Analyse relâchée à base de contraintes

Jean-Philippe Prost*
LIFO, Université d'Orléans
JPProst@gmail.com

Résumé. La question de la grammaticalité, et celle duale de l'agrammaticalité, sont des sujets délicats à aborder, dès lors que l'on souhaite intégrer différents degrés, tant de grammaticalité que d'agrammaticalité. En termes d'analyse automatique, les problèmes posés sont de l'ordre de la représentation des connaissances, du traitement, et bien évidement de l'évaluation. Dans cet article, nous nous concentrons sur l'aspect traitement, et nous nous penchons sur la question de l'analyse d'énoncés agrammaticaux. Nous explorons la possibilité de fournir une analyse la plus complète possible pour un énoncé agrammatical, sans l'apport d'information complémentaire telle que par le biais de mal-règles ou autre grammaire d'erreurs. Nous proposons une solution algorithmique qui permet l'analyse automatique d'un énoncé agrammatical, sur la seule base d'une grammaire modèle-théorique de bonne formation. Cet analyseur est prouvé générer une solution optimale, selon un critère numérique maximisé.

Abstract. The question of grammaticality, and the dual one of ungrammaticality, are topics delicate to address when interested in modeling different degrees, whether of grammaticality or ungrammaticality. As far as parsing is concerned, the problems are with regard to knowledge representation, processing, and obviously evaluation. In this paper, we concentrate on the processing aspect and we address the question of parsing ungrammatical utterances. We explore the possibility to provide a full parse for an ungrammatical utterance without relying on any kind of additional information, which would be provided by mal-rules or other error grammar. We propose an algorithmic solution in order to parse an ungrammatical utterance using only a model-theoretic grammar of well-formedness. The parser is proven to generate an optimal solution, according to a maximised criterion.

Mots-clés : grammaticalité, analyse syntaxique, contraintes, syntaxe modèle-théorique.

Keywords: grammaticality, syntactic parsing, constraints, Model-Theoretic Syntax.

1 Introduction

La notion même de grammaticalité est sujette à différentes interprétations, selon le cadre théorique auquel elle s'applique. Ainsi, il est possible de dégager trois grandes interprétations aux différences notables, qui s'appliquent à des cadres formels différents. Au sens génératif du terme, la grammaticalité est une notion strictement binaire selon laquelle une phrase est soit grammaticale, soit agrammaticale. Cette notion générative est étroitement liée au processus d'analyse et à l'existence ou non d'un arbre syntaxique pour cette phrase : une phrase est grammaticale

*Ces travaux ont été menés conjointement au CLT à Macquarie University, Sydney, Australie, et au LPL à l'Université de Provence.

si et seulement si il est possible d'en générer une analyse à l'aide d'une grammaire donnée. L'objet théorique de la Syntaxe Générative-Énumérative[1] (GES) ne concerne que l'explication des phénomènes linguistiques qui relèvent de la grammaticalité, et ne permet pas d'expliquer des phénomènes atypiques, tels que l'ouverture lexicale (apparition permanente de nouveaux termes), les fragments bien-formés (*e.g.* "Il me semble que...", "Oui, mais alors je...", *etc.*.), l'agrammaticalité, ou encore la grammaticalité graduelle (*i.e.* gradience) (Pullum & Scholz, 2001).

Au sens de la Théorie de l'Optimalité (OT) (Prince & Smolensky, 1993), la grammaticalité s'applique cette fois non pas à une phrase, mais à une structure : par définition, une structure candidate pour une phrase est grammaticale si et seulement si cette structure est optimale parmi l'ensemble des candidats. Dès lors, toute phrase pour laquelle la fonction GEN génère au moins une structure candidate est nécessairement grammaticale au sens d'OT. Cependant, l'objet d'OT n'est pas de décrire les mécanismes de génération de structures syntaxiques (la fonction GEN est donnée), mais bien de permettre d'expliquer pourquoi une structure donnée est optimale aux yeux d'une grammaire donnée. En ce sens, la couverture d'OT en termes de phénomènes linguistiques est donc théoriquement plus étendue que celle de la GES. En revanche, la notion de grammaticalité qu'elle introduit peut poser problème, puisqu'elle ne permet pas de distinguer la bonne-formation de la mal-formation, et donc la grammaticalité de l'agrammaticalité. Ceci est lié au fait que la violation de contraintes grammaticales n'est pas nécessairement significative d'agrammaticalité, ne serait-ce que partielle. Tout au plus, les différentes structures candidates associées à un énoncé peuvent être ordonnées en fonction de leur acceptabilité grammaticale, comme le permet la Théorie Linéaire de l'Optimalité (Keller, 2000). Si la nécessité de pouvoir clairement distinguer grammaticalité et agrammaticalité au sein d'une théorie linguistique est certes discutable (Meurers, 2007), il est des contextes applicatifs, comme l'apprentissage d'une langue ou la correction grammaticale, où une telle distinction s'impose.

