TALN 2011, Montpellier, 27 juin – 1er juillet 2011

<TextCoop>: un analyseur de discours basé sur les grammaires logiques

Patrick Saint-Dizier
IRIT-CNRS, Toulouse
stdizier@irit.fr

Résumé. Dans ce document, nous présentons les principales caractéristiques de <TextCoop>, un environnement basé sur les grammaires logiques dédié à l'analyse de structures discursives. Nous étudions en particulier le langage DisLog qui fixe la structure des règles et des spécifications qui les accompagnent. Nous présentons la structure du moteur de <TextCoop> en indiquant au fur et à mesure du texte l'état du travail, les performances et les orientations en particulier en matière d'environnement, d'aide à l'écriture de règles et de développement applicatif.

Abstract. In this paper, we introduce the main features of <TextCoop>, an environment dedicated to discourse analysis within a logic-based grammar framework. We focus on the structure of discourse rules (DisLog language) and on the features of the engine, while outlining the results, the performances and the orientations for future work.

Mots-clés : grammaire du discours, programmation en logique, grammaires logiques.

Keywords: discourse structure, logic programming, logic-based grammars.

1 Analyser quelles structures discursives ?

Lorsque l'on pense à l'analyse de structures discursives, il vient d'abord à l'esprit l'analyse des structures rhétoriques qui, d'une façon ou d'une autre, sont censées permettre de rendre compte de façon complète des diverses articulations discursives d'un texte (Marcu 97, 02). L'objectif est de relier tous les éléments d'un texte par le biais de ces relations, ce qui rend alors compte de la structure sémantico-pragmatique de ce texte. Outre le fait que ces relations existent en grand nombre et avec parfois des définitions un peu vagues et difficilement opérationalisables, il existe en fait, pour le besoin des applications, un grand nombre d'autres structures qui rentrent plus ou moins facilement dans le paradigme rhétorique.

C'est ainsi le cas des cadres du discours, initié en France par M. Charolles, pour lesquels les relations rhétoriques 'frame' ou 'background' ne sont pas tout à fait satisfaisantes. C'est aussi le cas de nombreux types de structures 'dédiées', comme par exemple les instructions dans le discours procédural. Enfin, notons toutes les structures qui relèvent de la typographie et qui ont un lien avec le contenu du texte (titres, notes, paragraphes, listes, etc.). Enfin, notons la complexité sous-jacente de certaines représentations qui forment des réseaux complexes de liens entre structures.

Dans la suite de ce document, nous proposons un environnement, <TextCoop>, dédié à l'analyse des structures discursives, basé sur la notion de grammaire logique. Nos expérimentations ayant largement tourné autour de l'analyse des diverses structures rencontrées dans les textes procéduraux, nombre d'exemples sont empruntés à ce cadre (Delpech et al 07, 08) (Aouladomar et al. 05), voir aussi (Delin 94) ou (Takechi 03). <TextCoop> désigne l'ensemble de l'architecture du système, y compris les outils d'aide à la mise au point et les ressouces linguistiques associées. DisLog (pour 'Dicourse in Logic' ou 'Discontinuities in Logic') désigne le langage qui décrit les règles d'analyse et les contraintes que l'on peut y associer.

Notre modélisation n'est pas dédiée à un cadre applicatif particulier ou à un genre textuel. Après un bref positionnement, nous présentons la syntaxe des règles de DisLog ainsi que des outils associés. Contrairement à une approche basée sur l'apprentissage (Marcu 02), l'ensemble de notre travail est positionné dans une modélisation linguistique et déclarative, typique des grammaires logiques, qui autorise le raisonnement. Notre approche est quelque peu basé sur une vision générative à base de principes. Nous présentons ensuite les fonctionalités du

moteur ainsi que son environnement actuel. Le développement de <TextCoop> est encore dans un stade expérimental : un travail est toujours en cours sur les propriétés de son environnement et des fonctions qu'il peut offir. Par contre, ses fondements sont globalement fixés, et ont été testés dans plusieurs cadres linguistiques et applicatifs.

Historiquement, <TextCoop> a initialement fait l'objet d'une étude dédiée aux procédures grand-public (Delpech et al 08) avec une implémentation simple en Perl de l'ensemble des fonctions. Toutefois, la rigidité, le peu de portabilité et les limites expressives de Perl nous ont poussé à refaire une implémentation en Java, à base de générateurs d'automates, utilisant JCUP. Cette approche a dû être abondonnée après 10 mois de programmation infructueuse. Voulant augmenter les possibilités expressives du système, en particulier au niveau raisonnement, et avoir un développement fiable et rapide, nous avons finalement opté pour une version en Prolog que nous pouvons facilement faire évoluer et maintenir. Via une collaboration avec une société, les aspects interfaces et aide à la mise au point seront développés dès que pertinent pour en faire une plateforme opérationelle. Une licence de type GPL est prévue au moins pour la partie noyau.

1.1 Le positionnement de <TextCoop>

La plateforme <TextCoop>, dédiée à l'analyse de structures du discours, doit permettre de pouvoir reconnaître une grande diversité de structures, génériques ou dédiées à des applications ou à des genres textuels. <TextCoop> vise à la fois le traitement de structures discursives génériques, dans des textes quelconques, et le traitement de structures plus spécifiques, 'métier', dans des textes plus spécialisés.

Considérant la complexité de la description des structures du discours, nous développons une vision qui s'appuie sur quelques considérations simples de la grammaire générative à savoir développer :
– des principes productifs, qui ont un bon niveau d'abstraction, linguistiquement adéquat, mais qui sur-reconnaissent dans certains cas,
– et des principes restrictifs qui viennent limiter la puissance des premiers, sur la base de contraintes de bonne formation, qui peuvent être généraux ou spécifiques.
Cette approche modulaire permet une meilleure modélisation des phénomènes, plus compartimentée, et un meilleur contrôle sur le résultat. Elle permet aussi une mise au point des règles et une évolutivité plus simple. Ces principes sont gérés par un traitement en cascade des règles, y compris de liage et de correction.

Par le biais des différentes contraintes introduites dans DisLog, il est possible de produire des représentations étiquetées complexes, sous forme d'arbres, de graphe ou de dépendances. DisLog permet d'introduire des relations de un vers plusieurs ou de plusieurs vers un, permettant ainsi qu'une structure soit en relation avec plusieurs autres structures de natures différentes. Cependant, dans la plupart des textes étudiés, ces relations sont relativement simples, le souci étant en général de préserver l'intelligibilité des documents.

