Construire des analyseurs avec DyALog

Éric Villemonte de la Clergerie
ATOLL - INRIA
Domaine de Voluceau
Rocquencourt, B.P. 105,78153 Le Chesnay (France)
Eric.De_La_Clergerie@inria.fr

Mots-clefs – Keywords

Analyse Syntaxique, Tabulation, DCG, TAG, RCG, BMG, TFS
Parsing, Tabulation, DCG, TAG, RCG, BMG, TFS

Résumé - Abstract

Cet article survole les fonctionnalités offertes par le système DyALog pour construire des analyseurs syntaxiques tabulaires. Offrant la richesse d'un environnement de programmation en logique, DyALog facilite l'écriture de grammaires, couvre plusieurs formalismes et permet le paramétrage de stratégies d'analyse.

This paper is a survey of the functionalities provided by system DyALog to build tabular parsers. Providing the expressiveness of logic programming, DyALog eases the development of grammars, covers several linguistic formalisms, and allows the parametrization of parsing strategies.

1 Introduction

Cet article présente les grandes lignes du système DyALog[1]. Issu de travaux sur les techniques de tabulation en programmation en logique, ce système permet la compilation d'analyseurs syntaxiques pour divers formalismes linguistiques à base d'unification. Faute de place, nous ne présentons pas les techniques de tabulation sous-jacentes ni l'architecture et le processus de compilation qui en résultent mais nous nous focalisons sur l'apport de DyALog pour le développement d'analyseurs syntaxiques.

L'écriture de grammaires avec DyALog s'appuie sur la flexibilité des notations à la PROLOG, complétée par diverses extensions plus spécifiquement conçues pour le champ linguistique, telles les structures de traits (Section 2).

Les mécanismes génériques de tabulation de DyALog facilitent la mise au point d'analyseurs

[1] Librement disponible sur le site `http://atoll.inria.fr`.

pour divers formalismes, comme les DCG, les Grammaires à Mouvements Restreints [BMG], les grammaires d'arbres adjoints [TAG] et les grammaires à concaténation d'intervalles [RCG] (Section 3). De plus, des directives de compilation sont disponibles permettant de paramétrer les stratégies d'analyse. Il devient dès lors plus facile de mener des expériences de comparaison entre formalismes et stratégies au sein d'un unique système.

Enfin, nous décrivons, dans la section 4, quelques fonctionnalités qui améliorent l'efficacité ou l'utilisation des analyseurs produits avec DyALog. Nous concluons en relatant quelques expériences menées avec DyALog et quelques évolutions en cours.

2 Faciliter l'écriture de grammaires

Notation Hilog Cette notation permet de décrire des termes d'ordre (pseudo) supérieur, comme par exemple P(X,Y) ou P_(X,Y) qui décrit un terme avec une variable de prédicat P. Ce genre de notation est très utile pour représenter des expressions sémantiques. La notation Hilog est aussi pratique pour associer des groupes distincts d'arguments à un prédicat, par exemple q(X1,X2)(Y1,Y2,Y3). Ainsi, nous l'utilisons pour les RCG (Section 3.5). En interne, le terme Hilog P(X,Y) est représenté par le terme du premier ordre apply(P,X,Y).

Unification immédiate L'opérateur binaire :: force l'unification de ses deux arguments dès la lecture des termes. Nous l'utilisons très souvent pour nommer des structures complexes ayant plusieurs occurrences, comme par exemple dans p(X::g(Z,f(a)),X). L'unification immédiate est également utile comme opérateur de conjonction de formules, par exemple dans les HPSG.

Structures cycliques À la différence de la plupart des systèmes de programmation en logique, DyALog admet les structures cycliques obtenues par exemple par l'unification X::f(X). Ces structures sont rarement nécessaires, sauf potentiellement dans le cas des grammaires HPSG.