Au sens de la Syntaxe Modèle-Théorique (MTS) (Pullum, 2007), la grammaticalité d'un énoncé est définie par la consistance d'un modèle pour cet énoncé, c'est-à-dire la satisfaction par ce modèle des contraintes grammaticales. La différence fondamentale d'avec la syntaxe générative vient de ce que le rôle de la grammaire n'est plus de générer mais de valider des structures linguistiques. Une grammaire MTS est alors un ensemble de contraintes non-ordonnées, qui peuvent être évaluées indépendamment les unes des autres. Cette propriété facilite également, en théorie, la relaxation de certaines contraintes, ce qui permet d'intégrer des degrés de grammaticalité et d'agrammaticalité, selon des modalités à définir. Les cadres formels pour la MTS les plus communs sont HPSG (Pollard & Sag, 1994), ou la famille des formalismes basés sur les grammaires de dépendances par contraintes (Maruyama, 1990).

Au-delà des possibilités offertes par tel ou tel cadre en matière d'analyse robuste, la question générale que nous posons est celle, double, de la représentation des propriétés syntaxiques d'un énoncé agrammatical, et de l'estimation de son degré de grammaticalité. Peut-on analyser les propriétés syntaxiques d'un énoncé agrammatical, en ne disposant que d'une grammaire de bonne-formation[2] ? L'aspect du problème qu'il est important de souligner concerne la nature de l'information syntaxique dont on dispose au sujet d'un énoncé, qu'il soit grammatical ou non. Cette information doit d'une part nous permettre de conclure quant à la grammaticalité d'un

[1] Le terme *Generative-Enumerative Syntax* a été introduit dans (Pullum & Scholz, 2001).

[2] Nous faisons référence ici à bon nombre de travaux en analyse syntaxique robuste, qui font appel, en plus d'une grammaire de bonne-formation, à un complément de règles de mal-formation (Bender *et al.*, 2004; Foster, 2007). Outre leur efficacité pratique, le problème de ces approches est qu'elles ne proposent pas de solution générale pour expliquer des phénomènes qui ne seraient pas couverts par les grammaires de bonne- et mal-formation.

énoncé, et d'autre part de permettre l'estimation de son degré d'(a)grammaticalité. Dans cet article nous nous concentrons uniquement sur l'aspect analytique et n'abordons pas la question de la gradation. Nous formulons l'hypothèse selon laquelle, dans un cadre de syntaxe modèle-théorique, un modèle (*i.e.* une structure syntaxique) qui optimise le nombre de contraintes satisfaites par rapport au nombre de contraintes violées peut constituer une analyse complète plausible pour une phrase agrammaticale. Afin de tester cette hypothèse nous avons développé un algorithme d'analyse syntaxique pour le cadre modèle-théorique des Grammaires de Propriétés (GP) (Blache, 2005) qui génère la structure de constituant de mérite maximum pour une phrase donnée, qu'elle soit grammaticale ou non. Notre analyseur diffère des analyseurs existants pour les GP (Morawietz & Blache, 2002; Dahl & Blache, 2004; VanRullen, 2005) principalement de par l'optimalité de la solution proposée. Après avoir introduit et detaillé cet algorithme nous présentons les résultats de l'évaluation faite de son implantation.

2 Algorithme

Cet article introduit un algorithme d'analyse tabulaire par satisfaction relâchée (*Loose Satisfaction Chart Parsing*, LSCP), décrit par l'Algorithme 1. L'analyseur LSCP est basé sur l'algorithme d'analyse tabulaire probabiliste de Cocke-Kasami-Younger (CKY). Il utilise le même