Le formalisme des règles, DisLog, permet d'introduire tout type de forme de raisonnement a priori. Ceci est un point original et crucial en analyse du discours, facilité par l'implémentation réalisée en Prolog. Ces formes de raisonnements permettent entre autres (1) de réaliser des calculs, reportés dans les annotations produites, (2) de lever des ambiguïtés d'analyse, (3) de compléter l'analyse grammaticale par l'appel par exemple à des connaissances (pour inclure des données pragmatiques). Si une requête de raisonnement échoue, alors la règle échoue.

La littérature est particulièrement abondante s'agissant de l'analyse du discours. On y trouve plusieurs directions. Un mouvement théorique assez important s'est développé depuis 20 ans environ, autour de plusieurs cadres dont la DRT et ses extensions. Ce cadre demeure essentiellement abstrait et orienté vers des modes de représentations peu expressifs. Notre orientation étant à la fois plus empirique et liée à une sémantique conceptuelle plutôt que formelle, ce cadre n'est a priori pas pertinent pour nos travaux et ne sera pas évoqué ici. Plusieurs approches empiriques sont par contre d'un intérêt marqué. Il y a tout d'abord les travaux qui caractérisent la nature et la forme des relations rhétoriques. (Mann et al 88) ont proposé une formulation contemporaine de ce cadre. De nombreux travaux ont suivi dont (Delin 94), (Kosseim et al 00), (Rossner et al 92), (Saito et al 06), (Vander Linden 93), etc. qui affinent ces relations ou les étudient dans des cadres spécialisés. cependant, on assiste alors à une prolifération de ces relations, où les définitions deviennent parfois vagues.

Un courant plus profond se préoccupe du sens véhiculé par ces relations, dans une perspective cognitive, de leur aspects pragmatiques ainsi que des intentions de communication sous-jacentes (Wright 04), (Moschler 85) (Davidson 63) (Anscrombre et al 81). Ceci est particulièrement intéressant dans différents cadres tels que le dialogue, l'argumentation et la négociation (Amgoud et al. 01, 05), et la génération de langue naturelle (Rosner et

al. 92) qui, au niveau de son composant de planification, s'appuie en particulier sur des schémas rhétoriques.

Une interrogation toujours d'actualité concerne la caractérisation en langue des relations de discours afin de pouvoir les identifier automatiquement. C'est de toute évidence un défi ouvert, où des solutions parfois parallèles ont été tentées, car ces relations n'ont pas systématiquement des marques qui les identifient. Notons par exemple (Mann et al. 88), (Saito et al. 06), (Takechi et al 03) (Di Eugenio et al 96) qui soulignent bien les résultats que l'on peut escompter. Récemment, par exemple via le projet ANR Annodis, une approche à base d'apprentissage à partir d'annotations manuelles s'est développée. Une telle entreprise se heurte à deux difficultés : les désaccords importants (mais inévitables) entre annotateurs et la difficulté de développer de l'apprentissage sur des segments textuels importants où peu d'information est en fait pertinente. Enfin, notons les travaux qui développement des grammaires pour le texte par exemple à partir de TAGs (Gardent 97) (Webber 04).

Au niveau des environnements, GATE (http ://gate.ac.uk/) est une plateforme très répandue et qui intervient dans de nombreux projets. Elle est essentiellement dédiée à l'analyse de phrases ou de courts fragments de textes. Par ailleurs, Linguastream
(http ://www.linguastream.org/home.html) est une plateforme ouverte pour l'analyse du langage qui peut accepter en entrée tout type de texte XML. Il est basé sur une architecture en composants et offre plusieurs API Java utiles pour l'intégration. C'est un système largement ouvert qui laisse une grande liberté à l'utilisateur tout en lui proposant un ensemble d'outils d'aide et d'interfaces très pertinents. Linguastream est d'abord dédié à l'analyse de la phrase au sein de textes. Cette plateforme ne permet pas d'inclure de modules de raisonnement comme cela est très utile en analyse de structures discursives, par exemple pour lever des ambiguités ou pour introduire des considérations pragmatiques. Il serait toutefois intéressant de voir avec ces plateformes comment on peut écrire des analyseurs de structures de discours.

Au niveau des grammaires de discours, nous pensons qu'il est nécessaire de préserver une analyse linguistique précise, qui permet de décrire les phénomènes à un bon niveau d'abstraction, en préservant une certaine prédictibilité. Nous nous attacherons donc à un travail essentiellement manuel, même si des traitements automatiques sur corpus sont utilisés pour explorer les constructions (par exemple par bootstrapping). Ce texte étant dédié à la partie grammaire, cet aspect méthodologique est traité ailleurs.

1.2 Le langage des structures du discours

Il n'est pas dans notre objectif d'argumenter pour les différents aspects un langage élargi qui rende compte de l'ensemble des formes que peuvent prendre les structures qui relèvent du discours. Nous nous contenterons d'en observer un certain nombre, fortement récurrentes dans les situations que nous avons examinées, et qui sont à la base du langage de description DisLog introduit dans <TextCoop>. Ce langage est conçu de façon assez ouverte pour pouvoir permettre de coder de nombreuses configurations.

Les relations rhétoriques (Mann et al 88) sont structurées sous forme de deux types de relations :
– une relation hiérarchique dite de noyau vers satellite. Ainsi dans *insérez verticalement la carte mère sinon vous risquez d'endommager les connecteurs'*, la première partie de l'énoncé est habituellement appelée conclusion d'argument (ici de type avertissement) tandis que la seconde partie, qui explique les risques encourus, est appelée support de l'argument. La conclusion peut apparaitre seule, avec un support vide, mais le support n'a de sens que s'il est relié à au moins une conclusion.
– une relation non hiérarchique de noyau vers noyau. Ainsi la relation 'parallèle' associe-t-elle deux structures de même niveau, comme dans l'ellipse : *Jean est reçu à son permis, Marie aussi*.

L'analyse discursive des textes s'applique souvent sur ces deux types de schémas. Cette structure est bien entendu récursive ou emboitée : un satellite peut être lui même une structure composite complexe.

