Atelier ATOLL pour les grammaires d'arbres adjoints

François Barthélemy[1], Pierre Boullier[2], Philippe Deschamp[2],
Linda Kaouane[2] et Éric Villemonte de la Clergerie[2]
(1) CEDRIC - CNAM,
92 Rue St Martin - FR-75141 Paris Cedex 03
barthe@cnam.fr
(2) ATOLL - INRIA,
Domaine de Voluceau - BP 105 - 78153 Le Chesnay Cedex
Eric.De_La_Clergerie@inria.fr

1 Résumé – Abstract

Cet article présente l'environnement de travail que nous développons au sein de l'équipe ATOLL pour les grammaires d'arbres adjoints. Cet environnement comprend plusieurs outils et ressources fondés sur l'emploi du langage de balisage XML. Ce langage facilite la mise en forme et l'échange de ressources linguistiques.

This paper presents the ATOLL workbench for Tree Adjoining Grammars. This workbench provides several tools and resources based on the use of the markup language XML which eases the construction and the exchange of linguistic resources.

Mots-clefs : TAG, XML, Ressources linguistiques

2 Introduction

L'équipe ATOLL se consacre à l'étude et la réalisation d'outils pour le linguiste. Notre activité principale actuelle porte sur des analyseurs syntaxiques efficaces pour divers formalismes grammaticaux utilisés dans le traitement de la langue naturelle. Les grammaires d'arbres adjoints (*Tree Adjoining Grammars* – TAG) représentent un de ces formalismes, important sur le plan linguistique mais aussi sur le plan informatique, car permettant la réalisation d'analyseurs tabulaires efficaces s'exécutant en temps polynomial et nécessitant un espace en mémoire également polynomial.
Cependant, notre travail sur les TAG nous a fait prendre conscience du manque de standardisation des grammaires et de la difficulté d'utiliser certains des outils existants, en particulier lors du traitement de grandes grammaires. Le système XTAG[1] [The XTAG Research Group, 1995] fournit un standard implicite, qui manque cependant de lisibilité et de spécifications explicites. Nous avons de plus rencontré différentes variantes de XTAG. Enfin, certaines des grammaires étudiées présentent des problèmes de cohérence, dûs peut-être à un manque d'outils pour la gestion et le développement des TAG.
Nous avons donc été amenés à examiner le langage de définition des TAG et à réfléchir à des outils permettant d'exploiter une représentation standardisée. Suivant d'autres, dont le groupe

[1] http://www.cis.upenn.edu/~xtag/

LT XML[2] et plus particulièrement [Bonhomme and Lopez, 2000], nous avons conclu que le langage de balisage XML[3] est un choix judicieux pour représenter les TAG. En premier lieu, l'utilisation de DTD permet d'exprimer clairement la structure logique des grammaires. Ensuite, le format XML est purement textuel, ce qui permet un échange facile de ressources linguistiques entre environnements hétérogènes ainsi qu'une lecture immédiate par un humain. Enfin, l'ensemble des outils pour le traitement de documents XML croît très rapidement. Partant de l'emploi de XML pour les grammaires, nous nous sommes aussi rendu compte de son intérêt pour stocker des résultats linguistiques comme, par exemple, les forêts de dérivation produites par nos analyseurs.
Nous appuyant sur une représentation en XML, nous avons développé ou adapté plusieurs outils. Notre philosophie dans ce développement est de privilégier une approche modulaire plutôt que monolithique.
Après un bref rappel sur les TAG (section 3), nous décrivons dans la section 4 les codifications en XML des grammaires et des forêts de dérivation. Les outils de gestion et de conversion s'appuyant sur ces codifications sont présentés dans la section 5. La section 6 passe en revue nos analyseurs TAG et introduit un serveur d'analyseurs permettant un accès uniforme aux analyseurs. Nous montrons dans la section 7 comment ce serveur est utilisé par une interface permettant de visualiser dérivations et grammaires.

3 Grammaires d'arbres adjoints

Le formalisme des TAG [Joshi, 1987] est particulièrement adapté à la description de nombreux phénomènes linguistiques. Une TAG comprend un ensemble *d'arbres élémentaires* partitionné en *arbres initiaux* et *arbres auxiliaires*. Les nœuds internes sont étiquetés par des non-terminaux et les feuilles par des terminaux ou des non-terminaux. Chaque arbre auxiliaire β comporte une feuille distinguée, appelée *pied*, possédant la même étiquette que sa racine.