Algorithme 1 Analyse tabulaire à base de satisfaction relâchée (*Loose Satisfaction Chart Parsing*)

```
/* Initialisation */
Create and clear the chart π : every score in π set to 0

/* Cas de base : peupler π avec des POS-tags pour chaque mot */
for i ← 1 to num_words
    for (each POS-construction T of w_i)
        if merit(T) ≥ π[i, 1, T] then
            Create constituent w_i^T, whose construction is T
            π[i, 1, T] ← {w_i^T, merit(w_i^T)}

/* Cas récursif */
/* Etape 1 : SÉLECTION de l'empan courant de référence */
for span ← 1 to num_words
    for offset ← 1 to num_words − span + 1
        end ← offset + span − 1
        K ← ∅
/* Etape 2 : ÉNUMERATION de toutes les configurations */
        for (every set partition P in [offset, ..., end])
            K_P ← buildConfigurations(P)
            K ← K ∪ K_P
/* Etape 3 : CARACTÉRISATION du système de contraintes de la grammaire */
            for (every configuration A ∈ K_P)
                χ_A ← characterisation(A)
/* Etape 4 : PROJECTION de constructions */
                /* C_A est un ensemble de constituants candidats. */
                C_A ← projection(χ_A)
                checkpoint(C_A)
/* Etape 5 : MÉMOISATION du constituant candidat optimal */
                for (every candidate constituent x ∈ C_A, of construction C)
                    if merit(x) ≥ π[offset, span, C] then
                        π[offset, span, C] ← {x, merit(x)}
        if π[offset, span] = ∅ then
            π[offset, span] ← preferred forest in K
```

squelette que le CKY, sur lequel vient se greffer un processus de *satisfaction relâchée de contraintes*, décrit plus bas. Le terme de *tabulaire* fait référence à l'utilisation d'une *table de programmation dynamique*. L'algorithme LSCP diffère cependant du CKY sur différents points.

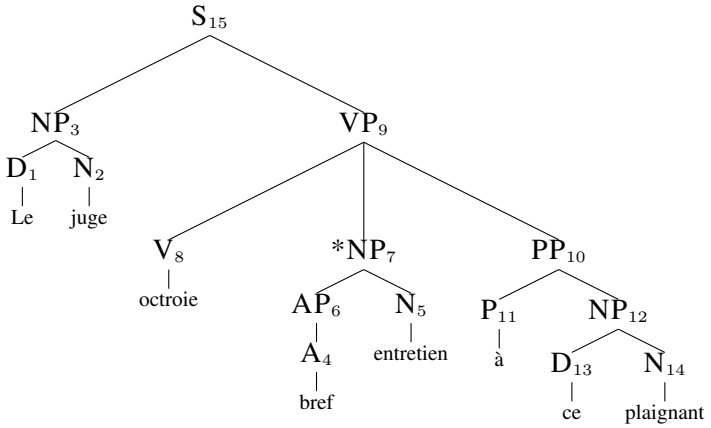

FIG. 1 – Exemple d'analyse générée par l'analyseur LSCP pour une phrase agrammaticale.

Alors que le CKY nécessite une grammaire en Forme Normale de Chomsky (CNF), le LSCP prend une grammaire de propriétés ordinaire, aucun équivalent de la CNF n'existant pour le formalisme des GP. La conséquence directe en est la génération de structures arborescentes n-aires, et non d'arbres binaires. Une autre différence tient dans l'utilisation de valeurs de *mérite* des constituants en lieu et places des probabilités. C'est sur la base de cette valeur de mérite à maximiser que l'analyseur optimise le choix de la solution. La Figure 1 illustre un exemple d'analyse générée par l'analyseur LSCP pour une phrase agrammaticale.

L'algorithme LSCP, comme le CKY, repose sur une technique de *programmation dynamique*. Ainsi, la recherche d'une solution générale optimale se décompose en sous-problèmes pour chacun desquels l'analyseur trouve une solution optimale. Ces solutions partielles intermédiaires sont dites *sous-structures optimales*. Dans notre cas, le principe fondamental qui s'applique est qu'une solution optimale n'est nécessairement composée que de sous-constituants qui tous optimisent la solution globale. En supposant que la fonction de mérite présente les propriétés adéquates quant à l'optimalité, il est aisé de montrer, par l'absurde, que s'il existe une solution de mérite optimal, et si l'un de ses sous-constituants n'optimise pas ce mérite, alors il existe nécessairement un autre sous-constituant qui, lorsqu'il est substitué au constituant sous-optimal, conduit à une solution de meilleur mérite, ce qui contredit l'hypothèse. Ces sous-structures optimales sont mémorisées dans une *table de programmation dynamique* (TPD), par un processus de *mémoisation*. La programmation dynamique permet également d'optimiser le processus en évitant les ré-itérations redondantes du même sous-problème. La TPD qu'utilise LSCP prend pour coordonnées les mots de la phrase à analyser comme abscisse, et les tailles d'empans analysés en ordonnées.