On observe cependant des situations plus complexes. Ainsi un noyau peut-il gouverner plusieurs satellites, éventuellement de statuts différents (une définition suivie d'un conseil, un avertissement au milieu d'instructions). Un cas courant de multiplicité de satellites dans les textes procéduraux est celui d'un titre (énoncé d'un but) et des instructions qui permettent de réaliser ce but. Dans un texte, le réseau de relations devient alors très complexe, suivant une structure de graphe orienté. de plus, les relations noyau-satellite(s) sont souvent floues et ambigues et varient selon le point de vue. Enfin, le caractère hiérarchique de certaines relations est difficile à établir et peut dépendre du contexte.

Comme cela est souvent indiqué dans la littérature sur les relations du discours, celles-ci peuvent être en très

grand nombre, et de définitions variables selon les auteurs ou les annotateurs. Cette situation est nettement plus complexe, pour faire un parallèle, que dans le cas de la relation prédicat arguments ou ajouts, où des relations thématiques sont souvent employées et relativement bien maitrisées. Si l'on considère par exemple des textes techniques, on observe que chaque genre peut avoir quelques structures spécifiques. C'est le cas par exemple des instructions, du sommaire et des pré-requis dans les procédures.

1.3 Caractériser les structures discursives

De nombreux auteurs se sont attaqués au problème difficile qui consiste à définir un formalisme grammatical pour reconnaitre les structures discursives. Nous faisons ici en quelque sorte une synthèse des principales difficultés. Si l'on veut caractériser la structure grammaticale d'une structure discursive, il convient de définir avec précision :

– Comment identifier un objet discursif, sur quelles bases linguistiques, pragmatiques, typographiques, etc. (DiEugenio et al. 96). Certaines relations sont relativement bien marquées dans la plupart des cas, alors que d'autres sont rarement marquées ou ne s'y prêtent pas. Les discours en langue contrôlée ou à visée finalisée, visant l'efficacité et la clarté, développent en général des marques nettement plus évidentes, comme, par exemple, dans le discours procédural ou didactique. Les marques peuvent de surcroit être ambiguës entre plusieurs relations. Assez souvent, enfin, on remarque qu'un noyau se trouve identifié parce qu'un satellite a été identifié et peut lui être associé : les satellites sont souvent mieux marqués que leur noyau.
– Comment délimiter l'objet discursif une fois identifié ? D'autres marques (ponctuation, typographie, connecteurs, etc.) peuvent être considérées en complément des marques identifiantes indiquées ci-dessus. Une approche intéressante vise à développer la notion de structure de discours élémentaire (EDU (Schauer 06)), comparable, dans la phrase, à la proposition. Il convient d'en évaluer l'utilisabilité. Des groupes d'EDUs peuvent parfois considérées et évaluées comme étant une structure discursive autonome (par exemple sur la base de la théorie du centrage).
– Comment, une fois une structure discursive identifiée, la relier à une ou plusieurs autres structures ? La difficulté est ici d'identifier les structures exactes à mettre en relation, par exemple une énumération (satellite) doit être liée exactement à l'élément initiateur de cette énumération, comme par exemple un terme ou une expression plus générique (un titre). Les composants sont souvent contigüs, mais leur délimitation peut s'avérer très difficile.

On se trouve donc devant une triade : **identification, délimitation, liage** (des différents protagonistes de la relation).

2 Le formalisme grammatical de <TextCoop>

Le formalisme grammatical de <TextCoop>, DisLog, étend les possibilités expressives des approches à base d'expressions régulières et les adapte aux besoins de l'analyse de discours, en y intégrant un composant de raisonnement souvent utile dans l'analyse de telles structures. Notre approche s'appuie aussi sur les travaux, maintenant assez anciens, mais toujours actuels, des grammaires logiques (DCGs, XGs, MGs, etc.), qui s'appuient sur des modèles d'exécution inspirés des programmes logiques, dont Prolog. D'autres schémas de stratégie, par exemple à base de contraintes ou utilisant du parallélisme ET-OU sont possibles. Cette approche nous parait intéressante pour l'analyse du discours en raison de son caractère déclaratif marqué, de son indépendance relative aux stratégies de traitement, et aussi de son aptitude à intégrer naturellement des modules de raisonnement et des structures de contraintes puissantes (par exemple des structures de traits typés, des contraintes d'arbres).

Le formalisme que nous proposons ici peut aussi bien permettre de coder des règles d'analyse de structures discursives conçues par des linguistes et codées manuellement que des règles issues de mécanismes d'apprentissage à partir de textes annotés, qui produisent en sortie des formes contraintes. Il est aussi possible de coder globalement une forme noyau-satellite que ces mêmes formes séparemment. Ce dernier choix est nécessaire lorsque la combinatoire entre noyau et satellite est élevée, ou que ces constituants sont discontinus, ce qui est assez fréquent.

2.1 DisLog : le formalisme grammatical

Le langage DisLog offert par <TextCoop> comprend les symboles suivants, ils suivent en général la syntaxe de Prolog et des DCGs (grammaires à clauses définies) :

– des **symboles pré-terminaux et non terminaux**. Les pré-terminaux se dérivent directement en des entrées lexicales ou des expressions (caractéristiques de structures rhétoriques ou de domaines, par exemple) ou bien encore en des marques typographiques ou des marques d'annotations (html, XML, ...). Ces marques d'annotations peuvent faire référence à des structures déjà identifiées. Les symboles non terminaux font appel à des grammaires, essentiellement à caractère local (par exemple grammaire des expressions temporelles) ou, plus rarement, des constructions standard de la langue (SN, SV, etc.). Les règles ne s'appellent pas entre-elles. Les liens entre structures sont réalisés par des opérations de liage sélectif (voir ci-dessous). Les symboles non terminaux et préterminaux peuvent être associés à des structures de traits attribut-valeur, ceci ne sera pas développé ici, tant bien connu. Enfin, ces symboles sont utilisés soit pour identifier un type de structure discursive soit comme élément de délimitation (inclus ou exclu). Lorsqu'ils sont exclus, ils apparaissent dans un prédicat 'borne'.
– des symboles terminaux indiqués entre crochets, cette possibilité est utile lorsqu'il y a peu de choix sur ces terminaux au sein d'une règle, dans le cas contraire, il est péférable de faire appel à un préterminal,
– des indications d'optionalité ou d'itérativité sur les symboles préterminaux et non terminaux,
– des symboles permettant d'exprimer la précédence linéaire (la ','), ainsi que la co-occurence de symboles (le ' ;') si l'on veut utiliser la forme abrégée des règles (non développée ici),
– des 'gaps' qui représentent des séquences finies de mots qui ne présentent pas d'intérêt pour la règle en cours de description. La condition d'arrêt est constituée par le symbole explicite qui suit le gap. Dès qu'un tel symbole est rencontré, le gap s'arrête. Les gaps peuvent être associés à des contraintes, en particulier des symboles terminaux ou non-terminaux qui ne doivent pas être ignorés. Si un gap rencontre un tel symbole avant d'atteindre sa condition d'arrêt alors il y a échec de la règle à reconnaitre la structure. Un gap ne peut ni commencer ni terminer une règle, il doit toujours être borné explicitement par un symbole ou un terminal.
– des appels à des prédicats qui introduisent des contraintes, des connaissances à intégrer ou des calculs divers. Ceux-ci sont représentés entre accolades comme dans les DCGs.
– des fonctions d'assignation, explicites ou par défaut, d'étiquettes dédiées permettant d'étiqueter les structures reconnues avec d'éventuels attributs, calculés par les prédicats ci-dessus.

