écrire des grammaires locales mais inconvenients :
 - Technique : assez lent
 - Théorique
 - Travaille sur du texte brut segmenté en phrases, impossible d'enchaîner plusieurs grammaires locales
 - Pas de gestion de l'ambiguïté, ni de la priorité d'un graphe par rapport à un autre
- → Création d'un outil qui traduit les grammaires **Unitex** en CFG pour **SxPipe**

Introduction (2)

- **SxPipe**
 - Chaîne de traitement : segmenteur, correcteur orthographique, entités nommées, etc.
 - Certains modules reposent sur un système de reconnaissance de motifs non contextuels
- Idée : traduire les graphes **Unitex** au format de ce système

Introduction (3)

- Win / win
 - Intégration de grammaires locales créées dans **Unitex** (p.ex. **ilimp**) à une chaîne de traitement plus large
 - **SxPipe** bénéficie des avantages d'une interface graphique

Iimp 1 (version Unitex)

Danlos (2005)

- Repère les instances impersonnelles (explétives) de *il*
- Ajoute les balises *[IMP]*, *[PERS]* ou *[AMB]*
 - *il pleut* → *[IMP] il [IMP] pleut*
 - *il part* → *[PERS] il [PERS] part*
 - *il est certain que P*
→ *[AMB] il [AMB] est certain que P*

Iimp – Principes théoriques (1)

- Extraction des têtes lexicales des constructions impersonnelles à partir du lexique-grammaire
- Intrinsèquement impersonnel

V31i	45 verbes	<i>il pleut</i>
V17	21 verbes	<i>il vaut mieux Vinf</i>
Vfige	38 expressions	<i>il était une fois</i>

Ilimp – Principes théoriques (2)

- Sujet profond extraposé
 - Sujet phrastique

etreAdj	682 adjectifs	<i>il est probable que P</i>
etrePrepX	88 expressions	<i>il est de règle de Vinf</i>
V5	21 verbes	<i>il plaît à X que P</i>
V6	140 verbes	passif (<i>il a été dit que P</i>)
V9	92 verbes	ou se-moyen (<i>il se raconte que P</i>)

- Sujet nominal

manquer / rester	<i>il reste du pain</i>
verbes inacusatifs ou passifs	<i>il est venu trois personnes</i> <i>il a été mangé du gâteau</i>

Iimp – Réalisation pratique (1)

- Contexte à gauche de la tête lexicale
 - *il ne nous semble vraiment pas probable que P*
 - Pas de problème réel

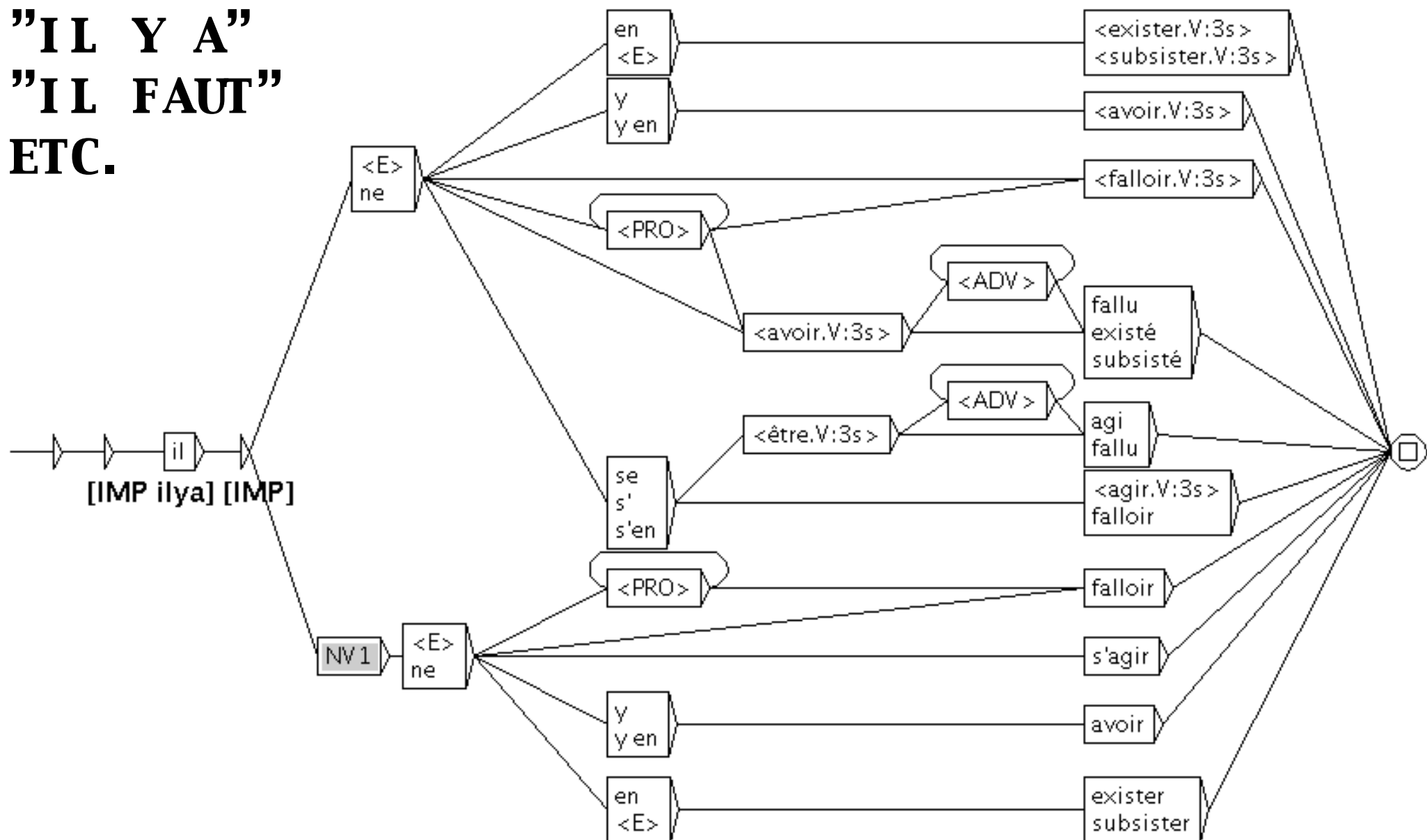
Iimp – Réalisation pratique (2)