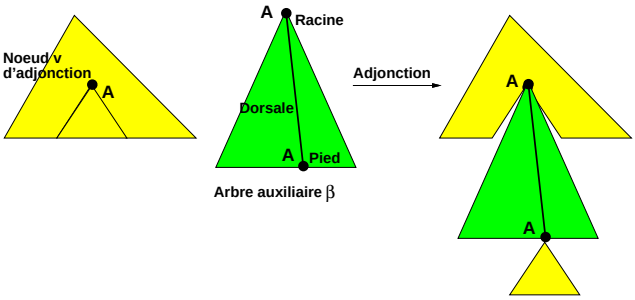


FIG. 1: Adjonction

Deux opérations servent à construire de nouveaux arbres à partir des arbres élémentaires. L'opération de *substitution* remplace une feuille ν par un arbre initial dont la racine possède la même étiquette que ν. L'opération d'*adjonction* est illustrée par la figure 1 : un arbre auxiliaire β dont la racine est étiquetée par un non-terminal A peut s'adjoindre sur un nœud ν également étiqueté par A ; le sous-arbre de racine ν est ensuite greffé au pied de β.
Les TAG avec structures de traits (*Feature TAG* – F-TAG) étendent les TAG en autorisant l'attachement sur chaque nœud d'une paire d'arguments « *haut* » et « *bas* », représentés par des structures de traits.

[2] `http://www.ltg.ed.ac.uk/`
[3] `http://www.w3c.org/XML/`

Les TAG lexicalisées (avec ou sans traits) imposent que chaque arbre élémentaire possède au moins une feuille lexicale, c'est-à-dire étiquetée par un terminal. Cependant, une grammaire totalement et explicitement lexicalisée serait gigantesque, avec de nombreux arbres pour chaque mot du langage. Le format XTAG permet de factoriser la description des grammaires et de donner suffisamment d'information pour lexicaliser à la demande des parties de la grammaire. Les entrées morphologiques (ou formes fléchies) regroupées dans un lexique morphologique font référence à des entrées syntaxiques (ou lemmes) regroupées dans un lexique syntaxique. Les lemmes font eux-même référence à des familles de schémas d'arbres. Un schéma d'arbres est simplement un arbre élémentaire avec une feuille distinguée appelée *ancre*, qui est remplacée lors de la lexicalisation par une forme fléchie. Chaque référence (à un lemme ou à des arbres) est éventuellement complétée par des contraintes additionnelles portant sur les arguments de l'ancre ou des autres nœuds des schémas d'arbres ou portant sur la lexicalisation d'autres feuilles (*co-ancres*) [4].
La figure 2 illustre ces différents composants sur un exemple issu d'une petite grammaire du français. Elle montre l'entrée morphologique pour `donne`, l'entrée syntaxique `\DONNER\` et le schéma d'arbres `tn1pn2` utilisé pour construire l'arbre lexicalisé `tn1pn2(donne)` correspondant au patron syntaxique (1). Le lemme stipule que le sujet NP_0 et le complément d'objet indirect NP_2 doivent être humains et que NP_2 doit être introduit par la préposition *à* (co-ancrage). Dans le schéma d'arbres `tn1pn2`, les nœuds de substitution sont indiqués par $\downarrow$ et le nœud ancre par $<>$.

(1) *quelqu'un* **donne** *quelque chose* **à** *quelqu'un*

```
donne: \DONNER\, V
       {mode=ind,num=sing}
\DONNER\,V: tn1pn2[p_2=à]
       {NP_0.t:restr=+hum,
        NP_2.t:restr=+hum}
```