A priori, les règles sont de type 2, avec la syntaxe des DCGs. Toutefois des règles de type 1 peuvent aussi être construites. Le symbole en partie gauche de règle contient une variable qui représente le résultat : en général il s'agit de la structure complète telle que lue avec des marques d'annotation, éventuellement avec des attributs, au début et à la fin correspondant à la structure reconnue. Il est aussi possible de repositionner des composants du texte lus en entrée.

A titre d'exemple, des expressions d'avertissement du type *il est conseillé de ne jamais ACTION parce que....* se représentent simplement (i.e. sans faire de généralisation) comme ci-dessous (Fontan et al. 08). On considère ici que la structure commençant par *parce que* ne fait pas partie de l'avertissement (en fait c'est un support de l'avertissement selon les théories de l'argumentation, celui-ci est reconnu séparemment) :
avertissement(R) –> [il, est], expr([type :conseil]), [de], negation, gap([connecteur([type :cause])), borne([parce, que]).

Cette règle débute par la mention de deux terminaux, qui font partie de la structure à reconnaitre, suivie d'un pré-terminal de type conseil (indiqué ici par la structure attribut valeur dans l'argument). Elle se poursuit par un autre terminal, un non terminal qui reconnait la négation, puis un gap dont on indique qu'il ne doit pas ignorer les connecteurs de cause sur son parcours qui se termine sur la borne (une marque externe à la règle, voir ci-dessous) qui est le terminal [parce,que]. La variable R représente la structure étiquetée, qui est ici de la forme :
<avertissement> ... texte lu ... < /avertissement>.

2.2 L'insertion d'étiquettes XML

DisLog prévoit la possibilité (1) de spécifier d'autres types d'étiquettes que celles liées au symbole en partie gauche, (2) d'inclure des attributs, et (3) d'insérer à tout endroit du segment de texte lu tout autre type d'étiquette. Considérons :

<avertissement> il est <exp-conseil force="modéré"> recommandé < /exp-conseil> de ne jamais ouvrir la boite < /avertissement>

Dans cet exemple, on a inséré une balise <exp-conseil> avec un attribut (déduit de propriétés lexicales) qui

indique la force du conseil, ici 'modéré'.

L'insertion d'étiquettes XML se fait comme suit :
– par défaut en début et fin de séquence reconnue, en utilisant le non terminal donné en partie gauche de règle,
– si l'on veut insérer une autre étiquette, alors celle-ci est spécifiée explicitement par une variable d'insertion. Ceci vaut aussi si on veut ajouter un ou plusieurs attributs.
– si l'on veut insérer des étiquettes complémentaires dans la séquence reconnue, celles-ci sont aussi spécifiées par des variables d'insertion dans la partie droite de règle.
– si, d'aventure, on ne veut rien insérer en début et fin de séquence, on emploie la notation $noinsert.

Les variables d'insertion sont représentées par : $insert1, $insert2, qui sont instanciées explicitement en fin de règle dans une section 'calculs' entre accolades. On peut employer des variables qui proviennent soit de déductions soit de caractéristiques héritées de données lexicales. Pour l'exemple ci-dessus, on doit ajouter dans la règle :
avertissement(R) –> [il, est], $insert1, expr([type :conseil, force :F]), $insert2, [de], negation,
gap([connecteur([type :cause])), borne([parce, que]), { $insert1= <exp-conseil, force :F>, $insert2=< /exp-conseil> }.

avec la donnée lexicale : expr([type :conseil, force :modéré]) –> [recommandé].

2.3 Les règles de liage sélectif

Les règles de liage sélectif permettent de lier deux structures ou plus comme évoqué dans l'introduction. L'objectif est de lier noyau et satellite(s), ou tout autre lien que l'on souhaite établir (par exemple, connecteur - EDU, etc.). Les règles de liage ont donc un statut de non-terminaux : elle lient entre-elles des règles, permettant de construire des arbres partiels dans un texte, indiquant les structures discursives qui sont en relation. Par exemple si l'on veut lier les structures discursives a et b pour former c, on peut définir une règle de liage comme suit :
c(R) –> [<a>], gap, [< /a>], gap, [<b>], gap, [< /b>].
ce qui produira : <c> <a>, ... < /a>, ... <b>, ... < /b> < /c>.
où les structures a et b sont reproduites telles quelles.

De façon plus concrète, si l'on considère la structure duale des arguments : conclusion-support, comme dans :
Il est capital d'insérer verticalement la carte mère car vous riquez d'endommager les connecteurs.,
on aurait la règle de liage sélectif :
argument(R) –> [<conclusion>], gap, [< /conclusion>], connecteur([type :cause]), [<support>], gap, [< /support>].
et la structure résultante est :
<argument> <conclusion> Il est capital d'insérer verticalement la carte mère < /conclusion>, car <support> vous riquez d'endommager les connecteurs < /support> < /argument>.

2.4 Les règles de correction

<TextCoop> permet la définition de règles de réécriture sur les balises permettant de repositionner certaines balises qui pourraient ne pas être positionnées correctement.

Un emploi immédiat est lié au ré-équilibrage des balises qui peuvent se chevaucher, en particulier lorsque les conditions de délimitation d'une règle sont trop peu contraintes. Typiquement, ces règles permettent de corriger une situation telle que :
<a>, ...<b> < /a>, ... < /b> en <a>, ... < /a>, ... <b>, ... < /b> .