- Contexte à droite de la tête lexicale
 - Nombreuses ambiguïtés → [AMB]
 - On peut limiter l'usage de cette balise dans certains cas, grâce à des heuristiques de fréquence
 - *il est difficile pour Ω <de V:W>*
 - *il [IMP] est difficile pour $(\Omega)_{GN}$ <de V:W>*
→ beaucoup plus fréquent
 - *il [PERS] est difficile pour $(\Omega$ <de V:W>) $_{GN}$*
 - Il reste des cas ambigus
 - *il [AMB] est certain que la grève durera au moins trois jours*
= *Paul / C'est certain que la grève [...]*

Iimp – Réalisation pratique (3)

- Quelques heuristiques "brutales"
 - *il y <avoir.V:3s>*
 - etc.

**"IL Y A"
"IL FAUT"
ETC.**



Ilimp – Taux de précision

97,5 %
sur corpus du Monde

Iimp – Coût (1)

- Trois passages
 - **I L <ÊTRE> <DET: F: P> <N F: P>**
 - *{ il [IMP] est des écrivains } qui sont empêchés de vivre*
 - Mais exception :
 - *{ il [PERS] est la coqueluche } des dames*
 - 1er passage pour les exceptions
 - 2e passage pour les cas réguliers

Iimp – Coût (2)

- Motifs imbriqués
 - *{ il [IMP] est probable qu' { il [IMP] reste } du pain }*
 - 1er passage
 - 2e passage pour les motifs imbriqués
- 3e passage : Balise [PERS] pour tous les *il* non balisés

Iimp – Coût (3)

- Chaque passage nécessite d'enregistrer les résultats dans un fichier intermédiaire
- Il faut appliquer les ressources lexicales aux fichiers intermédiaires, etc.

SxPipe – Introduction

- 1ère problématique :
 - On cherche à analyser des **corpus bruts**
 - Les analyseurs syntaxiques prennent en entrée des **suites (ou des DAG) de formes**
 - Une forme étant une entrée syntaxiquement atomique
 - Il faut donc savoir passer de l'un à l'autre
- 2ème problématique :
 - Certains traitements ne nécessitent pas une analyse syntaxique complète

SxPipe : formes, tokens, énoncés

- Un corpus brut est un flux de caractères
- Il faut donc découper ce flux
 - En unités typographiques raisonnables
→ les **tokens**
 - En éléments analysables individuellement
→ les **énoncés**
- Puis identifier les formes correspondant à chaque suite de tokens formant un énoncé

SxPipe : Formes et tokens (1)