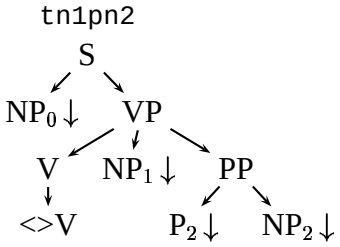


FIG. 2: Schéma d'arbre

4 Codification en langage XML

4.1 Représentation des grammaires

Nous avons conçu une DTD[5] qui spécifie clairement la structure logique des divers composants d'une TAG. L'extrait suivant de la DTD stipule par exemple qu'une entrée morphologique est introduite par une balise `morph`, est caractérisée par un champ `lex` servant de clé, et inclut une ou plusieurs références à un lemme (`lemmaref`). On peut également lui adjoindre des éléments de description (`desc`) pour la documentation. De même, un élément `lemmaref` est caractérisé par ses champs `name` et `cat`, et peut être complété par une structure de traits `fs`.

```
<!ELEMENT  morph  (desc*,lemmaref+)>
<!ATTLIST  morph   lex   CDATA   #REQUIRED>
<!ELEMENT  lemmaref  (fs?)>
<!ATTLIST  lemmaref   name   CDATA   #REQUIRED
                      cat    CDATA   #REQUIRED>
```

[4] En pratique, la distinction entre ancre et co-ancre n'est pas toujours très nette.

[5] `http://atoll.inria.fr/~clerger/tag.dtd,xml`

En respectant cette DTD, les composants de la figure 2 sont décrits par le fragment de XML suivant, en omettant les structures de traits pour des raisons de place et de simplicité.

```
<morph lex="donne">
      <lemmaref cat="v" name="*DONNER*">
          <fs>
              <f name="mode"><val>ind</val></f>
              <f name="num"><val>sing</val></f>
          </fs>
      </lemmaref>
</morph>
<lemma cat="v" name="*DONNER*">
      <anchor tree_id="family[@name=tn1pn2]">
          <coanchor node_id="p_2"> <lex>à</lex> </coanchor>
          <equation node_id="np_0" type="top">
              <fs><f name="restr"><val>plushum</val></f></fs>
          </equation>
          <equation node_id="np_2" type="top">
              <fs><f name="restr"><val>plushum</val></f></fs>
          </equation>
      </anchor>
 </lemma>
 <family name="tn1pn2">
     <tree name="tn1pn2">
          <node cat="s" adj="yes" type="std">
              <node cat="np" id="np_0" adj="no" type="subst" />
              <node cat="vp" adj="yes" type="std">
                  <node cat="v" adj="yes" type="anchor" />
                  <node cat="np" adj="no" type="subst" />
                  <node cat="pp" adj="yes" type="std">
                      <node cat="p" id="p_2" adj="no" type="subst" />
                      <node cat="np" id="np_2" adj="no" type="subst" />
                  </node>
              </node>
          </node>
     </tree>
 </family>
```

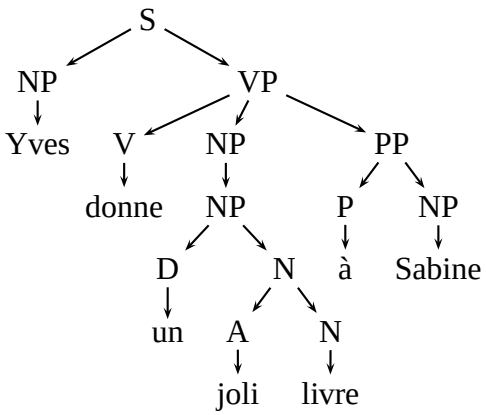
Pour l'instant, nous disposons sous cette forme XML d'une petite grammaire du français (50 schémas d'arbres, 117 lemmes et entrées morphologiques) et d'une grammaire de l'anglais (456 schémas d'arbres, 333 lemmes et 507 entrées morphologiques). Nous sommes en train de convertir d'autres grammaires plus importantes (pour le français et l'anglais).