Structures de traits, typées ou non Les formalismes linguistiques font grand usage de structures de traits, typées ou non. DyALog offre les deux variantes. Dans les deux cas, la première étape consiste à définir les traits possibles pour un foncteur, comme illustré ci-dessous dans le cas non typé.

```
:-features([np],[gen,num,pers,restr,wh]).
lexicon('Sabine', np{ gen=>fem, num=>sing, restr=>plushum, wh=>(-) }).
```

Les structures de traits typées [TFS] à la Carpenter [Carpenter, 1992] permettent l'héritage multiple. Un type hérite de l'ensemble de traits introduits par ses ancêtres et peut en introduire de nouveaux. Les valeurs des traits sont elles-mêmes typées. Ces informations sont exprimées à l'aide d'une hiérarchie de types. L'exemple suivant illustre les notations sur un fragment de grammaire HPSG. La syntaxe de description des hiérarchies de type est standard, à l'exception de la ligne 2 qui permet de lier le type `string` à un type de base de DyALog, à savoir `symbol`.

```
bot sub [string,list,cat,synsem].
    string escape symbol.
    cat sub [s,np,vp,det,n].
        s sub []. np sub []. vp sub [].
        det sub []. n sub [].
    synsem sub [frase,lexeme] intro [cat:cat].
        frase sub [root] intro [args:list].
            root sub [] intro [cat:s].
        lexeme sub [] intro [orth:string].
    list sub [ne_list,e_list].
        ne_list sub [] intro [hd:bot,tl:list].
        e_list sub [].
```

Le fragment suivant illustre l'utilisation de :: comme opérateur de conjonction, ainsi que l'exploitation des informations de la hiérarchie pour combler les informations manquantes. Ainsi, le trait `orth` introduit de lui-même le type `lexeme`.

```
hpsg(root{} :: args => ne_list{ hd => NP, tl => ( hd => VP :: tl => e_list{} ) })
   --> hpsg( NP :: cat => np{} ), hpsg( VP :: cat => vp{} ).
lexicon( orth => le :: cat => det{}).    %=> lexicon( lexeme{ orth=>le, cat => det{} } ).
```

La notation alternative par chemin A .> tl .> hd permet d'accéder (à la LFG) à la valeur associée au chemin `tl.hd` pour la structure liée à A.

Les structures de traits sont implantées comme des termes standards, un foncteur étant associé à chaque type et un rang fixe étant associé à chaque trait pour un type donné. Pour les types non maximaux (i.e. acceptant des sous-types), un argument supplémentaire (non visible à l'affichage) permet de gérer de manière transparente les cas d'instantiation vers un sous-type (lors de l'unification par exemple).

L'unification de structures typées est souvent considérée comme coûteuse car nécessitant (a) d'identifier les types unifiables ; (b) de trouver les traits devant être unifiés pour deux types distincts ; (c) de construire un nouveau terme quand deux types unifiables τ_1 et τ_2 s'instancient en un sous-type distinct τ_3, nouveau terme qui est perdu en cas d'échec ultérieur de l'unification. Dans le cas de DyALog, les hiérarchies de types sont compilées avec l'utilitaire `tfs2lib` (écrit en DyALog) pour construire une bibliothèque qui étend la syntaxe et l'affichage de DyALog ainsi que les opérations d'unification et de subsomption [Sanches, 1998]. Les points (a) et (b) sont gérés en associant à chaque paire de types distincts et unifiables une fonction spécialisée d'unification (idem pour la subsomption). Le point (3) se gère grâce au mécanisme de partage de structures utilisé par DyALog [Villemonte de la Clergerie, 1993]. Il n'y a pas construction d'un nouveau sous-terme mais réutilisation par partage d'un squelette calculé statiquement. Ces caractéristiques font que l'unification des structures de traits typées n'est guère plus coûteuse que celle des autres termes[2].

Domaines finis Ils permettent de manipuler des disjonctions de valeurs prises dans un ensemble fini qui est déclaré à l'aide des directives **finite_set** ou **subset**. Ils sont implantés dans DyALog à l'aide de vecteurs de bits, rendant leur utilisation très efficace. Ces domaines finis sont d'un usage très fréquent et très pratique pour des grammaires linguistiques, comme illustré par les deux exemples suivants.