- Mais il n'y a pas de correspondance biunivoque entre tokens et formes
 - Correspondances n à m : **amalgames**, **composés**, combinaison des deux
 - **Ambiguïtés** de correspondance
 - Fautes d'orthographe à corrections multiples
 - Ambiguïté de composition (*un peu de*)
 - **Formes spéciales** pour les (suites de) tokens formant une entité nommée syntaxiquement atomique
 - on les reconnaît par des grammaires locales

SxPipe : Formes et tokens (2)

au	à le
du	(du de le)
présen tation	présentation
uneforme	une forme
pese	(pesé pèse)
d' ores et déjà	d'ores_et_déjà
à l'instar du	à_l'instar_de le
aujourd' hui	aujourd'hui
2, rue de la Paix – 75008 Paris	_ADRESSE

SxPipe : Autres traitements

- Identifier des **motifs** syntaxiquement complexes
 - Pour **étiqueter** certaines formes
 - Pour **parenthéser** correctement certaines portions de l'entrée donnée à l'analyseur syntaxique
 - Pour repérer certains types d'expressions (**entités nommées** de type expressions temporelles, noms de personnes...)

SxPipe

- **SxPipe** est une chaîne de traitement pré-syntaxique complète
 - Ident. de la langue, tokenisation, segmentation
 - Découpage en formes / correction orthographique ambiguë
 - Nombreux types d'entités nommées (regexp, CFG)
- **Langues** traitées
 - Français, polonais
 - Préliminaire : anglais, slovaque, slovène
 - L'espagnol est prévu dans un avenir proche

SxPipe : reco. de motifs CFG (1)

- Idée générale : on veut reconnaître de façon ambiguë, dans des entrées ambiguës (DAG), des **motifs**
 - Pour identifier des entités nommées ou baliser des portions de textes
 - Pour étiqueter certains mots dans le texte
- La puissance des **CFG** est parfois nécessaire
- Elle n'est pas vraiment un handicap lorsqu'elle est superflue

SxPipe : reco. de motifs CFG (2)

- Mise en oeuvre :
 - Analyseur efficace à la **Earley** du système SYNTAX
 - En-têtes de grammaires prédéfinis pour n'avoir à écrire que les motifs
 - Possibilité d'utiliser un **lexique**
 - **DAG** en entrée + sortie (frontière de la forêt partagée)
 - Paramétrages pour **désambiguïsation** (partielle)
 - Garder ou non la lecture hors motifs
 - Garder ou non les séquences de motifs consécutifs qui sont elles-mêmes un motif
 - Garder ou non les motifs de longueur maximum

SxPipe : bilan

- Chaîne de traitement pré-syntaxique robuste
- En amont d'un analyseur syntaxique, permet
 - La **robustesse** (analyser n'importe quel corpus)
 - Le **guidage** (étiquetage, balisage, parenthésage, entités nommées, à l'aide de grammaires ultra-lexicalisées)
- En tant que tel, permet
 - La normalisation/correction de texte
 - Les entités nommées peuvent être une fin en soi

UniteX

Paumier (2006)

- Traite du texte à partir de dictionnaires électroniques, de grammaires locales et de tables de lexique-grammaire
- Détection de motifs
- Grammaires locales
 - graphes (et sous-graphes)
- \simeq Réseaux de transitions récursifs (RTN)
- "Transducteurs" (production de sorties)

Unitex – Fichiers utilisés

- .grf graphes
- .fst2 graphes compilés
- autolist grammaire "déployée"
- .dic dictionnaires

Unitex – Sous-programmes utilisés

- Grf2Fst2
 - Compile les graphes
- Fst2List [-p s -t s -s ' ' -s0 'xxx']
 - Génère le fichier autolist.txt
 - -p s séparation des sous-graphes
 - -t s production des sorties
 - -s ' ' insère un blanc entre chaque élément
 - -s0 'xxx' insère 'xxx' entre l'entrée et la sortie

Unitex vs. SxPipe

- Boucles
 - $S \rightarrow a^*$
- Codes sémantiques
 - $\langle N+duree \rangle / \langle N-duree \rangle$
- Un seul motif reconnu
 - Motifs ambigus
 - Motifs imbriqués

Unitex2SxPipe

- Ensemble de scripts Perl
- Créé et testé pour **lilimp** mais portable à d'autres grammaires locales

U2SP – Principes de base

- Création CFG
 - Fst2List
 - adaptation au bon format
- Création lexicale
 - Lefff → flexion des masques lexicaux (<**FAI RE. V: 3S**>)
 - Ajout des unités lexicales (*b e a u*)
 - Dictionnaires Unitex → Ajout des entrées lexicales spécifiées par des codes sémantiques (<**ADV+NEG**> → *j a m a í s*)