4.2 Code pour les dérivations

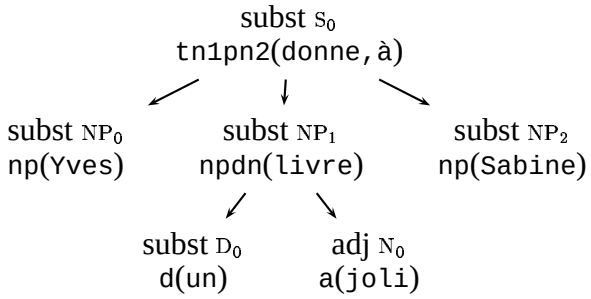
Pour une phrase non ambigüe, un analyseur syntaxique TAG retourne soit le résultat de l'analyse sous forme d'un arbre d'analyse, soit les étapes de l'analyse sous forme d'un arbre de dérivation. Ces deux possibilités sont illustrées pour la phrase (2) par les figures 3(a) et 3(b), la figure 3(c) explicitant les arbres élémentaires impliqués. L'arbre de dérivation indique quelles opérations (substitutions et adjonctions) sont appliquées sur quels nœuds et quels arbres élémentaires sont utilisés pour ce faire. Ainsi, l'arbre de la figure 3(b) signale par exemple l'adjonction de l'arbre $\texttt{a}(\texttt{joli})$ sur le nœud NP_1 (de $\texttt{npdn}(\texttt{livre})$). Noter que les arbres d'analyse peuvent se construire à partir des arbres de dérivation, ce qui motive notre intérêt pour ces derniers.

(2) Yves donne un joli livre à Sabine

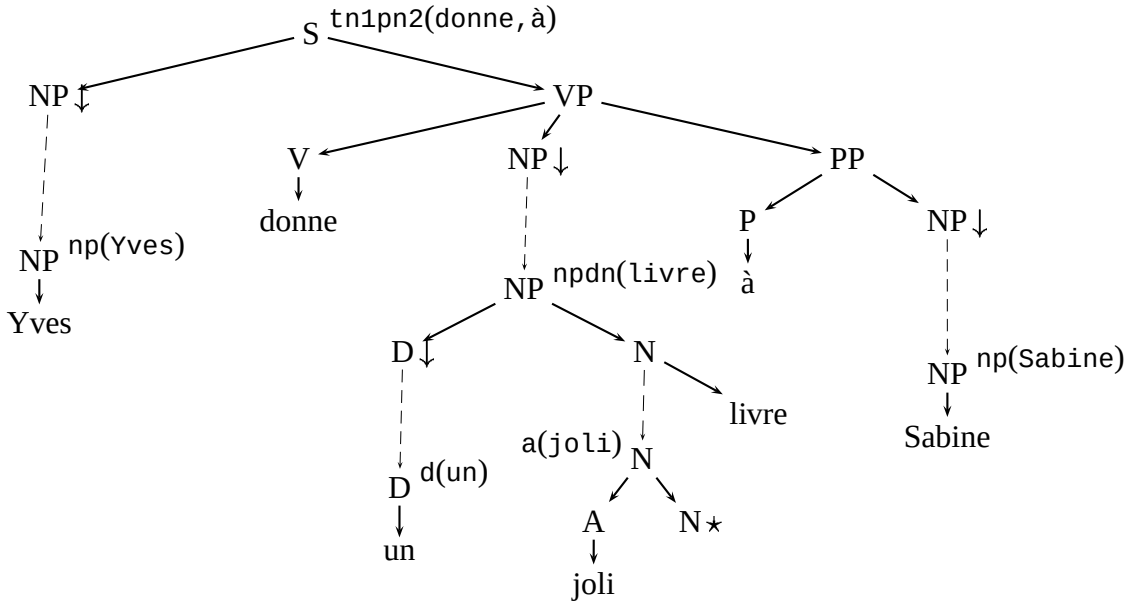
Dans le cas, très fréquent, d'ambigüité, plusieurs ou même une infinité d'arbres de dérivation peuvent être regroupés au sein d'une forêt de taille polynomiale partageant les dérivations, équivalente formellement à une grammaire non contextuelle [Lang, 1991].