2.1 Étape de sélection

L'étape de sélection est une itération sur la taille d'empan (*span*) entre 1 et le nombre de mots que comporte la phrase à analyser, ainsi que sur la position du coin gauche de l'empan (*offset*), de façon à couvrir tous les mots de la phrase. L'itération telle que $\{\textit{offset}, \textit{span}\} = \{i, j\}$ resoud le sous-problème dont la solution est mémorisée dans π aux coordonnéees $\{i, j\}$. La sélection $\mathcal{S}$ de constituants pour l'empan courant contient tous les constituants présents dans π aux coordonnées $\pi[i, 1], \pi[i, 2], \ldots, \pi[i, j-1], \ldots, \pi[i+1, 1], \ldots, \pi[i+1, j-2], \ldots, \pi[\textit{end}, 1]$.

Notons que l'itération sur la variable *span* débute avec la valeur 1 (et non 2 comme dans le

CKY original), ce qui permet de générer des constituants dont l'empan n'est que d'un mot (par exemple, un GN seulement composé d'un N).

2.2 Étape d'énumeration

L'empan d'une configuration de constituants valide doit couvrir les mots entre *offset* et *end* (empan courant). Lorsqu'on considère une partition d'ensemblede $S = [\mathit{offset} \ldots \mathit{end}]$ chaque sous-ensemble correspond à un sous-problème déjà résolu, puisque sa cardinalité est nécessairement inférieure à celle de l'empan courant. La sous-structure optimale correspondante est donc déjà mémorisée dans π et peut donc être récupérée. Une configuration s'obtient à partir d'une partition, en combinant ensemble toutes les sous-structures de cette partition. En calculant toutes les partitions possible de S on s'assure donc que toutes les configurations possibles sont parcourues pour l'empan courant. De façon intuitive, une configuration peut être vue comme une structure de constituant dont le label de la racine (*i.e.* sa Construction) n'est pas encore connu. Lorsque nécessaire, nous utiliserons par la suite le terme de *configuration non-labélisée*.

2.3 Étape de caractérisation

Une fois les configurations énumérées, chacune d'entre elles doit être *caractérisée*. En GP, le processus de caractérisation permet d'associer une valeur booléenne à chaque contrainte caractéristique d'un constituant, selon que cette contrainte est satisfaite ou non. Ce processus de satisfaction de contrainte est ici implanté à l'aide de l'Algorithme 2 ; une configuration joue le rôle d'une assignation de variables. Plus précisément, l'assignation est faite de tous les constituants

Algorithme 2 Fonction de caractérisation

function `characterisation`($\mathcal{A} = \langle c_1, \ldots, c_n \rangle$: assignment, $\mathcal{G}$: grammar)
returns the set of evaluated properties relevant to $\mathcal{A}$,
and the set of projected constructions for $\mathcal{A}$.

/∗ Pour mémoriser le résultat de la caractérisation : ∗/
create and clear $\chi_{\mathcal{A}}$[*property*] : table of *boolean*, indexed by *property*

/∗ Pour mémoriser les constructions projetées : ∗/
create and clear $\mathcal{C}_{\mathcal{A}}$: set of *construction*

/∗ Pour mémoriser temporairement les propriétés (i.e. contraintes) à évaluer : ∗/
create and clear S : set of *property*

for (*mask* $\in [1 \ldots 2^n - 1]$)
key ← `applyBinaryMask`($\mathcal{A}$, *mask*)
if (*key* is in the set of indexes for $\mathcal{G}$) **then**
/∗ Les propriétés sont récupérées depuis la grammaire, puis évaluées ∗/
$S \leftarrow \mathcal{G}$[*key*].`getProperties`()
$\chi_{\mathcal{A}} \leftarrow$ `evaluate`(S)

/∗ Etape de projection : récupération des constructions à projeter ∗/
$\mathcal{C}_{\mathcal{A}} \leftarrow \mathcal{G}$[*key*].`getDominantConstructions`()

return $\chi_{\mathcal{A}}, \mathcal{C}_{\mathcal{A}}$

La clef *key* déterminée par `applyBinaryMask` est une combinaison de constructions (*i.e.* ces constructions de $\mathcal{A}$ pour lesquelles le bit correspondant dans le masque *mask* est positionné à 1) ; elle est utilisée, après application d'une fonction de hashage, comme un index, pour accéder aux contraintes de la grammaire qui concernent cette combinaison.

immédiats de la configuration non-labélisée courante. L'étape de Projection dans l'Algorithme

1 est présentée séparément de celle de Caractérisation à des fins de clarté ; ces deux étapes sont en fait implantées au sein d'une même fonction afin d'économiser des itérations.