Les règles de correction ont la même forme que celle ci-dessus, la variable R contenant la structure corrigée :
corriger([<A>], gap, [< /A>], gap, [<B>], gap, [< /B>]) –> [<A>], gap,[<B>],gap, [< /A>], gap, [< /B>].
On notera qu'ici A et B sont des variables représentant a priori n'importe quel identifiant de balise.

2.5 L'art d'écrire des règles

Dans notre approche, les règles sont écrites pour l'instant totalement de façon manuelle, à partir d'analyse de documents et de repérage de marques. Toutefois, le formalisme a été conçu pour accueillir les spécifications,

relativement ouvertes, de systèmes basés sur l'apprentissage. Nous souhaitons aussi introduire des outils d'aide à l'écriture de règles, par exemple basés sur des techniques de bootstrapping. Ceci reste toutefois à approfondir car l'analyse de discours a des caractéristiques très différentes de celle de la phrase où bootstrapping et apprentissage ont été largement testés. Notre expérience est que l'écriture de règles qui reconnaissent des structures de discours et les lient sont complexes : elles présentent peu de marques explicites, ce qui les rend ambiguës, elles couvrent aussi très souvent des fragments de texte conséquents. Ces raisons font que nous avons privilégié dans notre approche une écriture manuelle des règles que nous associerons à un ensemble d'outils de visualisation de façon à guider les auteurs. Cette écriture, nous pensons, permet d'accéder à une meilleure adéquation linguistique et un meilleur niveau de généralisation.

Par exemple, c'est à ce niveau que des mécanismes de raisonnement peuvent être introduits. Dans le cas ci-dessous :
La confiture se prépare avec des fruits rouges (cassis, fraises, framboises) afin de ...
pour identifier que la structure entre parenthèses est une illustration, faute de marques explicites, via le contrôle : cassis est_un 'fruit rouge', etc. une terminologie simple est nécessaire. La règle suivante intègre noyau et satellite et s'écrit, par exemple :
illustration(R) –> Nom(Type), ['('], liste_Noms(Type1), [')'], { subsume(Type,Type1) }.

Enfin, l'analyse de structures du discours mène à de nombreuses ambiguités. Les règles décrites par les auteurs reflètent ces ambiguités. Notre système est conçu soit pour refléter toutes les analyses possibles (mode mise au point) soit pour privilégier un choix a priori (via le mécanismes de cascades de règles ou des heuristiques). Toutefois, les techniques d'interprétation des programmes logiques pourraient permettre des modes intermédiaires, ainsi qu'une interprétation à base de contraintes 'actives' permettant de produire l'ensemble des analyses possibles.

2.6 Gestion de la concurrence entre règles

Nous proposons ici quelques contraintes qui gèrent la concurrence entre règles. Les règles étant parfois trop permissives (ou pourrait parler de principe productif comme pour la syntaxe X-bar), il est nécessaire d'ajouter des contraintes qui en limitent la puissance (on pourrait parler de principes restrictifs). Notre système fonctionne par segments appelés unités textuelles. Celles-ci peuvent être des paragraphes, des sections, des arbres (pour des documents semi-structurés, etc.). Nous ferons une présentation ici illustrée de ces contraintes.

Certaines structures discursives ont des formes très proches, difficiles à distinguer. <TextCoop> offre actuellement deux possibilités pour gérer la concurrence au niveau de la reconnaissance de structures. Le moteur de <TextCoop>, présenté ci-dessous, exécute les règles en cascades. Le langage de <TextCoop> permet de spécifier l'ordre dans lesquels les règles sont exécutées.

Nous appellerons ci-dessous *paquet de règles* l'ensemble des règles qui sont liées à la reconnaissance d'une structure donnée (par exemple, support d'argument, illustration, reformulation) identifiée par l'emploi d'un symbole identique en partie gauche de règle. L'analogie avec les paquets de clauses en Prolog est immédiate. <TextCoop> permet de spécifier une structure d'ordre (éventuellement partiel) qui indique dans quel ordre les paquets de règles doivent être exécutés. Ainsi dans :
titre < prérequis < sommaire.
les règles reconnaissant les titres seront exécutées d'abord puis celles liées aux pré-requis, etc. sans possibilité de retour arrière.

Associé à ce mécanisme de cascades, il est possible de définir des *zones fermées* où une fois une zone reconnue, aucune règle ne pourra être appliquée sur cette zone. Une zone fermée est définie sur un segment de texte inclus dans une balise ouvrante et fermante du même type. Lorsque la zone est identifiée, il n'est pas possible de tenter d'appliquer d'autres règles à l'intérieur de cette zone. Par exemple, la balise <titre> définie dans le traitement des procédures introduit une telle zone :
zone_fermee([titre]).
Un titre ressemble en effet, quant à sa structure, à une instruction : *monter votre mezzanine*, on ne veut pas, une fois un titre reconnu, l'étiqueter aussi comme une instruction. <TextCoop> permet de définir une liste de zones fermées. Cette liste comprend des structures de deux types : (1) des structures qui peuvent être reconnues par plusieurs paquets de règles, mais où l'on veut privilégier un choix et ne pas provoquer de double étiquetage, afin de limiter les problèmes d'ambiguités (cut 'rouge' en Prolog), (2) des structures discursives 'terminales', c'est à

dire qui ne doivent pas être davantage décomposées, tout au moins par rapport à la grammaire de discours telle qu'elle est écrite. Par exemple, la structure de pré-requis (liste d'ingrédients ou d'équipements) est analysée dans les procédures comme une structure terminale (mais elle pourrait être plus finement analysée dans un autre cadre). Ce second cas est défini d'abord pour des raisons d'efficacité (comme un cut 'vert' en Prolog).

Outre la possibilité de zones fermées, DisLog offre la possibilité de spécifier d'autres types de contraintes. Tout d'abord on peut indiquer si une structure doit en dominer une autre :
dom(instruction,but).
indique que toute structure de type but est dominée par une instruction. De la même façon, on peut indiquer que deux structures doivent être dans deux branches différentes de la structure reconnue, sans aucune dominance possible :
not_dom(instruction,avertissement).
Une instruction ne peut contenir un avertissement. Enfin, nous permettons de spécifier directement une relation rhétorique :
rel_rhetorique(noyau,satellite,structure_englobante)., comme dans :
rel_rhetorique(conclusion_avt,support_avt,avertissement).
comme décrit ci-dessus. A ce stade aucune contrainte de précédence n'est donnée. Si l'on veut que le noyau soit toujours avant le satellite, il faut ajouter :
prec(conclusion_avt,support_avt). La contraite rel_rhetorique inclut donc la contrainte 'sister' et un liage sélectif des deux premiers éléments spécifiés.