(a) Arbre d'analyse



(b) Arbre de dérivation



(c) Arbres élémentaires

FIG. 3: Arbres d'analyse et de dérivation pour « Yves donne un joli livre à Sabine »

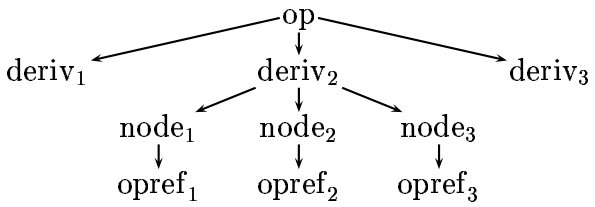


FIG. 4: Structure d'une forêt partagée de dérivation

Cette remarque a guidé notre conception d'une DTD[6] pour représenter les forêts partagées de dérivation. Cette DTD s'appuie sur les éléments principaux `op`, `deriv` et `node` ainsi que sur l'élément `opref` permettant de référencer un élément `op`. L'articulation de ces éléments est donnée par la figure 4 :

`op` est identifié par son attribut `id` et désigne une opération de substitution ou d'adjonction sur une catégorie syntaxique (attribut `cat`) et couvrant une certaine portion de la chaîne d'entrée (attribut `span`). Un élément `op` indique également les valeurs des arguments (éléments `fs`) relatives à l'opération.

`deriv` indique comment réaliser une opération avec un arbre lexicalisé donné par un schéma d'arbres (attribut `tree`) et une ancre (attribut `anchor`).

`node` spécifie quelle opération `op` est effectuée sur un nœud d'un arbre élémentaire nommé par l'attribut `node_id`.

Un arbre de dérivation peut s'exprimer de manière enchâssée en n'utilisant que les éléments principaux `op`, `deriv` et `node`. Une forêt partagée va nécessiter l'emploi de `opref` pour représenter les multiples occurrences d'une même opération. Plus généralement, l'emploi de `opref` permet une description « plate » des forêts.
Ainsi, l'arbre de dérivation précédent s'exprime sous une forme plate par le fragment de XML suivant (sans les structures de traits).

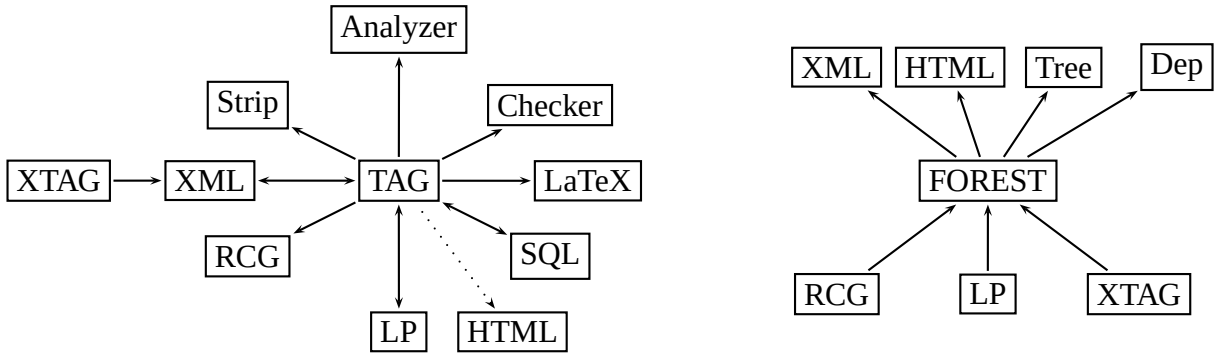
```
<forest parser="Small French TAG - Light DyALog - Hybrid Strategy">
  <sentence> Yves donne un joli livre à Sabine </sentence>
  <op cat="s" span="0 7" id="1" type="subst">
    <deriv tree="tn1pn2" anchor="donne">
      <node id="p_2"><opref ref="5" /></node>
      <node id="np_0"><opref ref="2" /></node>
      <node id="1"><opref ref="4" /></node>
      <node id="np_2"><opref ref="6" /></node>
    </deriv>
  </op>
  <op cat="np" span="0 1" id="2" type="subst">
    <deriv tree="np" anchor="Yves" />
  </op>
  <op cat="np" span="2 5" id="4" type="subst">
    <deriv tree="npdn" anchor="livre">
      <node id="n_"><opref ref="10" /></node>
      <node id="0"><opref ref="8" /></node>
    </deriv>
  </op>
  <op cat="p" span="5 6" id="5" type="subst"> <deriv tree="p" anchor="à" /> </op>
  <op cat="np" span="6 7" id="6" type="subst"> <deriv tree="np" anchor="Sabine" /> </op>
  <op cat="d" span="2 3" id="8" type="subst"> <deriv tree="d" anchor="un" /> </op>
  <op cat="n" span="3 5 4 5" id="10" type="adj"> <deriv tree="an" anchor="joli" /> </op>
</forest>
```

5 Outils de gestion

5.1 Pour les grammaires

La représentation en XML des grammaires est parfaite pour la gestion et l'échange des sources. Cependant, elle ne correspond pas aux formats d'entrée attendus pour la construction des analyseurs (voir section 6). Ainsi, le système DyALog exploite une représentation logique à la PROLOG tandis que le constructeur d'analyseurs RCG nécessite des clauses RCG.