La caractérisation est un processus de *satisfaction relâchée*. Les contraintes de la grammaire ($\mathcal{G}$) sont *lâchement satisfaites* (*loosely satisfied*) pour une assignation $\mathcal{A}$ en ce sens qu'elle peuvent être satisfaites ou violées. Notons que $\mathcal{A}$ ne constitue pas nécessairement une assignation pour l'ensemble des contraintes de $\mathcal{G}$. Certaines contraintes peuvent contenir des variables pour lesquelles $\mathcal{A}$ n'assigne aucun constituant. Ces dernières sont considérées comme étant trivialement satisfaites et sont donc ignorées. Pour des raisons d'efficacité le processus fait appel à une table d'indexation des contraintes, qui a été créée lors de l'initialisation de l'analyseur. Chaque contrainte de $\mathcal{G}$ figure dans la table, indexée par une *clef* (hash-code). Le détail de cette clef, ainsi que de la table d'indexation, ne sont pas l'objet de cet article. Brièvement, l'idée est d'utiliser une configuration de constituants ($\mathcal{A}$), d'en extraire la combinaison de constructions correspondante, et d'utiliser cette combinaison comme une clef d'accès pour isoler toutes les contraintes dont les variables sont compatibles avec les constructions de la combinaison. Par exemple, la combinaison (D, N) formée d'un Déterminant et d'un Nom permet d'accéder aux contraintes $\mathrm{D} \prec \mathrm{N}$ et $\mathrm{N} \Rightarrow \mathrm{D}$ (entre autres).

Bien qu'en apparence nombre de contraintes semblent être réévaluées pour les mêmes constructions, ces réévaluations ne sont en fait redondantes que dans le cas des contraintes monotones (Linéarité et Dépendance). Pour tous les autres types, le fait que chaque nouvelle configuration corresponde à une assignation différente signifie que les contraintes peuvent être évaluées différement sous chaque assignation, d'où le besoin d'une réévaluation.

2.4 Étape de projection

Conceptuellement, l'étape de projection consiste à porter un *jugement* quant à la Construction d'un constituant. Le problème est celui de la *catégorisation* d'une configuration non-labélisée au sein d'une construction, en fonction de la caractérisation de cette configuration. De façon pratique, il s'agit de labéliser des configurations non-labélisées. Une grammaire en GP est généralement présentée comme une collection de constructions, chacune d'entre elles étant spécifiée à l'aide d'un ensemble de contraintes. L'opération qui doit maintenant être effectuée nécessite d'inverser l'information, en ce sens que la connaissance d'un ensemble de contrainte doit permettre de déterminer quelles constructions peuvent être projetées. Une table d'indexation est créée à cet effet lors de l'initialisation de l'analyseur, qui permet un accès direct aux constructions concernées en utilisant une contrainte comme clef d'accès. Cette partie du processus est en fait implantée au sein de la fonction de caractérisation (Algorithme 2).

2.5 Étape de mémoisation

L'étape de mémoisation vise à la fois à mémoriser et optimiser les sous-structures : elle mémorise donc le constituant optimal (*i.e.* dont le score de mérite est maximal) pour chaque construction possible, pour une cellule donnée de la TPD. La fonction de mérite utilisée pour l'optimalité de la structure est la proportion de contraintes satisfaites, par rapport au nombre total de contraintes satisfaites ou violées. Notons que l'optimalité ne garantit pas l'unicité de la solution : il est possible d'obtenir plusieurs analyses distinctes mais de scores identiques pour un même énoncé. Ce sera typiquement le cas en présence d'une ambiguïté linguistique, telle

que dans "**le chats*". Le but du LSCP n'étant pas de régler ces cas ambigus, l'algorithme a été implanté de façon à fournir toutes les solutions équivalentes possibles.