U2SP – Exemple

Lexique

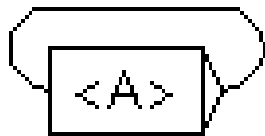
"BEAU"	' BEAU'	[] ;
"FAI SAI T"	' ===FAI RE. V: 3S==='	[] ;
"FAI T"	' ===FAI RE. V: 3S==='	[] ;
"FERA"	' ===FAI RE. V: 3S==='	[] ;
"I L"	I LI MP!	[] ;
"JAMAI S"	' ===ADV+NEG==='	[] ;
"MAUVAI S"	' MAUVAI S'	[] ;
"NEI GE"	' ===NEI GER. V: 3S==='	[] ;
"NEI GEAI T"	' ===NEI GER. V: 3S==='	[] ;
"NEI GERA"	' ===NEI GER. V: 3S==='	[] ;
"PAS"	' ===ADV+NEG==='	[] ;
"PLUS"	' ===ADV+NEG==='	[] ;

CFG

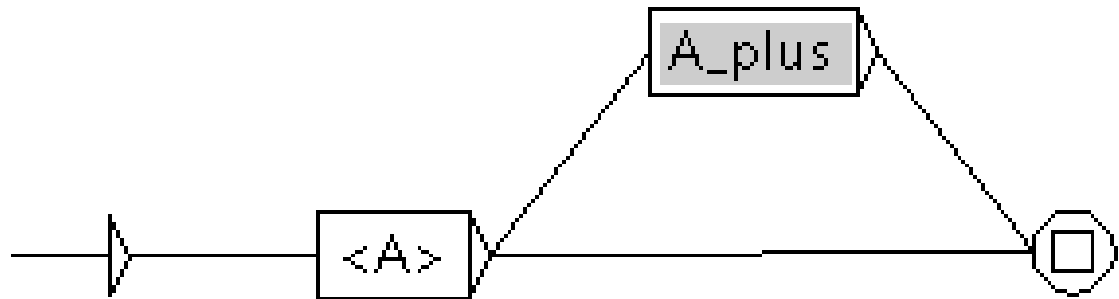
<PATTERN> = <V31I> ;
 <ADJ-V31I> = "BEAU" ;
 <ADJ-V31I> = "MAUVAI S" ;
 <T> = "===ADV+NEG===" ;
 <T> = "===FAI RE. V: 3S===" ;
 <T> = "===NEI GER. V: 3S===" ;
 <T> = "BEAU" ;
 <T> = "MAUVAI S" ;
 <V31I-3S> = "===NEI GER. V: 3S===" ;
 <V31I> = I LI MP! <V31I-3S> ;
 <V31I> = I LI MP! "===FAI RE. V: 3S===" <ADJ-V31I> ;
 <V31I> = I LI MP! "NE" "===FAI RE. V: 3S===" "===ADV+NEG===" <ADJ-V31I> ;

U2SP – Boucles simples

- Modification des graphes en amont
- Remplacées par des appels à des sous-graphes récurrents



Boucle



Sous-graphe A_plus

U2SP – Séquences composées

- Segmentées au trait d'union et à l'apostrophe
- Segmentation souhaitée varie selon les cas :
 - "y" "a" "-t-il"
 - "je-m'en-fichiste"
- Solution ad hoc : fichier de configuration
 - Pour ilimp, 5 séquences concernées

U2SP – Règles redondantes

- 100 000 règles (ilimp)
- Script de réduction par regroupement (items optionnels)

<PATTERN> = I L I M P! "===FALLOI R. V: 3S=== " ;

<PATTERN> = I L I M P! "NE" "===FALLOI R. V: 3S=== " ;

<PATTERN> = I L I M P! <_NE_OPT> "===FALLOI R. V: 3S=== " ;

<_NE_OPT> = "NE" ;

<_NE_OPT> = ;

- → 5000 règles

U2SP - Traits négatifs

Ces masques lexicaux (p.ex. **<N-DUREE>**) n'ont pas d'entrée dans le lexique → on génère une série de règles