Le premier, issu d'une grammaire TAG, définit les valeurs possibles pour les modes du verbe et spécifie les valeurs possibles pour la forme *aime*.

```
:-finite_set(mode, [ind,subj,inf]).
tag_lexicon(aime, '*AIMER*', v, v{ mode => mode[ind,subj], num => sing }).
```

Le second exemple, issu d'un analyseur morphologique jouet pour l'akkadien, développé par François Barthélemy, définit un alphabet comme un domaine fini et caractérise la sous-classe des voyelles.

```
:-finite_set(letter, [a,i,u,e,aleph,b,g,d,w,z,h,'t.',j,k,l,m,n,s,p,'s.',q,r,sh,t]).
:-subset(voyelle, letter[a,i,u,e]).
add_lattice(P1, P2, [K,letter[j,aleph], voyelle[] ]).
```

[2]Cependant d'autres problèmes ne sont pas complètement réglés, en particulier les problèmes d'indexation efficace des TFS.

3 Proposer une palette de formalismes

Nous présentons plusieurs formalismes, linguistiques ou non, couverts par DyALog. Cette diversité permet la comparaison d'analyseurs pour divers formalismes au sein d'un même système. Du point de vue du développement de DyALog, cette diversité permet également de tester plus largement nos techniques tabulaires et de généraliser certaines optimisations proposées dans la littérature.

3.1 Programmation en logique

Historiquement, DyALog est issu de la volonté d'exploiter les techniques de tabulation en programmation en logique. En conséquence, DyALog offre actuellement la puissance d'un environnement de programmation en logique, permettant, en particulier, d'auto-amorcer (*bootstrap*) son compilateur qui est écrit en DyALog.

Le fait de pouvoir utiliser la puissance de la programmation en logique présente au moins deux avantages. Premièrement, il est immédiat de réaliser des échappements vers des prédicats logiques dans les grammaires linguistiques. Cela permet de gérer plus facilement certains détails d'un formalisme, comme par exemple la gestion des contraintes de co-ancrage dans les grammaires XTAG [Doran et al., 1994]. Deuxièmement, que le compilateur DyALog soit écrit en DyALog illustre une caractéristique bien connue de la programmation en logique, à savoir la facilité d'écriture de méta-interprètes. En pratique, cela signifie qu'il est facile d'étendre le compilateur de DyALog pour intégrer de nouveaux formalismes ou alternativement de construire rapidement un méta-analyseur.

Par défaut, les prédicats logiques sont tabulés. Ce comportement peut être modifié par des directives de compilation, permettant de ne pas tabuler certains prédicats et d'optimiser leur traitement. Nous avons ainsi des optimisations spécialisées pour les prédicats uniquement définis par des faits ou les prédicats dont tous les descendants ne sont pas tabulés. Cette classification des prédicats s'étend dans une certaine mesure aux non-terminaux des formalismes grammaticaux.

La plupart des prédicats standard de PROLOG sont disponibles sous DyALog et l'appel de fonctions C est possible avec `$interface`. Dans le cadre d'une interface vers les bases de données, nous définissons ainsi le prédicat pg_tuple qui retourne, de manière non-déterministe, les valeurs d'une requête SQL.

```
pg_tuple(Res,Val) :- '$interface'( 'DyALog_PQtuple'(Res:ptr,Val:term), choice_size(1)] ).
```

3.2 DCG

Comme la plupart des systèmes de programmation en logique, les *Grammaires de Clauses Définies* [DCG] [Pereira and Warren, 1980] sont disponibles sous DyALog. Cependant, l'intégration de la tabulation rend plus immédiate l'utilisation des DCG en évitant les problèmes de bouclage et en offrant une trace des analyses au travers des forêts partagées d'analyse[3]. Nous avons également étendu les DCG. Ainsi, l'opérateur & d'intersection permet, par exemple, de définir très facilement le langage $a^n b^n c^n$ avec la clause DCG «s −−> 'AnBnC∗' & 'A∗BnCn'».