[6] `http://atoll.inria.fr/~clerger/forest.dtd,xml`



(a) Grammaires (b) Forêts

FIG. 5: Modules de gestion

Nous avons donc développé en Perl des modules de conversion complétés ensuite par d'autres modules (figure 5(a)). Le module pivot **TAG** implémente une représentation orientée objets de la structure logique spécifiée par la DTD. Les modules périphériques ajoutent des méthodes supplémentaires pour telle ou telle tâche.
Outre les modules de conversion déjà cités **LP** et **RCG** utilisés pour compiler les grammaires, nous disposons aussi d'un module **XML** de lecture et écriture du format XML. Le module **Checker** vérifie la cohérence des grammaires et produit diverses statistiques. Le module **Analyzer** extrait des informations utiles pour la compilation d'une grammaire. Le module **Strip** élimine les structures de traits d'une grammaire.
Le module **SQL** permet de charger une grammaire dans une base de données SQL (et de restaurer une grammaire à partir de la base). Ce module peut être utile pour vérifier des propriétés d'une grammaire à l'aide de requêtes en SQL[7].
L'utilisation de Perl a été motivée par la disponibilité de nombreux modules logiciels libres sur les sites d'archivage, gérant par exemple l'analyse de XML ou l'accès aux bases de données. Le développement d'un module en Perl est de plus très rapide, en général de l'ordre de quelques heures pour un prototype opérationnel. Nous avons ainsi réalisé un prototype de module **LaTeX**, utile pour construire une forme écrite d'une grammaire. Nous envisageons également un module **HTML** pour produire une version consultable en ligne.

5.2 Pour les forêts de dérivation

De façon similaire, un jeu de modules en Perl permet de traiter les forêts de dérivation (figure 5(b)). Autour du module pivot **FOREST** viennent s'ajouter des modules de conversion. Les modules **LP**, **RCG** et **XTAG** lisent les formats de sortie des forêts de nos analyseurs et de l'analyseur XTAG. Les modules **XML** et **HTML** émettent les forêts sous forme XML et HTML. Deux autres modules **Tree** et **Dep** construisent des spécifications de graphes, visualisables grâce à l'outil **Graphviz**[8].
D'autres modules sont prévus dans un futur proche, tels un module **SQL** de lecture et d'écriture dans une base de données servant de *banque d'arbres*, un module **Strip** d'élimination des traits,

[7] En fait, un prototype de « serveur de grammaire » au dessus d'une base de données MySQL est en cours de finition pour lequel Java a été préféré à Perl. Ce serveur s'appuie néanmoins sur la DTD XML des grammaires.

[8] `http://www.research.att.com/sw/tools/graphviz`

ou des modules pour l'extraction d'arbres dans les forêts.