Dans le cas où la cellule courante dans la TPD ne pourrait pas se voir affecter au moins un constituant, une *forêt d'arbres partiels* lui est attribuée à la place. Cette forêt préférée est construite au fur et à mesure de l'énumération des configurations possibles (au sein de la fonction `buildConfigurations`, lors de l'étape d'énumeration). La préférence s'établit selon l'ordre suivant :
– les constituants avec l'empan le plus large ;
– les forêts avec le plus faible nombre de constituants.
Cette préférence se traduit par une heuristique numérique, par le biais d'un *score de préférence*. Ce score est calculé de la façon suivante (où F dénote la forêt, C_i ses constituants, $\texttt{merit}(C_i)$ le mérite de chacun de ses constituants, *span* la taille de l'empan, et p_F le score de préférence associé à F) : $p_F = \textit{span} \cdot (\texttt{merit}(C_i) + \textit{span})$. p_F est une sorte de "score de la dernière chance" : lorsque le processus d'analyse ne parvient pas à trouver une construction dominante pour un ensemble de constituants, les différentes configurations de ces constituants entrent en compétition afin que l'une d'entre elles soit choisie comme structure par défaut (pour l'empan concerné). Le vainqueur est la configuration avec le meilleur score de préférence. Notons que dans le pire des cas, lorsque la TPD est entièrement peuplée de sous-structures par défaut, le résultat final est alors une séquence d'éléments du discours (*Part-of-Speech*). De cette façon, l'algorithme LSCP fournit toujours une analyse, quelle que soit la phrase en entrée. Bien évidement, puisque l'heuristique p_F n'est que l'une des possibilités de calculer une telle préférence parmi de nombreuses autres possibilités, il serait intéressant d'explorer dans plus de détail laquelle de ces possibilités permet les meilleurs résultats. La question, cependant, de savoir en quoi une forêt d'analyses partielles est meilleure qu'une autre n'est pas sans poser des problèmes, et sort du champ d'investigation de cet article. Il serait probablement utile de mémoiser également d'autres informations, telles que les partitions d'ensembles, ou les contraintes monotones. Ces dernières, en particulier, pourraient être mémoisées en s'inspirant des techniques de compilation de contraintes mises en œuvre dans le SeedParser de (VanRullen, 2005). Cette option reste cependant à explorer.

3 Évaluation

La question de l'évaluation est délicate : le but de l'analyseur LSCP (dont l'implantation est baptisée *Numbat*) étant de fournir une structure hiérarchique de constituants, tant pour les phrases bien formées que celles mal formées, il est difficile de déterminer avec certitude quelle doit être la structure de référence attendue. L'idéal serait de disposer d'un corpus de phrases grammaticales et agrammaticales, chacune étant annotée avec une unique structure de constituants (ou éventuellement une série d'alternatives). Un tel Gold Standard n'est, à notre connaissance, malheureusement pas disponible. Pour pallier ce problème nous avons mené deux évaluations distinctes, qui visent à mesurer les performances de *Numbat* sur des phrases grammaticales pour l'une, et sur des phrases agrammaticales pour l'autre. Chacune de ces deux évaluations présente ses faiblesses. L'Évaluation 1 s'effectue uniquement sur un analyse partielle des énoncés, et ne mesure donc pas l'aptitude à produire une structure hiérarchique unique, c'est-à-dire en constituants imbriqués (contrairement à une forêt). L'évaluation 2, pour sa part, mesure bien cette aptitude (une structure est dite *complète* pour une phrase lorsqu'elle est constituée d'un constituant dominant unique), mais souffre d'une absence de mesure de référence.

L'Évaluation 1, dont les résultats sont rapportés dans la Table 1, mesure les performances de l'analyseur[3] pour la bonne formation. Cette première évaluation a été effectuée selon le proto-

	Précision	Rappel	F-mesure
Total	**0.7835**	**0.7057**	**0.7416**
general_lemonde	0.8187	0.7515	0.7837
general_mlcc	0.7175	0.6366	0.6746
general_senat	0.8647	0.7069	0.7779
litteraire	0.8124	0.7651	0.788
mail	0.7193	0.6951	0.707
medical	0.8573	0.678	0.757
oral_delic	0.6817	0.621	0.649
questions_amaryllis	0.8081	0.7432	0.7743
questions_trec	0.8208	0.7069	0.7596

TAB. 1 – Évaluation 1 de *Numbat*, selon le protocole EASY

cole élaboré pour la campagne EASY d'Évaluation d'Analyseurs SYntaxiques (Paroubek *et al.*, 2003), sur une partie du corpus utilisé lors de cette même campagne. À titre de comparaison, la Table 2 rapporte les performances mesurées dans les mêmes conditions pour deux autres analyseurs que *Numbat* : l'un superficiel (VanRullen, 2005), également basé sur le formalisme des GP, et l'autre stochastique (VanRullen *et al.*, 2006). L'Évaluation 2, dont les résultats sont

	Précision	Rappel	F-mesure
Analyseur superficiel	0.7846	0.8376	0.8102
Analyseur stochastique	0.9013	0.8978	0.8995