L'analyse peut se faire sur le contenu d'une liste PROLOG de terminaux, comme cela est standard pour les DCG, mais aussi, alternativement, sur un treilli de mots ou même sur un automate

[3] ou plus précisement de forêts de dérivation isomorphes aux forêts d'analyse.

à états finis [FSA]. Le passage aux FSA rend l'analyse plus efficace et offre une grande flexibilité pour traiter des phrases incomplètes ou à mots ambigus ou inconnus, comme illustré par la phrase «*[mot inconnu] regarde [mots inconnus] avec un télescope*.» codée en FSA par la base suivante de faits 'C'(Left,Token,Right) [4] :

```
'C'(0,_,1). 'C'(1,regarde,2). 'C'(2,_,2). 'C'(2,avec,3). 'C'(3,un,4). 'C'(4,télescope,5).
```

Annoter les clauses avec les opérateurs +> et <+ permet de définir des stratégies d'analyse bidirectionnelle, généralement dirigées par les têtes. Ainsi, la clause « a −−> b <+ **c** +> d +>e. » donne l'ordre de reconnaissance c, d, e, b. Cette notation étant plus opérationnelle que déclarative, il est préférable de convertir une grammaire avec des indications de tête en une grammaire avec des annotations de direction, comme réalisé pour une grammaire DCG du portugais [Rocio et al., 2001][5] :

```
s −−>np,vp.                          head(s,vp).  %==>s −−>np <+ vp
vp −−>v(Type), v_args(Type).         head(vp,v).  %==>vp −−>v(Type) +> v_args(Type)
```

Parallèlement à l'écriture des grammaires, DyALog offre la possibilité de spécifier la stratégie d'analyse à utiliser au niveau des non-terminaux en utilisant des directives de modulation [Barthélemy and Villemonte de la Clergerie, 1998]. Ainsi la directive dcg_mode(np/2,+(−,−),+,−) exprime le fait que, pour reconnaître un non-terminal np(sing,fem) entre les positions gauche L et droite R de la chaîne d'entrée, seuls np et L sont utilisés pour la prédiction (phase descendante d'appel), les autres informations étant vérifiées lors de la propagation des réponses (phase ascendante de retour). Il est ainsi immédiat de basculer d'une stratégie totalement descendante gauche-droite avec la directive de modulation «:−dcg_mode(_,+,+,−)» à une stratégie totalement ascendante avec la directive «:−dcg_mode(_,−,−,−)».

3.3 BMG

Les grammaires à mouvements restreints [BMG] sont une variante de grammaires d'extraposition qui se codent comme des extensions des DCG. Elles ont été ajoutées à DyALog dans le cadre du développement d'une grammaire du portugais [Rocio et al., 2001] et héritent des propriétés des DCG, en terme de modulation, de bidirectionalité et de lecture des terminaux. Le principe de ces grammaires est que des constituants peuvent être empilés temporairement et être déchargés ultérieurement pour combler des trous dans la phrase. Des barrières (*island*) permettent de bloquer le déplacement de constituants, si nécessaire. Comme illustré par le fragment suivant (pour le portugais), les directives bmg_stacks, bmg_pushable et bmg_island définissent les piles (ici pour les relatives, les interrogatives et les topicalisations), les constituants autorisés sur chaque pile et les opérateurs de barrières (en plus des opérateurs automatiquement définis pour chaque pile). La clause ligne 8 permet ainsi d'empiler sur **slash** un groupe prépositionnel pp topicalisé, celui-ci pouvant être déchargé dans le groupe verbal (ligne 9) mais pas dans le groupe nominal sujet (ligne 6) à cause de la barrière **isl_slash**.

```
:−bmg_stacks([slash,rel,quest]).
:−bmg_pushable(np,[quest,rel]).
:−bmg_pushable([v,pp],[slash]).
:−bmg_island(isl_relquest,[rel,quest]).

s −−> isl_slash np, vp.
s −−>comp, s.
comp slash pp −−> isl pp.
vp −−>v,np,pp.
```

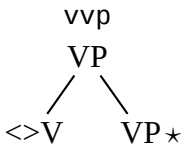
[4] Il est possible de spécifier un prédicat de lecture autre que **C**/3 à l'aide de la directive scanner.