3 Le moteur de <TextCoop>

Nous décrivons ici le fonctionnement du moteur <TextCoop> qui est, pour l'heure, un interpréteur. Nous indiquons d'abord la forme des règles telles que traitées par cet interpréteur puis le fonctionnement du moteur lui-même. D'autres fonctionnements peuvent être envisagés, sur le même schéma que celui des DCGs qu'il ne fait que généraliser. Nous ne nous étendrons pas ici sur les aspects théoriques du système. Les textes analysés étant composés de phrases à nombre de mot finis, les gaps ignorant eux aussi des suites finies de mots, on peut, via énumérabilité récursive, appliquer le théorème du point fixe pour donner une sémantique déclarative 'simple', comme dans les programmes logiques en général.

3.1 Traduction des règles

Le moteur fonctionne comme un interpréteur, les règles ainsi que les divers dispositifs présentés ci-dessus sont donc traduits sous forme de structures de données directement utilisables par le moteur. Pour les règles, cette structure est proche de celle définie pour les DCGs dans un mode interprété, elle a la forme suivante :
forme(Identifiant,Entrée,Sortie,Partie_droite,Résultat).
où :
– Identifiant est le nom du symbole en partie gauche de règle,
– Entrée et Sortie contiennent le texte en cours de traitement, ceci traduit la technique des listes de différences des DCGs, dont nous avons un besoin explicite ici pour reconstruire le texte étiqueté,
– Partie_droite est la partie droite de règle, développée ci-dessous,
– Résultat est la variable R, résultat de l'analyse (avec étiquetage) comme présenté ci-dessus.

Les symboles en partie droite sont représentés sous forme de liste, après développement complet de la règle lorsque la forme abrégée est utilisée. Les symboles terminaux et non terminaux sont augmentés de trois arguments :
– un argument qui représente la chaine de mots couverte par ce symbole,
– les deux variables qui représentent la liste de différence propre à ce symbole lorsqu'il est développé.

Ainsi, le symbole préterminal : expr([type :conseil]) qui représente une expression (terminale) de type conseil est-il traduit en : expr(EXPR, [type :conseil],E0,E1). En ce qui concerne le symbole gap, les trois mêmes variables sont ajoutées ainsi qu'une marque qui indique le symbole d'arrêt du gap, cette marque est l'identifiant du symbole qui suit le gap dans la règle et ses contraintes.

Plus globalement, la règle suivante, qui traite de la structure d'une conclusion de conseil :
concl_conseil –> pro(_), aux([ty :etre]), gap([supconseil]), expr([type :conseil]), gap([supconseil]), mfin(_).

est traduite de la façon suivante :

```
forme(c-cons-fr, E, S, [  pro(PRO,_,E,E2), aux(AUX,etre,E2,E3),
     gap(supconseil, [expr,conseil], E3,E4,Saute1),expr(EXPR,conseil,E4,E5),
     gap(supconseil, [mfin,_], E5,E6,Saute2),   mfin(MFIN,_,E6,S)], [], % no reasoning
    [ '<concl-cons>' ,PRO, AUX, Saute1, EXPR, Saute2, '</concl-cons>', MFIN ]).
```

'aux' demande l'auxiliaire être, 'mfin' est une marque de fin, répertoriée dans le lexique. Les deux symboles gap contiennent les variables Saute1 et Saute2 qui indiquent la chaine de mots qu'ils ont parcourue, et qui sera ignorée, jusqu'à rencontrer le symbole suivant dans la règle. L'argument qui contient les restrictions (le seul présent dans la structure initiale) peut être soit une constante Prolog, dont l'interprétation est directe soit une structure de traits, qui est interprétée de façon standard, avec la subsomption sur les traits sémantiques.

3.2 Le fonctionnement global du moteur

Le moteur de <TextCoop> est écrit en Prolog SWI, et tous les composants linguistiques attachés sont aussi implémentés dans ce cadre. Nous avons cherché à optimiser les traitements sans que ceci soit la priorité : l'objectif sont les traitements de structures du discours en mode batch.

Comme indiqué ci-dessus, le moteur fonctionne par cascades de règles, suivant les priorités énoncées. Il n'y a pas de retour arrière sur les étapes précédentes d'une cascade. Une étape de cascade est identifiée par le ou les symboles en partie droite de règle : des paquets de règles sont donc exécutés à chaque étape. Au sein d'une étape, les règles dans un paquet sont exécutées dans l'ordre dans lequel elle sont écrites. Une option du moteur permet d'éviter les retours arrière (couteux) sur ces règles. Cette option est désactivée en phase de mise au point. les contraints énoncées sont vérifiées à chaque étape de l'exécution des règles.

A titre expérimental, l'interpréteur est pourvu, outre la stratégie de traitement de la gauche vers la droite, d'une stratégie inverse, de la droite vers la gauche. Celle-ci est utilisée lorsque les marques identifiantes d'une relation rhétorique sont placées à la fin de la structure, alors que la structure ne débute que par une marque de délimitation. C'est le cas, par exemple des structures d'illustration du type : *(a,b,c,.., par exemple)*, ou la parenthèse ouvrante sert de délimiteur (elle est peu discriminante de la relation) et où la marque 'par exemple' est située en fin de chaine. Les résultats que nous obtenons indiquent un meilleur taux de reconnaissance et une meilleure efficacité. Cette stratégie sera automatisée, à partir de l'analyse de la position des marques identifiantes.

Le résultat final de l'analyse est le texte donné en entrée augmenté de balises XML qui délimitent et caractérisent les structures analysées.