6 Analyse Syntaxique

6.1 Les analyseurs

Les analyseurs syntaxiques pour les TAG lexicalisées s'appuient généralement sur une phase préliminaire de sélection et d'instantiation d'un sous-ensemble d'arbres ancrables par les mots de la phrase d'entrée. Nous avons choisi de ne pas suivre cette voie. Notre pari est de construire des analyseurs efficaces à partir de la compilation de la grammaire complète et de réaliser l'ancrage des schémas d'arbres lors de l'analyse. Cela permet que notre travail s'applique également à des TAG non lexicalisées (et non lexicalisables). Notons en outre que notre approche n'est pas incompatible à terme avec des techniques de filtrage.
Pour l'instant, nous avons expérimenté plusieurs analyseurs, correspondant à diverses approches et stratégies d'analyse.
La première approche repose sur l'emploi du système DyALog pour compiler des analyseurs tabulaires pour les TAG avec traits. DyALog est un système généraliste permettant l'étude de techniques de tabulation pour des programmes logiques et des formalismes grammaticaux à base d'unification. Nous l'avons étendu pour pouvoir traiter les TAG [Éric Villemonte de la Clergerie and Alonso Pardo, 1998, Alonso Pardo et al., 2000, Éric Villemonte de la Clergerie, 2001]. En particulier, nous avons testé deux schémas de tabulation, ainsi que deux stratégies d'analyse, à savoir une stratégie strictement descendante vérifiant la propriété de validité des préfixes et une stratégie hybride descendante avec vérification ascendante des traits.
La seconde approche utilise les grammaires à concaténation d'intervalles (*Range Concatenation Grammars* – RCG) pour lesquelles nous construisons des analyseurs très efficaces [Boullier, 2000]. Cela nécessite la suppression des informations de traits et de co-ancrage des F-TAG, puis la conversion des TAG obtenues en RCG [Boullier, 1998, 1999, Barthélemy et al., 2001].
La troisième approche [Barthélemy et al., 2000] combine les deux premières. En effet, la forêt partagée de dérivation produite par un analyseur RCG (P_1) est utilisée pour guider un analyseur DyALog P_2 modifié. L'analyseur P_2 vérifie les contraintes de traits et de co-ancrage tandis que les informations fournies par P_1 rendent le traitement extrêmement efficace.

6.2 Le serveur d'analyseurs

Étant donné le nombre croissant d'analyseurs que nous gérons, nous avons cherché à faciliter leur utilisation au moyen d'un serveur d'analyseurs, écrit en Perl. Une fois la connexion établie avec ce serveur, l'utilisateur sélectionne un analyseur et envoie sa phrase ; le serveur retourne alors la forêt de dérivation correspondante. Trois formats de sortie pour les forêts sont actuellement disponibles, à savoir « sans formatage » (la forêt telle que produite par l'analyseur), format XML et format HTML. Ce dernier est en particulier utilisé par une interface WEB accessible en ligne[9].

7 Visualisation des dérivations et des grammaires

En relation avec le serveur d'analyseurs et dans l'optique de comparer les résultats produits par nos analyseurs, nous [Kaouane, 2000] avons modifié et enrichi une interface de visualisation,

[9] `http://medoc.inria.fr/pub-cgi-bin/parser.cgi`

originellement développée par Patrice Lopez [Lopez, 2000] et réalisée en Java. Notre version lit les grammaires et les forêts de dérivation données en XML selon nos DTD. Elle est également capable de se connecter au serveur d'analyseurs, d'envoyer une phrase et de récupérer la forêt de dérivation.