[5] Ce mécanisme sera systématisé à terme dans le compilateur de DyALog.

3.4 TAG

DyALog permet le traitement des grammaires d'arbres adjoints [TAG] [Joshi, 1987], avec décoration possible des non-terminaux par des arguments `top` et `bottom` (Feature TAG). De plus, il est possible (mais non obligatoire) d'organiser les grammaires selon une architecture à la XTAG [Doran et al., 1994] où (a) les arbres élémentaires sont ancrables (par des terminaux) et regroupés par familles et (b) le lexique est hiérarchisé en formes fléchies référençant des lemmes qui indiquent quels arbres ils ancrent.

Ainsi l'exemple suivant, issu d'une TAG jouet pour le français, montre l'arbre auxiliaire `vvp` de la famille éponyme `vvp` ancrant les verbes de modalité comme *pouvoir* ainsi que les entrées associées à la forme *peut* et au lemme *pouvoir*.




```
tag_tree{ name => vvp, family => vvp,
          tree=> auxtree bot=VP::vp{}
                 at vp( <> v, id=vp_ and bot=VP at *vp) }.
tag_lemma('*POUVOIR*',v,
          tag_anchor{ name=>vvp, equations=>[bot = vp{ mode=>inf } at vp_]}).
tag_lexicon(peut, '*POUVOIR*', v, v{ mode => ind, num => sing }).
```

Les TAG peuvent être écrites directement en DyALog, mais il existe également un format de représentation XML pour ces grammaires et des outils de conversion vers le format d'entrée de DyALog [Barthélemy et al., 2001]. Comme pour les DCG et les BMG, les stratégies d'analyse pour les TAG sont modulables. En interne, l'analyse tabulaire des TAG s'appuie sur l'utilisation d'automates à 2 piles [Villemonte de la Clergerie, 2001]. Notons qu'en sus de l'extension du compilateur, nous nous sommes aussi amusés à réaliser un méta-interprète pour les TAGs.

3.5 RCG

Les grammaires à concaténation d'intervalles [RCG] [Boullier, 2000] forment une classe très puissante de grammaires pouvant néanmoins s'analyser en temps polynomial. De nombreux formalismes, comme les TAG, peuvent être encodés à l'aide des RCG. Les clauses RCG ressemblent à des clauses DCG, à la différence que les arguments spécifient les intervalles de la chaîne d'entrée couverts par les non-terminaux. Ces arguments sont des terminaux ou des variables (X, Y, ...) séparés par l'opérateur de concaténation **@**. En plus de ces arguments, l'implantation DyALog des RCG permet d'associer un second jeu d'arguments logiques aux non-terminaux (XRCG). Ainsi, la grammaire suivante reconnait le langage $a^n b^n c^n$, utilisant un argument logique attaché aux non-terminaux s et a pour retourner n.

```
s(N) (X@Y@Z) --> a(N) (X,Y,Z).                    a(0) ("","","") --> true.
a(M) ("a"@X,"b"@Y,"c"@Z) --> a(N) (X,Y,Z), {M is N+1}.  axiom(s(N)).
```

L'intégration des RCG dans le compilateur DyALog reste préliminaire mais s'est effectuée en moins de 2 jours, en commençant par réaliser un méta-analyseur, puis une extension du compilateur. Nous avons testé des RCG résultant de la conversion (automatique) de TAG (50 et 420 arbres) et obtenu de bon résultats [6].

[6] Sans rivaliser néanmoins avec l'implantation très performante de Pierre Boullier.

3.6 Autres formalismes logiques

Pour l'instant, DyALog n'intègre pas les formalismes LFG et HPSG mais propose déjà certaines fonctionnalités (programmation en logique et structures de traits typées) qui doivent faciliter cette intégration.