TAB. 2 – Évaluation d'un analyseur superficiel et d'un analyseur stochastique selon le protocole EASY

rapportés dans la Table 3, mesure les performances de *Numbat* pour l'agrammaticalité. Pour cette évaluation nous avons demandé à cinq annotateurs experts, tous linguistes, de décider si la structure syntaxique fournie par *Numbat* pour chaque phrase agrammaticale était *correcte* ou non. Pour décider, les annotateurs devaient déterminer si la solution de *Numbat* était un arbre syntaxique possible et acceptable pour la phrase. Des instructions spécifiques étaient fournies pour que le jugement ne porte pas sur l'acceptabilité grammaticale de la phrase support, mais bien sur l'arbre qui lui était associé. Le corpus utilisé est celui construit artificiellement par (Blache *et al.*, 2006), qui est constitué à 94% de phrases présentant des irrégularités grammaticales. Pour les besoins de cette évaluation les mesures de Précision et Rappel ont dû être modifiées. Le nombre total de phrases en entrée est interprété comme le nombre de *prédictions*, le nombre de structures complètes est interprété comme le nombre d'*observations*, et le nombre de structures ayant fait l'objet d'un jugement humain CORRECT est interprété comme le nombre de solutions correctes. Nous obtenons ainsi les formulations suivantes :

$$\begin{aligned} \text{précision} &= \text{CORRECT}/\text{COMPLET} \\ \text{rappel} &= \text{CORRECT}/\text{total} \\ F &= 2 \cdot \text{précision} \cdot \text{rappel}/(\text{précision} + \text{rappel}) \simeq 0.71 \end{aligned}$$

[3]La grammaire utilisée pour cette évaluation est celle de (VanRullen, 2005), tandis que l'Évaluation 2 utilise une grammaire propre, basée sur la précédente. La différence entre ces deux grammaires tient essentiellement dans la profondeur des constituants décrits.

Nb. total annotaté	Nb. structures correctes	Nb. structures complètes	Précision = $\frac{Correct}{Complet}$	Rappel = $\frac{Correct}{Total}$
469	694	632	**0.74**	**0.68**

TAB. 3 – Évaluation 2 de *Numbat* pour des phrases agrammaticales

L'évaluation 1 montre que la précision obtenue par *Numbat* est du même ordre que celle obtenue par l'analyseur superficiel, pour un rappel sensiblement plus faible. Ce résultat est globalement positif, compte-tenu de ce que l'analyseur LSCP n'est initialement pas conçu pour générer des structures superficielles telles que celles attendues par le protocole EASY. L'analyse attendue pour une phrase n'est, en effet, pas un constituant unique couvrant toute la phrase, mais une succesion de constituants multiples couvrants chacun une partie seulement de la phrase, tels que GN, GV, GP, etc. De ce fait, l'analyse générée par *Numbat* repose en grande partie sur l'heuristique de préférence décrite ci-dessus, qui permet de produire une forêt d'arbres partiels. Cette heuristique n'est cependant pas conçue à cette fin, mais simplement pour fournir une analyse par défaut en l'absence d'une solution englobante. Les résultats de l'évaluation 1 montrent donc à la fois que cette heuristique pourrait être améliorée, mais également que l'analyseur LSCP est suffisament flexible pour s'adapter à différentes granularités d'analyse.

L'évaluation 2 donne, pour sa part, des indications encourageantes quant à la possibilité de générer une analyse complète pour une phrase agrammaticale, puisque 92% des phrases du corpus sont analysés par une structure de constituant complète. La mesure de Précision indique que 74% de ces analyses complètes sont évalués comme étant syntactiquement correctes, tandis que le Rappel indique que les analyses correctes représentent 68% du corpus. Comparés aux scores obtenus sur les phrases grammaticales (précision/rappel/F-mesure = 0.78/0.71/0.74), et sachant que la quasi-totalité du corpus est constitué de phrases agrammaticales (94%), les scores de l'évaluation 2 mettent en évidence la bonne performance de *Numbat* face aux énoncés agrammaticaux, par rapport aux objectifs fixés d'analyse complète.