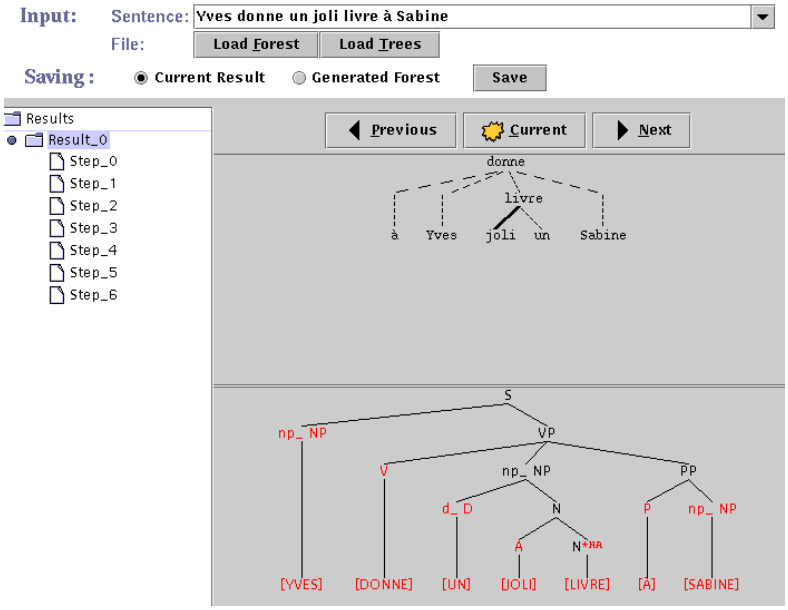


FIG. 6: Capture d'écran de l'outil de visualisation

Les arbres de dérivation sont extraits de la forêt partagée de dérivation et affichés, en regard avec les arbres d'analyse correspondants. De plus, il est également possible d'avancer et de reculer dans les étapes de dérivation, en observant à chaque pas l'arbre partiel de dérivation et l'arbre partiel d'analyse (voir la figure 6).
Outre la visualisation des dérivations, l'interface permet la visualisation des différents composants d'une grammaire (lexiques et schémas d'arbres). Cet outil nous sert à étudier le comportement des analyseurs en fonction des composants d'une grammaire mais il a également un fort potentiel pédagogique car il permet de montrer et d'expliquer les étapes d'une analyse TAG.

Conclusion

Les expériences menées confortent notre opinion que XML est un langage adapté pour représenter et échanger des ressources linguistiques. De plus, l'existence d'outils fondés sur l'emploi de XML a permis le développement rapide de nos outils sur une courte période (autres que les analyseurs syntaxiques, bien entendu).
Nous continuons à compléter progressivement notre atelier ATOLL pour les TAG par de nouveaux modules et outils. En particulier, nous sommes en train de mettre en place un serveur de grammaire permettant l'extraction de fragments de grammaires. D'autres extensions sont également envisagées pour la gestion de banques de forêts de dérivation.
Les outils et ressources présentés ici sont librement disponibles. Notre souhait est de voir d'autres équipes compléter cet atelier pour les TAG par leurs propres outils, tâche facilitée par son architecture modulaire.

Références

Miguel Alonso Pardo, Djamé Seddah, and Éric Villemonte de la Clergerie. Practical aspects in compiling tabular TAG parsers. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 27–32, Université Paris 7, Jussieu, Paris, France, May 2000.

F. Barthélemy, P. Boullier, Ph. Deschamp, and É. de la Clergerie. Guided parsing of range concatenation languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, University of Toulouse, France, July 2001. to be published.

François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric Villemonte de la Clergerie. Shared forests can guide parsing. In *Proceedings of TAPD'00*, September 2000. En ligne en `ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/tapd00.ps.gz`.

Patrick Bonhomme and Patrice Lopez. TagML : XML encoding of resources for lexicalized tree adjoining grammars. In *Proceedings of LREC 2000*, Athens, 2000.

Pierre Boullier. A generalization of mildly context-sensitive formalisms. In *Proceedings of the 4th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 17–20, University of Pennsylvania, Philadelphia, PA, USA, August 1998.

Pierre Boullier. On TAG parsing *et* On multicomponent TAG parsing. In *6ème conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, pages 75–84 et pages 321–326, Cargèse, France, July 1999. Voir aussi le *Rapport de recherche n° 3668*, en ligne en `http://www.inria.fr/RRRT/RR-3668.html`, INRIA, France, Avril 1999, 39 pages.

Pierre Boullier. Range concatenation grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February 2000. Voir aussi le *Rapport de recherche n° 3342*, en ligne en `http://www.inria.fr/RRRT/RR-3342.html`, INRIA, France, January 1998, 41 pages.

Aravind K. Joshi. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia, 1987.

Linda Kaouane. Adaptation et utilisation d'un environnement graphique pour les tag au dessus du système dyalog. Mémoire de DEA, Université d'Orléans, 2000.

Bernard Lang. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current issues in Parsing Technology*, chapter 11. Kluwer Academic Publishers, 1991. Also appeared in the Proceedings of International Workshop on Parsing Technologies – IWPT89.

Patrice Lopez. LTAG workbench : A general framework for LTAG. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, May 2000.

Éric Villemonte de la Clergerie. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, June 2001. À paraitre.

Éric Villemonte de la Clergerie and Miguel Alonso Pardo. A tabular interpretation of a class of 2-stack automata. In *Proceedings of ACL/COLING'98*, August 1998. En ligne en `ftp://ftp.inria.fr/INRIA/Projects/Atoll/Eric.Clergerie/SD2SA.ps.gz`.

The XTAG Research Group. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA, USA, March 1995.