4 Améliorer les parseurs

Forêts partagées Les analyseurs construits avec DyALog admettent l'option `-forest` permettant de visualiser l'ensemble des arbres de dérivation produits par une analyse. Cet ensemble d'arbres est émis sous une forme partagée, reflétant le partage de calcul qui résulte de l'emploi des techniques de tabulation. Dans le cas des DCG, la forêt de dérivation est isomorphe à la forêt d'analyse mais ce n'est plus le cas pour les TAG. Le format de représentation utilisé pour les forêts correspond à une vue «grammaire» et illustre le fait que ces forêts ont formellement la structure de grammaires hors-contexte avec comme non-terminaux les constituants de la phrase et comme productions les règles de dérivation des constituants. Dans le cas des TAG, les non-terminaux dans le corps des productions sont éventuellement précédés par un label indiquant le nom du noeud sur lequel s'attache le constituant (par substitution ou adjonction).

Ainsi, l'exemple suivant provient de l'analyse de la phrase «*Yves aime Sabine*» par une TAG, avec l'indice 1 représentant le constituant s{inv=> −, mode=> mode[ind, subj]}(0,3) construit avec l'arbre `tn1` par ancrage de la forme `aime` sur le noeud <> et de substitutions sur les noeuds np_0 et np_1 des constituants associés aux indices 2 et 4.

```
s{inv=> −, mode=> mode[ind, subj]}(0,3)  1 <−− [np_0]2 [<>]3 [np_1]4
np{gen=> masc, num=> sing }(0,1)         2 <−− [<>]5
tag_anchor(aime,1,2,tn1)                 3 <−−
np{gen=> fem, num=> sing }(2,3)          4 <−− [<>]6
tag_anchor(Yves,0,1,np)                  5 <−−
tag_anchor(Sabine,2,3,np)                6 <−−
```

Un constituant peut avoir plusieurs dérivations possibles, chacune étant associée à une productions, comme par exemple dans le cas de «*Yves cherche les fleurs sur la table*», avec ambiguïté de l'attachement prépositionnel :

```
s{}(0,7)      1 <−− ( [np_0]2 [<>]3 [np_1]4 [vp]5 | [np_0]2 [<>]3 [np_1]6 )
```

Dans le cas des TAG, des outils de conversion existent pour transformer la vue grammaire en une représentation XML, qui sert ensuite de pivot pour des représentations graphiques sous forme d'arborescences ou de graphes de dépendance [Barthélemy et al., 2001][7].

Le fait de pouvoir annoter un non-terminal par un label (tel np_0) dans le corps des productions aide à la lecture des forêts. Le principe s'étend à d'autres formalismes que les TAG en définissant un opérateur d'étiquetage permettant de nommer les non-terminaux dans les clauses, comme dans le cas de la clause DCG suivante :

```
:− tagop(':').  s −−> sujet:np, verbe:v, objet:np.
```

L'extraction des forêts s'effectue grâce aux pointeurs arrières vers les parents qui sont présents dans les objets tabulés lors de l'analyse. Des prédicats permettent l'examen des objets tabulés

[7] Le serveur de parseurs accessible à `http://medoc.inria.fr/pub-cgi-bin/parser.cgi` permet de comparer les différentes vues des forêts.

et de leurs pointeurs arrières, rendant possible le développement d'algorithmes de traitement de forêts (recherche des points d'ambiguïté, calculs de formes sémantiques).

Lexicalisation DyALog ne fait aucune hypothèse sur le statut lexicalisé ou non des grammaires. Cependant, des extensions récentes améliorent le traitement des structures (clauses ou arbres) lexicalisées (ou lexicalisables) en pouvant leur associer une condition d'activation qui est vérifiée au début de l'analyse. Ainsi, l'exemple suivant permet de bloquer l'activation de la clause si le lexical «*qui*» n'est pas présent dans la chaîne d'entrée.

```
'$loader'( phrase([qui],_,_), ( np-->np, [qui], s_rel )). %%'$loader'(Cond,Clause).
```

Bien qu'il soit préférable que le compilateur déduise de lui-même ces conditions d'activation (comme c'est déjà le cas pour les TAG), ce méchanisme permet à un utilisateur de tester ou d'affiner ses propres conditions d'activations.