3.3 Evaluation

La première version du moteur est à présent disponible avec son environnement. Celle-ci a été réalisée dans l'objectif de valider les idées et de définir précisémment les fonctionnalités internes et externes utiles en analyse discursive. L'optimisation du code n'y est que partielle. Les performances du système actuel sont les suivantes, sur la base de 30 patrons (liés au traitement des procédures), avec un lexique de 1300 mots ou expressions et un accès à un analyseur morphologique, lui aussi en Prolog. Le calcul se fait sur un PC standard. Nous traitons 60 Mo de texte (hors balises) par heure. La taille du lexique dans cette expérience est assez importante pour le traitement du discours. Dans de nombreuses applications, le lexique utile peut être nettement plus réduit. Ceci a un impact important sur les performances. Le lexique qui a servi à l'expérimentation contient 800 verbes, cependant, dans une application d'analyse de procédures, ce nombre descend à environ 150 verbes, et à des variations morphologiques très réduites. On peut alors traiter jusqu'à 200 Mo de texte par heure. Bien entendu la complexité des patrons a aussi une influence sur les performances : les gaps, la longueur des règles, les restrictions mais aussi le non déterminisme introduit par les règles d'un même paquet sont des facteurs importants, mais qui restent difficiles à analyser finement.

Il est nettement plus difficile d'évaluer la qualité de reconnaissance des règles. Cela dépend beaucoup des types de textes traités et de la qualité de conception des règles, comme dans tout type de programmation. En ce qui concerne les procédures, des résultats détaillés sont donnés dans (Fontan et al. 08). Les textes professionnels

donnent des résultats très bons, du fait de leur qualité de rédaction et de leur régularité, aussi bien au niveau de l'écriture des instructions que de celle des conseils, avertissements, définitions, commentaires ou illustrations. Nous travaillons actuellement au dévelopement de techniques basées sur le bootstrapping, mais adaptées à la problèmatique du discours, visant à améliorer la définition des règles pour un type de phénomène donné. Cela inclut les aspects structurels aussi bien que les aspects lexicaux, en particulier la caractérisation fine des marques. Ces marques sont de types très diversifiés. Pour bien organiser nos ressources lexicales, il convient d'élaborer, au sein du système, une architecture souple et modulaire, mais non redondante, des ressources lexicales (marqueurs, classes sémantiques de verbes, expressions dédiées, etc.).

4 L'environnement linguistique de <TextCoop>

Nous avons présenté essentiellement dans ce document le moteur de <TextCoop> ainsi que le formalisme des règles. Il est clair qu'un tel système ne peut fonctionner qu'en s'appuyant sur un ensemble de ressources lexicales, et ne peut être véritablement intégré dans des traitements de grande échelle que si, outre ses performances, ses modules de ressources (lexique, grammaires locales, ontologies et terminologies, etc.) et formats d'entrée sortie des documents suivent des formats normalisés. Il est essentiel aussi qu'il dispose d'un module de mise au point des règles, de visualisation des résultats et des recommandations et une méthode pour son intégration dans des applications.

Pour l'heure, nous avons développé et intégré différents modules de ressources qui sont plus particulièrement pertinents pour l'analyse du discours que l'on peut utiliser tels quels dans les règles, comme par exemple :
- des listes de connecteurs typés : de cause, concession, temps, etc.
- des listes de termes spécifiques pouvant être utilisés comme marques : marques de l'illustration, de la reformulation, du développement, etc.
- des listes de verbes par classes et sous-classes sémantiques, inspirées des classes de WordNet,
- des listes de termes avec polarité négative ou positive, ceci est utile en argumentation et en analyse d'opinions.
- des grammaires locales : expression du temps, de la quantité, expressions évaluatives simples, etc.
- enfin, des modules qui contiennent quelques règles simples pour reconnaitre des structures telles que : illustration, reformulation, but, condition, définition et élaboration (une relation nettement plus complexe et composite). De façon plus générale, nous souhaitons étudier comment on peut greffer un analyseur de phrases au sein de <TextCoop> pour le compléter.

En matière de visualisation des résultats, nous avons conduit une expérimentation très concluante en utilisant NAVITEXTE (http ://panini.u-paris10.fr/jlm/?Start :projets :NaviTexte), même si cette interface est un peu trop élaborée pour nos besoins. Nous envisageons à présent une visualisation qui soit davantage sous forme d'abre, tout en préservant la structure du document original. En matière de normalisation des données et de certains traitements, nous étudions le cadre de UIMA (http ://www.uima-fr.org/) qui est certainement une direction forte. L'aide à la mise au point des règles est difficile à concevoir dans le cadre du discours. Dans notre cadre, elle se limite actuellement à quelques recommandations sur leur forme et la façon de les simplifier ou de les généraliser. Nous envisageons de développer un analyseur qui détecterait certains types d'erreurs ou d'incompatibilités entre règles, comme cela existe dans certains compilateurs de règles. Etant dans une étape qui demeure expérimentale, il nous faut bien identifier les besoins 'raisonnables' en matière de mise au point de règles et de ressources avant d'en réaliser une implémentation. L'architecture des ressources linguistiques utilisées dans les règles, pour éviter les doubles et les incohérences est aussi un sujet d'étude actuel.

5 Les applications

Le projet <TextCoop> est à l'origine dédié à l'analyse des procédures (Delpech et al. 2008). Nous avons donc conduit nos premières expérimentations sur les structures liées aux procédures, qui ont des caractéristiques assez diversifiées et qui permettent de bien tester à la fois le moteur et l'écriture des règles. Un élément intéressant est que, dans ce projet, nous avons analysé à la fois des structures linguistiques connues et des structures dédiées (titres, instructions, etc.). Les caractéristiques globales de ces structures sont les suivantes :
- titres : ressemblent à des instructions, bien que souvent très elliptiques (verbe ou objet sous-entendu), les règles de reconnaissance s'appuient sur de nombreuses considérations typographiques.

– instructions : nous traitons ici en fait de composés instructionnels qui peuvent contenir plusieurs instructions élémentaires. Dans ces règles, les difficultés sont la délimitation des instructions ainsi que le grand nombre de symboles gaps introduits. L'identification des instructions se fait simplement sur la base de verbes et d'indications morphosyntaxiques.
– prérequis : longue structure sous forme de liste, pauvre en verbes (les gaps excluent les verbes), elle se situe en général au début du document, elle est parfois difficile à identifier à cause de conflits avec des sommaires ou de la publicité. La structure typographique, codée dans les règles, est un élément essentiel.
– conseils et avertissements, ont la forme d'arguments : séquence d'une conclusion (une instruction particulière) suivie d'un ou plusieurs supports (les difficultés à prévoir si on ne fait pas ce qui est demandé). Ces deux structures sont liées et demandent un traitement conjoint via deux paquets de règles. Des règles de liage lient les deux composants. Conseils et avertissements ressemblent à la structure des instructions, mais sont pourvus de marqueurs spécifiques (Fontan et al. 2009) riches et diversifiés. Ces structures sont traitées dans une étape de cascade qui précède celle des instructions. Des zones fermées évitent les doubles analyses.