4 Conclusion

Dans cet article nous avons abordé le problème de l'analyse syntaxique automatique du langage tant grammatical qu'agrammatical. Parmi les nombreuses questions que ce problème soulève, nous nous sommes plus particulièrement penchés sur celles de la représentation d'une structure syntaxique d'un tel énoncé, et celle de sa grammaticalité. Nous avons vu que la notion même de grammaticalité d'un énoncé se décline sous différentes formes, selon la théorie syntaxique sous-jacente. Dès lors qu'on souhaite fournir une analyse syntaxique pour un énoncé quelconque, tout en conservant la possibilité de conclure sur son caractère grammatical ou agrammatical, un cadre de syntaxe modèle-théorique s'impose pour la représentation de cette analyse. Dans un tel cadre, une analyse syntaxique de l'énoncé est un modèle (au sens de la théorie des modèles) pour la grammaire. En matière de traitement, se posent alors les questions de la génération d'un tel modèle d'une part, et celle du choix de la meilleure solution parmi un ensemble de candidats d'autre part. Nous répondons à ces deux questions en proposant une solution algorithmique qui génère l'analyse syntaxique optimale pour un énoncé quelconque, sur la seule base d'une grammaire modèle-théorique de bonne formation. Cette analyse est optimale en ce sens qu'elle maximise la proportion de contraintes grammaticales satisfaites. L'algorithme que nous présentons (LSCP) fait appel à un processus de *satisfaction relâchée* de contraintes, ainsi qu'à des techniques de programmation dynamique. La satisfaction relâchée est un mécanisme par lequel un modèle légitime pour un énoncé peut à la fois satisfaire une partie de la grammaire et en violer une autre. L'évaluation de cet analyseur montre que ses performances sur des énoncés grammaticaux sont comparables à celles d'analyseurs traditionnels. Les performances obtenues sur des énoncés agrammaticaux montrent, elles, que les structures de constituants optimales proposées comme solutions sont jugées acceptables dans une très large mesure selon

des critères humains.

Références

BENDER E. M., FLICKINGER D., OEPEN S., WALSH A. & BALDWIN T. (2004). Arboretum : Using a precision grammar for grammar checking in CALL. In *Proc. of InSTIL/ICALL2004*, volume 17.

BLACHE P. (2005). Property Grammars : A Fully Constraint-based Theory. In *Constraint Solving and Language Processing*, volume 3438 of *LNAI*. Springer.

BLACHE P., HEMFORTH B. & RAUZY S. (2006). Acceptability prediction by means of grammaticality quantification. In *Proc. of CoLing/ACL 2006*, p. 57–64 : ACL.

DAHL V. & BLACHE P. (2004). Directly Executable Constraint Based Grammars. In *Actes de JPFLC'2004*, p. 149–166.

FOSTER J. (2007). Real bad grammar : Realistic grammatical description with grammaticality. *Corpus Linguistics and Lingustic Theory*, **3**(1), 73–86.

KELLER F. (2000). *Gradience in Grammar - Experimental and Computational Aspects of Degrees of Grammaticality*. PhD thesis, University of Edinburgh.

MARUYAMA H. (1990). Structural Disambiguation with Constraint Propagation. In *Proc. of ACL 1990*, p. 31–38.

MEURERS W. D. (2007). Advancing linguistics between the extremes : Some thoughts on geoffrey sampson's *Grammar without Grammaticality*. *Corpus Linguistics and Linguistic Theory*, **3**(1).

MORAWIETZ F. & BLACHE P. (2002). Parsing Natural Languages with CHR.

PAROUBEK P., ROBBA I. & VILNAT A. (2003). EASY : An Evaluation Protocol for Syntactic Parsers. http ://www.limsi.fr/RS2005/chm/lir/lir11/. (as of August 2008).

POLLARD C. & SAG I. (1994). *Head-driven Phrase Structure Grammars*. CSLI, Chicago University Press.

PRINCE A. & SMOLENSKY P. (1993). *Optimality Theory : Constraint Interaction in Generatire Grammar*. Rapport interne, TR-2, Rutgers University.

PULLUM G. & SCHOLZ B. (2001). On the Distinction Between Model-Theoretic and Generative-Enumerative Syntactic Frameworks. In *Proc. of LACL'2001*, number 2099 in LNAI, p. 17–43 : Springer Verlag.

PULLUM G. K. (2007). The Evolution of Model-Theoretic Frameworks in Linguistics. In *Proc. of MTS@10*, p. 1–10.

VANRULLEN T. (2005). *Vers une analyse syntaxique à granularité variable*. PhD thesis, Université de Provence.

VANRULLEN T., BLACHE P. & BALFOURIER J.-M. (2006). Constraint-Based Parsing as an Efficient Solution : Results from the Parsing Evaluation Campaign EASy. In *Proc. of LREC 2006*, p. 165–170.