Robustesse Convertir un analyseur complet en un analyseur partiel robuste revient simplement à changer la requête initiale :

```
?-recorded('N'(N)), A=0, tag_phrase(top=S::s{} at s,A,N).                    %analyse complète
?-tag_phrase(top=U1::s{} at s,A,N) ; tag_phrase(top=U2::np{} at np,A,N). %analyse partielle
```

Nous avons déjà mentionné la possibilité de définir des stratégies d'analyse bidirectionnelle, lesquelles sont bien adaptées dans le cas d'analyseurs partiels.

L'utilisation d'automates finis comme entrée pour l'analyse donne également une robustesse accrue en permettant le traitement de phrases avec mots ambiguës, mots inconnus, voire portions inconnues de phrases.

Enfin, les techniques de tabulation couplées aux propriétés d'examen de la table à l'aide de prédicats logiques permettent la mise au point d'algorithmes de correction des erreurs. Cette possibilité est en cours d'étude dans le cadre d'un analyseur robuste du portugais.

5 Expériences et évaluations

Nous utilisons DyALog en interne pour diverses grammaires DCG et surtout TAG. En particulier, nous menons des expériences sur une TAG jouet pour le français de 50 arbres et une de l'anglais de 400 arbres et prévoyons de passer très prochainement aux grammaires du français et de l'anglais à large couverture (plusieurs milliers d'arbres). Nous avons testé la robustesse (analyse partielle et traitement de mots inconnus) ainsi que l'intérêt des techniques de filtrage résultant de la lexicalisation (Tables 1 et 2)[8].

Par ailleurs, DyALog est utilisé pour traiter les niveaux 2 des 3 niveaux d'un analyseur robuste multi-niveaux du portugais[Rocio et al., 2001], à savoir (a) l'identification de syntagmes non-récursifs avec une grammaire DCG et une stratégie bidirectionnelle dirigée par les têtes ; et (b) le rattachement des syntagmes avec une grammaire BMG. Les résultats obtenus par rapport à un analyseur à charte standard sont parlants, à savoir 254 mots par seconde pour DyALog contre 1,66 pour l'analyseur standard sur un PC 200MHz. Enfin, très récemment, l'ensemble des niveaux plus une phase complémentaire de correction d'erreurs a été réécrit avec DyALog.

[8] La plupart des analyseurs résultants de ces expérimentations sont accessibles en ligne sur `http://medoc.inria.fr/pub-cgi-bin/parser.cgi`.

Phrase	Filtrage faible	Filtrage fort	Robuste
∅	169	89	86
Jean aime Marie	230	90	98
Yves montre des fleurs à Sharon	260	130	129
Kathy regarde un livre avec Stéphane	250	129	115
un ami américain de Yves qui visite la France boit du bon vin	522	213	215
la jolie fille pense que Andrew est le grand ami de son chien	336	179	180
Marie comprend que Kathy dort bien chez Andrew	306	122	127
Anne voit les fleurs que Yves cherche sur la table	362	164	173
Jean pense que Béatrice perd les pédales	297	133	133
le petit chat de Sabine court vite	270	135	137
la fille de Marie aime bien les fleurs	279	127	122
qui dit que Yves a un chien	322	128	127
un enfant intelligent de Marie comprend lentement que sa mère est bête	378	225	225

TAB. 1: Temps d'analyse avec une petite TAG du français (en ms sur un PC 450MHz)

Phrase	Filtrage faible	Filtrage Fort	Robuste
∅	506	119	110
John loves Mary	13190	442	378
John eats	8031	391	344
John thinks that he is eating	43340	3173	2875
John is not advising her not to eat the apple	64750	5010	5311
John advises Mary that she should eat an apple	70500	5025	4214
the book belongs to John	11730	459	551
the apple falls apart	11450	382	479
John gives Mary an apple	13690	754	708
John eats while giving Mary an apple	15300	1697	1394
perhaps John is eating an apple	20280	903	930
nevertheless John is not eating an apple	22880	1164	1235
John was also eating an apple	19620	1445	1569