Par ailleurs, selon le même schéma, nous avons développé une analyse d'avis consommateurs où nous faisons ressortir la structure du discours, globalement : type de produit, circonstances d'achat, liste d'arguments (propriétés avec des avis positifs et négatifs), recommandation, commentaires. Bien que ce travail soit encore dans un stade expérimental, <TextCoop> est bien adapté pour rendre compte de la structure globale de ce type de texte.

Enfin, <TextCoop> a été utilisé ponctuellement comme outil d'exploration à l'enrichissement de documents semi-structurés, comme par exemple l'ajout d'une zone de pré-requis dans les procédures qui n'en ont pas.

En perspective, via un projet ANR, LELIE, qui débute, <TextCoop> sera utilisé pour l'analyse et la prévention des risques industriels tels qu'ils peuvent apparaitre dans les procédures (mauvaise rédaction, non suivi d'exigences réglementaires ou métier). Dans ce cadre pré-industriel, <TextCoop> recevra un environnement plus professionnel.

6 Conclusion

Dans ce document, nous avons présenté les principales caractéristiques de <TextCoop>, un environnement basé sur les grammaires logiques dédié à l'analyse de structures discursives via le langage offert par DisLog. Nous avons détaillé en particulier la structure des règles et du moteur. Nous avons indiqué au fur et à mesure du texte l'état du travail, les performances et les orientations en particulier en matière d'environnement, d'aide à l'écriture de règles et de développement applicatif.

S'il existe plusieurs plateformes très élaborées pour l'analyse de la phrase et de structures plus petites, il y a peu de plateformes qui soient dédiées à l'analyse de structures du discours, qu'elles soient rhétoriques ou dédiées à des applications. Outre cette originalité relative, un élément important dans sa viabilité est le développement d'aide à l'écriture de règles et des éléments du langage qui y sont associés.

Le code <TextCoop> et des données linguistiques associées seront prochainement mis à disposition sous licence GPL.

Remerciements

Ce projet est soutenu par un projet de coopération franco-indien (IFCPAR) ainsi que par l'ANR (projet terminé TextCoop, et projet actuel LELIE).

Références

Amgoud, L., Bonnefon, J.F., Prade, H., *An Argumentation-based Approach to Multiple Criteria Decision*, in 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'2005, Barcelona, 2005.

Amgoud, L., Parsons, S., Maudet, N., *Arguments, Dialogue, and Negotiation*, in : 14th European Conference on Artificial Intelligence, Berlin, 2001.

Anscombre, J.-Cl. Ducrot, O., *Interrogation et Argumentation*, in Langue francaise, no 52, L'interrogation, 5 - 22, 1981.

Aouladomar, F., Saint-Dizier, P., *An Exploration of the Diversity of Natural Argumentation in Instructional Texts*, 5th International Workshop on Computational Models of Natural Argument, IJCAI, Edinburgh, 2005.

Bouffier, A., Poibeau, T., *Re-engineering free texts to obtain XML documents : a discourse based approach*, RANLP 2007. Carberry, S., Plan Recognition in natural language dialogue, Cambridge university Press, MIT Press, 1990.

Cruse, A., *Lexical Semantics*, Cambridge Univ. Press, 1986.

Delin, J., Hartley, A., Paris, C., Scott, D., Vander Linden, K., *Expressing Procedural Relationships in Multilingual Instructions*, Proceedings of the Seventh International Workshop on Natural Language Generation, pp. 61-70, Maine, USA, 1994.

Delpech, E., Saint-Dizier, P., Investigating the Structure of Procedural Texts for Answering How-to Questions, LREC 2008, Marrakech.

Davidson, D., *Actions, Reasons, and Causes*, Journal of Philosophy, 60, 1963

Di Eugenio, B. and Webber, B.L., Pragmatic Overloading in Natural Language Instructions, International Journal of Expert Systems, 1996.

Lionel Fontan, Patrick Saint-Dizier. Constructing a Know-How Repository of Advices and Warnings from Procedural Texts. Dans ACM International Conference on Document Engineering, Sao Paolo, Dick Bulterman, Luiz Soares (Eds.), ACM, p. 234-240, september 2008.

Gardent, C., Discourse tree adjoining grammars, report nb. 89, Univ. Saarlandes, Saarbrucken, 1997.

Kosseim, L., Lapalme, G., *Choosing Rhetorical Structures to Plan Instructional Texts*, Computational Intelligence, Blackwell, Boston, 2000.

Mann, W., Thompson, S., Rhetorical Structure Theory : Towards a Functional Theory of Text Organisation, TEXT 8 (3) pp. 243-281, 1988.

Marcu, D., The Rhetorical Parsing of Natural Language Texts, ACL 1997.

Marcu, D., Au unsupervised approach to recognizing Discourse relations, ACL 2002.

Moschler, J., *Argumentation et Conversation, Eléments pour une Analyse Pragmatique du Discours*, Hatier - Crédif, 1985.

Rosner, D., Stede, M., *Customizing RST for the Automatic Production of Technical Manuals*, in R. Dale, E. Hovy, D. Rosner and O. Stock eds., Aspects of Automated Natural Language Generation, Lecture Notes in Artificial Intelligence, pp. 199-214, Springler-Verlag, 1992.

Saito, M., Yamamoto, K., Sekine, S., Using Phrasal Patterns to Identify Discourse Relations, ACL, 2006.

Schauer, H., From Elementary Discourse Units to Complex Ones, ACL 2006.

Takechi, M., Tokunaga, T., Matsumoto, Y., Tanaka, H., *Feature Selection in Categorizing Procedural Expressions*, The Sixth International Workshop on Information Retrieval with Asian Languages (IRAL2003), pp.49-56, 2003.

Vander Linden, K., *Speaking of Actions Choosing Rhetorical Status and Grammatical Form in Instructional Text Generation* Thesis, University of Colorado, 1993.

Webber, B., D-LTAG : extending lexicalized TAGs to Discourse, Cognitive Science 28, pp. 751-779, Elsevier, 2004.

Wright, von G.H., Explanation and understanding, Cornell university Press, 2004.