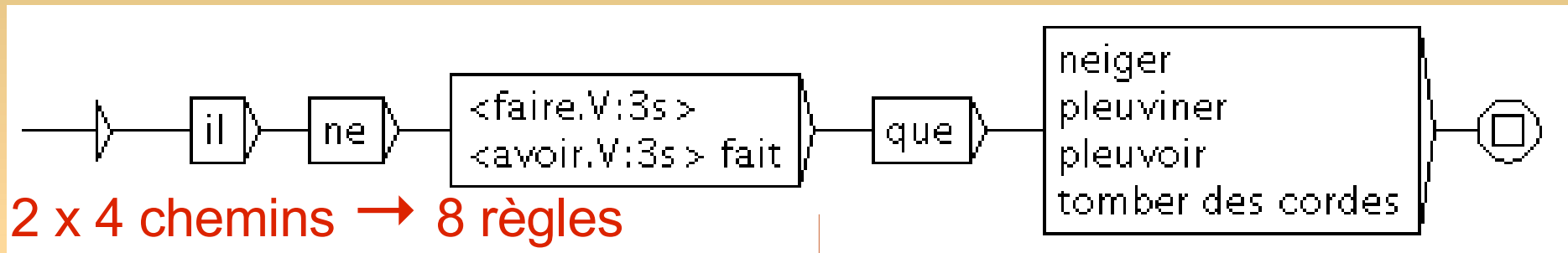
- **<N>** = "===N===" ;
 <N> = "===N+duree===" ;
 <N> = "===N+fois===" ;
- **<N-duree>** = "===N===" ;
 <N-duree> = "===N+fois===" ;
- **<N-FOI S>** = "===N===" ;
 <N-FOI S> = "===N+DUREE===" ;
- **<T>** = "===N===" ;
 <T> = "===N+duree===" ;
 <T> = "===N+fois===" ;

U2SP – Règles non prioritaires

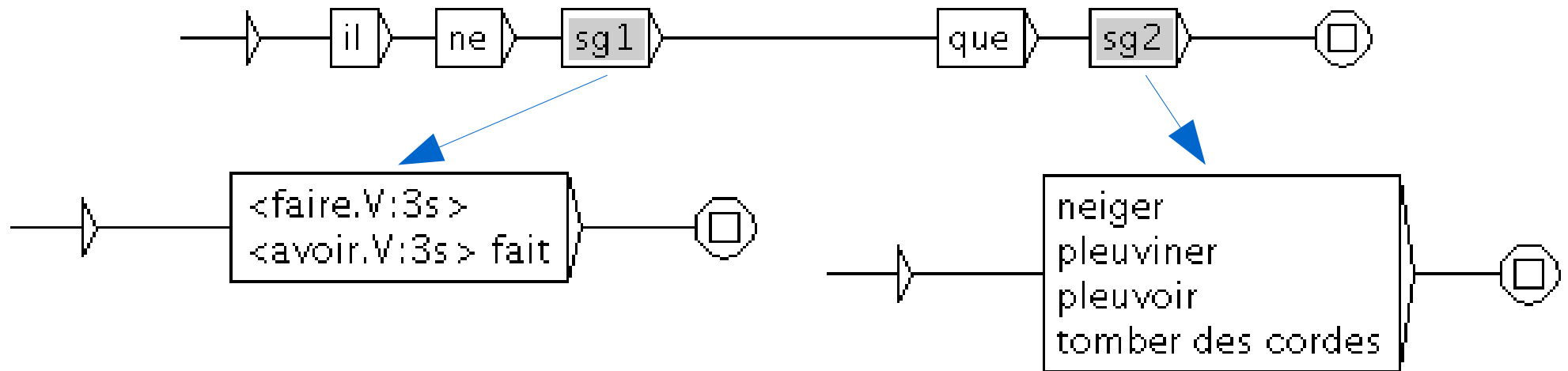
- Possibilité d'attribuer une priorité relative
 - $\langle \text{PATTERN} \rangle = \langle \text{ETREGNPERS} \rangle ;$
 $\langle \text{PATTERN} \rangle = \langle \text{ETREGN} \rangle ; -1$
 - *{ il_cln est la coqueluche } des dames*
 - *{ il_ilimp est des écrivains } qui sont empêchés de vivre*
- Motif doit recouvrir la même séquence

U2SP – Modification des graphes en amont

- Réduction du nombre de règles "à la source" (via modification des fichiers .grf)



1+2+4 chemins → 7 règles



U2SP – Fichier de configuration

- Sorties
- Règles non prioritaires
- Modifications CFG
 - Remplacements
 - Ajouts de règles
- Ajouts au lexique
- Balises
- Entités nommées

U2SP – Évaluation pour ilimp

- Sur "80 jours", résultats équivalents
- Autres évaluations à faire

U2SP – Problèmes en suspens

- Boucles complexes
- Autres, à déterminer

Développement futur

- Expérimentation sur d'autres grammaires
Unitex
 - time_french de M. Constant
 - autres (?)