TAB. 2: Temps d'analyse avec une petite TAG de l'anglais (en ms sur un PC 450MHz)

6 Évolutions

Le système DyALog nous sert de plate-forme pour tester de nouvelles idées en matière d'analyse syntaxique, et notre catalogue d'évolutions possibles reste fourni. Ainsi, nous comptons enrichir la palette des stratégies d'analyse possibles en implantant efficacement des stratégies comme celles par coin gauche (*left corner*) qui reposent sur des tables de décision statiquement calculées. Nous prévoyons également de modifier (profondément) le modèle de compilation des grammaires pour permettre la factorisation des préfixes communs dans les parcours des clauses ou des arbres TAG. Pour les TAG, nous souhaitons optimiser le traitement des cas d'adjonctions non enveloppantes.

Enfin, de récents résultats théoriques sur la tabulation vont permettre d'uniformiser le traitement des formalismes actuels (en particulier les RCG) et de traiter de nouveaux formalismes comme les Multi-Component TAG[Villemonte de la Clergerie, 2002]. Nous allons également poursuivre l'intégration des formalismes à base de structures de traits tels HPSG ou LFG.

7 Conclusion

Nous avons présenté les grandes lignes du système DyALog qui systématise l'emploi de techniques de tabulation, se veut multi-formalismes, multi-stratégies, et est orienté vers l'écriture de grammaires. Nous pensons que DyALog peut jouer un rôle à plusieurs titres. Dans un cadre pédagogique, l'intégration de base des techniques de tabulation permet de développer et tester

rapidement des grammaires jouets pour divers formalismes. Pour les concepteurs de nouveaux formalismes, la facilité d'extension du compilateur ou d'écriture de méta-analyseurs permet de tester ceux-ci au sein d'une architecture uniforme. Pour les concepteurs d'analyseurs syntaxiques, DyALog sert de plate-forme d'intégration pour tester de nouvelles optimisations et les généraliser à un ensemble de formalismes. Enfin, les analyseurs produits avec DyALog étant relativement efficaces, on peut envisager leur utilisation dans le cadre d'applications réelles.

Références

François Barthélemy, Pierre Boullier, Philippe Deschamp, Linda Kaouane, and Éric Villemonte de la Clergerie. Atelier ATOLL pour les grammaires d'arbres adjoints. In *Proceedings of TALN'01*, pages 63–72, Tours, France, July 2001.

François Barthélemy and Éric Villemonte de la Clergerie. Information flow in tabular interpretations for generalized push-down automata. *Theoretical Computer Science*, 199 :167–198, 1998.

Pierre Boullier. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February 2000.

Bob Carpenter. *The Logic of Typed Feature Structures with Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Number ISBN 0-521-41932. Cambridge University Press, 1992.

Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. XTAG system — a wide coverage grammar for English. In *Proc. of the 15th International Conference on Computational Linguistics (COLING'94)*, pages 922–928, Kyoto, Japan, August 1994.

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.

Fernando C. N. Pereira and David H. D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13 :231–278, 1980.

Vitor Jorge Rocio, Gabriel Pereira Lopes, and Éric Villemonte de la Clergerie. Tabulation for multi-purpose parsing. *Grammars*, 4(1) :41–65, 2001.

Fernand Sanches. Étude et implantation modulaire d'algorithmes d'analyse syntaxique pour des grammaires utilisées en langue naturelle (grammaires d'arbres adjoints ou grammaires lexicales fonctionnelles). Mémoire d'Ingénieur CNAM, 1998.

Éric Villemonte de la Clergerie. Layer sharing : an improved structure–sharing framework. In *Proc. of POPL'93*, pages 345–356, 1993.

Éric Villemonte de la Clergerie. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, pages 167–174, CMU, Pittsburgh, PA, USA, June 2001.

Éric Villemonte de la Clergerie. Parsing MCS languages with thread automata. In *Proc. of TAG+6*, May 2002.